Kolmogorov Complexity and How it Illuminates our Limitations to Let Machines Learn Simple Functions

by

Lukas J. Rüttgers

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

at the

RWTH AACHEN UNIVERSITY

September 2024

Authored by: Lukas J. Rüttgers
Department of Computer Science
June 30, 2024

Certified by: Jingzhao Zhang
Assistant Professor of Computer Science, Tsinghua University, Thesis Supervisor

Certified by: Hector Geffner
Professor of Computer Science, RWTH Aachen University, Thesis Supervisor

Kolmogorov Complexity and How it Illuminates our Limitations to Let Machines Learn Simple Functions

by

Lukas J. Rüttgers

Submitted to the Department of Computer Science
on June 30, 2024 in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

ABSTRACT

The reasonable maximum demand on out-of-distribution generalization is that the learning algorithm should infer the *simplest* function that remains consistent with the observed data. Kolmogorov complexity intuitively formalizes the notion of simplicity from an information-theoretic viewpoint, as it associates the simplicity of a function with the description length of the shortest program that can compute it. In light of this descriptive complexity, this thesis addresses limitations in our model classes, optimization techniques, and statistical learnability conditions that impede learning simple functions.

Firstly, it demonstrates that models with a *non-recursive structure* such as feed-forward neural networks are incapable of even expressing a class of functions that yet preserve a constantly low Kolmogorov complexity. On the other extreme, models that allowed to express *any partial computable function* would render learnability from finite datasets practically impossible with standard learning algorithms. This is because present learnability conditions in theory either restrict themselves to exemplary model or hypothesis classes [1], [2], or rely on overly conservative assumptions such as the ubiquitous *i.i.d.* assumption [3], [4].

In this setting of overarching model expressivity, this work instead substantiates how incorporating Kolmogorov complexity as a simplicity bias into the optimization objective function carves the way to formulate general distribution-free, both necessary and sufficient information-theoretic conditions that a dataset must satisfy to learn *any* partial computable function, and exemplifies how such an inductive bias can further reduce the sample size that is usually required for classical learnability guarantees. To that end, it proposes to directly draw upon Kolmogorov complexity to quantify the information that finite datasets convey about the functions that could have possibly generated it.

Because Kolmogorov complexity is incomputable in general, compression algorithms are typically proposed as viable approximations [5]. However, for *any* choice on the encoding of Turing Machines, contemporarily employed compression algorithms such as the Lempel-Ziv-Welch compression can not even yield approximate guarantees about the order between the Kolmogorov complexity of two binary strings, because such compression algorithms do not keep up with the compression power of Turing Machines.

Thesis supervisor: Jingzhao Zhang
Title: Assistant Professor of Computer Science, Tsinghua University

Thesis supervisor: Hector Geffner
Title: Professor of Computer Science, RWTH Aachen University

Contents

List of Figures

Chapter 1

Introduction

Think back to the moment where you and your classmates were taught elementary arithmetic on numbers. Teachers across the world suggest their students to represent numbers in a particular numeral system, the decimal system. In this vein, numbers are nothing but a sequence of symbols $a \in \{0, 1, \ldots, 9\}$, and elementary arithmetic operations such as addition or multiplication can be briefly described as to inductively apply some simple symbol transformations over the symbol sequence. Although the teacher gave you merely a few examples for small numbers, they contained sufficient information to describe this inductive algorithm, and enabled you to reproduce this algorithm for numbers beyond any witnessed scale.

The expedient lessons of this exemplary memory are twofold. Firstly, it demonstrates that recursion is a simple, yet powerful mechanism to express unknown concepts with more fundamental ones and to generalize these concepts to unknown problem instances. Secondly, it signifies that this generalization ability of humans rests on some kind of inductive bias that favours inferring *simpler* concepts over more difficult ones. That way, the amount of information that the teacher needed to convey in terms of definitions and examples to let its students entirely comprehend the concept is related to some notion of the *simplicity* of this concept.

1.1 Purpose and contributions

In the setting of supervised learning, we also provide a learning algorithm examples in terms of functional mappings (x, y) to eventually guarantee that it must detect the correct function that produces these mappings.

This thesis unravels how such a simplicity bias that draws on Kolmogorov complexity could enable the learnability of any computable function with finite information-theoretic resources, and even allows to undercut the sample sizes that are usually required to learn such functions in other learnability frameworks, such as the PAC model. At the same time, it points up limitations in our models, learning algorithms, and learnability conditions that hinder us from learning simple functions. Using the example of certain recursive functions, we[1] first demonstrate how feed-forward neural networks and any other model with a *non-*

[1] If the author uses the pronoun "we" in the subsequent work, he jointly refers to the readership and himself.

recursive structure can not even express functions that are however simple to describe given what functions — activation functions, arithmetic operations, constants, logical expressions, etc. — they have access to. These restraints of their representation class hence bias *against* some simple functions. On the other hand, learnability guarantees such as classical ones that are based on the Rademacher complexity or VC dimension of hypothesis classes rest on some limitations of the hypothesis class. By quantifying the information in a dataset about the underlying function in terms of Kolmogorov complexity, such restraints however disappear, as the hypothesis class may now admit *any* partial computable function and still render finite information sufficient to learn the true function. Since Kolmogorov complexity K is incomputable, this thesis finally touches on viable approximations of this quantity. Although compression algorithms are frequently proposed in practice to estimate the simplicity within data, this thesis corroborates that any of the contemporary compression algorithms are unable to compress simple regularities in strings v to an arbitrary scale, and therefore can not even yield approximate theoretical guarantees on the *order* between the Kolmogorov complexity of two strings, $K(v) < K(w)$, even if we allow for any exponential, multiplicative, and additive approximation offsets.

1.2 Content structure

To begin with, Chapter 2 provides essential preliminaries and contextualizes our work in light of prior research. While Section 2.1 presents pertinent work in the field of domain generalization and indicates remaining intricate problems, Section 2.2 provides a comprehensive introduction to Kolmogorov complexity.

Subsequent to that, Chapter 3 addresses the expressive limitations of non-recursive models and the discriminative flaws of optimization objectives. To that end, Section 3.1 first rigorously defines *non-recursive models*. Then, Section 3.2 introduces a class of recursive, simple functions that such models cannot express. While the inexpressivity result is proven in Section 3.3, Section 3.4 shows that these functions have a constantly low Kolmogorov complexity given the functions these models have access to. Finally, Section 3.5 demonstrates that infinitely many functions that could be expressed by these non-recursive models achieve the optimal score according to standard optimization objectives like Empirical Risk Minimization or even Invariant Risk Minimization, because these models could essentially memorize the training data. Thereon, it exemplifies how the discriminative capability of Kolmogorov complexity renders all these functions suboptimal for sufficient large finite datasets, and hence allows to learn the true recursive function.

Generalizing this insight, Chapter 4 formulates sufficient information-theoretic conditions that allow to learn the overarching hypothesis class of computable functions, but still remain optimal, in the sense that less information would in general not allow to learn the respective function. After Section 4.1 exemplifies that it is neither possible nor meaningful to merely condition learnability of computable functions on the number of samples in the dataset, Section 4.2 alternatively elaborates on quantifying the information that datasets convey about the functions that could have generated it in terms of Kolmogorov complexity. To that effect, it substantiates that weaker formulations of this information based on Kolmogorov complexity violate desirable properties, and thereby concludes with the ultimate

formulation, coined *functional information*. Thereafter, it elucidates how this functional information smooths the way to the aforementioned learnability conditions, and juxtaposes them to prior works in learnability and compression. In particular, it exemplifies on the hypothesis class of d-dimensional parity functions that the additional discriminative power of the incorporated simplicity bias allows to learn parity functions with less than d samples, given their Kolmogorov complexity is relatively small. Finally, Section 4.3 illustrates how an oracle for functional information could be used to realise a simplicity bias in practice.

Although Kolmogorov complexity is incomputable, a simplicity bias merely requires an approximation that could yield some approximate guarantee on the *order* between the Kolmogorov complexity of two strings $K(v) < K(w)$. However, Chapter 5 addresses why compression algorithms, which are frequently proposed as approximations of Kolmogorov complexity, do not even yield such approximate guarantees. For this purpose, Section 5.1 uses the Lempel-Ziv-Welch algorithm as a classical example of compression algorithms to demonstrate their limitations in compressing strings that have been generated by simple algorithms. After corroborating the arbitrary compression power of Turing Machines in Section 5.2, Section 5.3 instantiates that such compression algorithms can not yield such guarantees for any exponential, multiplicative, and additive approximation offsets.

Finally, we summarize the above results and their impact in Chapter 6 and raise future research questions.

Chapter 2

Preliminaries and Related Work

Generalization guarantees in classical learning theory via the Probably Approximately Correct (PAC) Model, uniform convergence, or Rademacher complexity all rely on the assumption that the samples that a model observes during training are identically independently distributed (*i.i.d.*) from an underlying, pristine distribution [6, Chapter 3f]. In practice however, it is infeasible to design data collection procedures that satisfy this assumption. Instead, training datasets naturally incorporate severe biases and consequently only account for a small fraction of the diversity of the entire distribution. The field of *domain generalization* tries to model this distortion of the training distribution and studies what statistical conditions this distribution must yet satisfy to still enable certain models optimized with certain objective functions and techniques to provably learn the overall true function [7]. As these statistical conditions are yet quite strong, scientists in this field are also dedicated to incorporate more inductive biases into models and optimization functions that coincide with our notion of reasonable inference [2][8]. After all, we human beings seem to be able to infer *simple* regularities that generalize particular and quite limited observations. First of all, we introduce the formal setting of domain generalization in Section 2.1, the most prominent optimization objectives it has produced, but also demonstrate their indifference regarding the *simplicity* of the learned representation function.

To establish a formal definition of *simplicity*, Kolmogorov complexity is presented in Section 2.2 as a natural measure of the descriptive simplicity of objects such as functions or strings.

2.1 Domain generalization

Consider some function $f : \mathcal{X} \to \mathcal{Y}$ from spaces $\mathcal{X}, \mathcal{Y}$. When trying to learn this function f given some functional tuples $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$. Throughout this thesis, we refer to (x_i, y_i) as a *sample*, where x_i is an *instance* of the learning problem f and y_i its *label*. Since a candidate hypothesis h is rated by its *expected risk*, we have to assign a probability to each instance x. For that reason, problem instances are usually assumed to originate from a *marginal* probability distribution $x \sim P_{\mathcal{X}}$ over $\mathcal{X}$.

The expected risk therefore writes as

$$R(h) := \mathbb{E}_{x \sim P_{\mathcal{X}}} \left[\ell(h(x), f(x)) \right], \tag{2.1}$$

where ℓ is a *loss function* evaluating the deviation of the estimation $h(x)$ from the ground truth $f(x)$.

However, both the marginal $P_{\mathcal{X}}$ and the true function f are usually impossible to specify exactly. Without further assumptions, the former is best approximated by the empirical distribution $\hat{P}_{\mathcal{X}}$ of training instances.

The expected risk is accordingly estimated by the *empirical risk*

$$\hat{R}(h) := \mathbb{E}_{x \sim \hat{P}_{\mathcal{X}}} \left[\ell(h(x), f(x)) \right] = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i). \tag{2.2}$$

A fundamental problem in learning theory is to bound the expected risk $R(h)$ of some hypothesis h given only its empirical risk $\hat{R}(h)$. If we assume that each instance observed during training is independently drawn from the true marginal $P_{\mathcal{X}}$, $\hat{P}_{\mathcal{X}}$ is an optimal estimate of $P_{\mathcal{X}}$. Upon this i.i.d. assumption, several frameworks in statistical learning theory have decades ago provided bounds on the deviation of $R(h)$ from $\hat{R}(h)$ that become increasingly sharp for growing sample size n.

However, the data-collection procedures that generate the training instances in practice naturally deviate strongly from the randomness by which instances occur according to $P_{\mathcal{X}}$. This is particularly the case for audiovisual data, where measurements are accompanied with noise and context-specific superfluous information that is in general yet unrelated with the function f. In such settings where the *i.i.d.* assumption is profoundly violated, a small empirical risk on the training dataset does not necessarily guarantee a small expected risk. Of course, dropping the *i.i.d.* assumption without replacement renders any general guarantee on the expected risk impossible, since $\hat{P}_{\mathcal{X}}$ could now be arbitrarily unrepresentative of $P_{\mathcal{X}}$.

Instead, the *domain generalization* literature identifies the aforementioned superfluous information with *measurement conditions* ε [7]. These measurement conditions incorporate anything that sufficiently determines the information in a measured instance x that does generally not relate to the output $y = f(x)$. For images, such measurement conditions include lighting conditions, color encoding, resolution, filters, noise, and unrelated artifacts such as the background behind a measured object.

Now, the instances observed during training stem from a distribution P_ε that in turn originates from a parametrized family of distributions $\mathcal{P}_{\mathcal{E}} := \{P_\varepsilon \mid \varepsilon \in \mathcal{E}\}$, where $\mathcal{E}$ is the set of all possible measurement conditions. The objective is to derive a hypothesis that achieves a minimal risk in *all* distributions $P_\varepsilon \in \mathcal{P}_{\mathcal{E}}$.

2.1.1 Invariant Risk Minimization

One of the most prominent publications in this domain is Invariant Risk Minimization (IRM) [2], which extends the ubiquitous Empirical Risk Minimization (ERM) principle [9] by invariance constraints.

Given multiple training distributions $\mathcal{P}_{tr} := \{P_1, \ldots, P_j\}$ with functional tuples (x, y), $x \in \mathbb{R}^d$, and a model with a class of realizable hypotheses $\mathcal{H}$, a single hypothesis $h \in \mathcal{H}$ is learned for all distributions $P_\varepsilon \in \mathcal{P}_{\mathcal{E}}$. This hypothesis is composed of a representation function $\Phi : \mathbb{R}^d \to \mathbb{R}^r$ and a predictor $w : \mathbb{R}^r \to \mathbb{R}^k$. For that reason, we denote $h =$

$(w \circ \Phi)$. Moreover, we distinguish the random variables $X \sim P_{\mathcal{X}_\varepsilon}, Y = f(X)$ by the specific realisations $x \in \mathcal{X}_\varepsilon, y = f(x)$ of draws from this distribution $P_{\mathcal{X}_\varepsilon}$.

As the presence of multiple distributions furnishes a fit occasion for concepts of *causality*, the authors assume that the function f incorporates an *invariant causal mechanism* as in [10, Section 4] that describes a fixed causal relationship between the label y and some information in x that is always extractable from any instance x drawn from any distribution $P_\varepsilon \in \mathcal{P}_\mathcal{E}$. The representation Φ should merely encode information that has a high predictive power of the true label $y = f(x)$ in all training distributions P_i and could therefore belong to this invariant mechanism in f. Accordingly, the authors refer to this essential piece of information as the *invariant* information x_{inv}.

However, there is also information that might correlate with the label $y = f(x)$ in some training distributions, but fails to render useful for prediction in some other training distributions. By the assumption of an invariant causal mechanism f, this piece of information shall be ignored by the representation learned by h. The authors coin the attribute *spurious* to describe such information x_{spu}.

To force h to effectively restrict itself to information that yields an invariantly optimal predictive power, the authors introduced IRM, an optimization objective that constraints the predictor w in h to achieve the lowest possible risk R_i in *all* training distributions $P_i \in \mathcal{P}_{tr}$ given the information in Φ. If that was not the case, then there are two possible scenarios. Either, there is a predictor w' that performs better in all distributions and thus shall be preferred over w. Otherwise, there is another w' that only surpasses w's accuracy in *some* training distributions. For these training distributions P_i, this implies that there is some spurious information in $X \sim P_i$ that is preserved by the representation $\Phi(X)$ which w' again utilizes to gain an edge over w. As this information does not belong to the invariant causal mechanism f, it shall therefore rightfully be omitted by Φ. A representation Φ' that would not include this spurious information anymore would render the risk of $w \circ \Phi'$ at least as low as that of $w' \circ \Phi'$. While the first case concerns the optimization over the predictor w given Φ, the latter case affects the representation function Φ itself.

The optimization objective IRM is therefore formally written as

$$\min_{(w \circ \Phi) \in \mathcal{H}} \sum_{i=1}^{j} R_i(w \circ \Phi) \quad \text{such that} \quad w \in \arg\min_{w} R_i(w \circ \Phi) \text{ for all } 1 \leq i \leq j. \tag{2.3}$$

Indeed, the IRM objective might successfully avoid to integrate noise or spurious features that only occur in some of the training distributions into the representation Φ. However, for such spurious features to be excluded, they must increase the predictive power w in some, but not all training distributions.

2.1.2 Invariant spurious information

It therefore still lacks a fundamental problem that was already partially addressed by [1], [11], [12]. Spurious features that do not increase the predictive power of w in any training distribution but *aggravate* the predictive power outside of the training distributions are not excluded by Φ. If these spurious features introduce additional noise and consequently increase the entropy of the distribution of $\Phi(X)$ for some training environment $X \sim P_i$,

then the extension of IRM by an additional entropy penalty on the representation would encourage Φ to exclude any such features if they do not increase the predictive power in every training distribution. They call this extended objective *Information Bottleneck IRM*, short IB-IRM.

In the following, we adopt the notation of the authors and define

$$\operatorname{supp}(P) := \{x \in \mathcal{X} \mid P(X = x) > 0\} \tag{2.4}$$

as the set of all elements with a non-zero probability according to some distribution P over $\mathcal{X}$. This set is commonly referred to as the *support* of P. For the specific class of noise-distorted, linear classifiers f they assume in their formal setup, x_{inv} and x_{spu} are retrievable by a linear transformation of x. Consequently, there exists a matrix A_{inv} such that, $x_{inv} = A_{inv}x$.

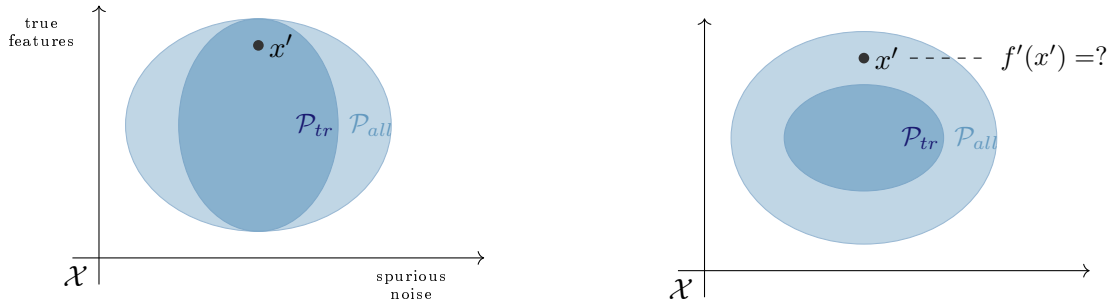
With this setup at hand, they prove that information-theoretically, the global optimum of IB-IRM coincides with the true classifier f if

$$\cup_{\varepsilon \in \mathcal{E}} A_{inv} \operatorname{supp}(P_\varepsilon) \subset \cup_{1 \leq i \leq k} A_{inv} \operatorname{supp}(P_i), \tag{2.5}$$

where the multiplication of a matrix A with a set of vectors Σ is understood as the set of matrix-vector products $\{Ax \mid x \in \Sigma\}$. In more natural language, they assume that the training distributions must already include any realisation of invariant information x_{inv} that could occur in the entire distribution family $\mathcal{P}_\mathcal{E}$.

To support the necessity of this strong assumption, they state in their Theorem 2 that for any $\mathcal{P}_{tr}$ that does not satisfy this condition, no deterministic algorithm will always identify the correct linear classifier f. The reason is that for linearly separable classes, there could always be another linear classifier f' that could have generated the same training data but disagrees with f on some instance outside out of $\mathcal{P}_{tr}$. As any deterministic algorithm has to decide for at most one of f or f', it will certainly fail to generalize out-of-distribution for the other one. On the first sight, this argument seems to annihilate any hope to attain out-of-distribution generalization with means beyond an overarching training dataset. However, this notion of out-of-distribution generalization that was proved impossible in the aforementioned work excels the maximum demand that we could reasonably pose to out-of-distribution generalization. Without any additional prior knowledge, the most reasonable inference one could make given some observations is arguably to assume the *simplest* underlying pattern that could have generated these observations [13]. Of course, in general there are infinitely many functions that remain consistent with any finite training dataset, but most of them do not identify simple concepts.

In Chapter 4, this work hence considers how much limited information is sufficient to ensure that the simplest function that remains consistent with this information indeed coincides with the true function. By this means, this work proposes to accompany our definition of out-of-distribution generalization with an inductive bias that resolves the additional degrees of freedom outside of the training distribution support, and showcases that a simplicity bias by virtue of Kolmogorov complexity realises a desirable inductive bias that favours simplicity. Figure 2.1 juxtaposes this new setting with the commonly adopted setting in prior domain generalization literature. On the one hand, Figure 2.1a illustrates the separation of the information into true and spurious information. While the former is exhaustively covered by the training distribution support, the variation in the latter is only observed in a limited



(a) Commonly studied overlap assumption [1] (b) The more general setting of this work.

Figure 2.1: While prior work in domain generalization such as [1] considers scenarios where the range of true functions is entirely available during training and focus on the problem how the representation shall exclude noise or other features that exhibit spurious correlations with the label and hence ostensibly raise the predictive power, this work considers what information about the true features themselves is necessary and sufficient to correctly infer the true function as the *simplest* function that remains consistent with the information.

range, and the objective is to ensure that the predictor shall remain robust against possible deviations in these spurious features during testing or deployment. On the other hand, our setting in Figure 2.1b rests on a simplicity bias to determine how the behaviour of a predictor f' on instances x' outside of the training distribution support shall be determined. In this way, it reconciles our demands on out-of-distribution generalization with the limited information available in practice.

On the contrary, optimization objectives like IRM or IB-IRM are however indifferent about the simplicity of an hypothesis h, as they only evaluate h within $\mathcal{P}_{tr}$, and assign the optimal score to h as long as these in-distribution optimality conditions are satisfied. In fact, the mere constraints that control how h behaves outside of $\mathcal{P}_{tr}$ are imposed by the choice of the hypothesis class $\mathcal{H}$. However, this work argues that commonly employed model classes can not represent an important body of simple functions and instead restrict themselves to more complex ones, resulting in unreasonable out-of-distribution behaviour.

Take the Multilayer Perceptron (MLP) [14] as an example.

When endowed with the omnipresent ReLU activation function $ReLU(x) = \max(0, x)$, the MLP will fit the data-generating function f within the training distribution by a function h of piece-wise linear interpolations.

If samples were actually drawn *i.i.d.* from an underlying distribution, we did not have to care for what happens outside the training distribution, as MLPs are evidentially universal function approximators when provided with a sufficiently large hidden layer width or appropriate activation functions [15] [16].

In practice however, where training and deployment distributions differ, h will quickly converge to a linear function outside, irrespective of what pattern h was imitating inside the training distribution [17].

In the infinite hidden layer width limit, Xu et al. even formally proved that this convergence occurs with a linear rate outside of the training distribution support [18]. Based on the

example of Graph Neural Networks and Dynamic Programming problems, they further argued that the generalization ability of models is largely determined by how well task-specific non-linearities are encoded into the model class.

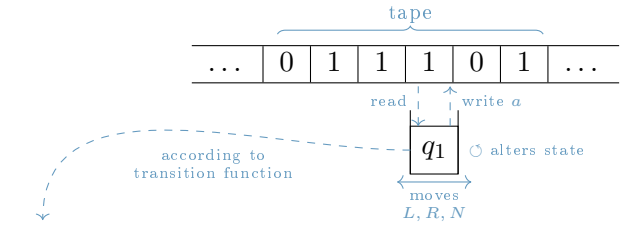
Generalization is therefore still not achieved by inference principles that favour simplicity, but mainly engineered ab initio by model designers. To substantiate this insight with formal arguments, we first require a definition of the *simplicity* of a function. This is subject of the next section, which introduces Kolmogorov complexity as an rigorous measure for this informal notion. By and large, the Kolmogorov complexity of a function f considers how much information is needed to describe the simplest algorithm that produces the same output as f. Although an entire theory of inductive inference was developed around this complexity [13], its incomputable quantities are hard to incorporate into models and viable learning algorithms [19]; with the consequences that are pointed up in the subsequent sections.

2.2 Kolmogorov Complexity

The Turing Machine is a formal description scheme for algorithms that was invented to study what are the functions that are computable. The finding that the expressive power of this computational framework coincides with that of other contemporary formalisms of computability led to the nowadays widely accepted Church-Turing Thesis [20][21].

It states that *effectively calculable* functions — those functions that we intuitively believe to be computable by some algorithm — are also computable by the class of Turing Machines [22].

Formally, a Turing Machine is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, B, q_0, q_{halt}\delta)$. It receives its input x as a word over an alphabet Σ, which is a finite set of symbols. In this work, we will always stick to the archetypical, binary alphabet $\Sigma = \{0, 1\}$. The input word x is provided to $\mathcal{T}$ on a working tape of infinite cells, which can be understood as an infinite amount of working memory that $\mathcal{T}$ can utilize. Each cell comprises exactly one symbol from the tape alphabet $\Gamma = \Sigma \cup \{B\}$, where $B \notin \Sigma$ is the blank symbol that indicates unused tape cells. The expressive power of Turing Machines does not change if we allow them access to multiple tapes, and we will adopt the convention that $\mathcal{T}$ always has a dedicated input and output tape. The deterministic algorithmic behaviour of $\mathcal{T}$ is described in its *transition function* $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$. Endowed with a tape head, $\mathcal{T}$ can access only one cell at once. The internal states $q \in Q$ hence contain contextual information that the Turing Machine must store to interpret the symbol in the current cell in the correct way. Figure 2.2a illustrates this algorithmic behaviour. Depending on the internal state q_i it is situated into and the symbol a it reads, $\delta(q, a) = (q_{next}, a', \text{dir})$ determines to what internal state q_{next} to transition, what symbol a' to write, and into what direction dir — left, right, or none — to move the tape head. Figure 2.2b exemplifies such a transition function for the modulo function mod_2. The Turing Machine runs over the input string and determines in which modulo class the string falls. To that end, it must memorize the last symbol, which requires two states q_0 and q_1. By convention, q_0 identifies the initial state, and q_{halt} the final state that indicates that $\mathcal{T}$ halts. Similarly, a Turing Machine that computed the mod_3 or mod_5 function required three or five states respectively.



δ	0	1	B
q_0			
q_1		$(q_{next}, a, \text{dir})$	
$\vdots$			
q_n			

(a) A general conceptualization of a Turing Machine.

δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	$(q_{halt}, 0, N)$
q_1	(q_0, B, R)	(q_1, B, R)	$(q_{halt}, 1, N)$

(b) A Turing Machine that computes the modulo function mod_2.

Figure 2.2: Illustration of the algorithmic behaviour of a Turing Machine.

Functionally, $\mathcal{T}$ coincides with a *partial computable* function

$$f_{\mathcal{T}}(x) := \begin{cases} y, & \mathcal{T} \text{ halts and outputs } y, \\ \perp, & \mathcal{T} \text{ does not halt}, \end{cases} \tag{2.6}$$

where $\perp$ means that the output is undefined. If $\mathcal{T}$ halts on all inputs, we call f *total computable*.

As there are only countably infinitely many distinct Turing Machines, each of them can be uniquely associated with a binary string, which is accordingly called the *Gödel number* of a Turing Machine. A suitable approach to obtain such a Gödel numbering is to directly encode the transition function δ of a Turing Machine $\mathcal{T}$.

Besides merely identifying $\mathcal{T}$, δ *describes* the algorithm that $\mathcal{T}$ computes. An encoding of δ with a computable inversion thereby allows to fully reconstruct $\mathcal{T}$ in a uniform algorithmic manner. For any such encoding $\text{enc} : \mathcal{T} \mapsto \text{enc}(\mathcal{T})$, there is a *universal Turing Machine* U that given input $\text{enc}(\mathcal{T}), x$ can simulate $\mathcal{T}$ on input x. In this manner, U computes the partial computable function

$$f_U(x) = \begin{cases} f_{\mathcal{T}}(z), & x = \text{enc}(\mathcal{T})z \\ \perp, & \text{otherwise.} \end{cases} \tag{2.7}$$

To enable U to unambiguously separate the concatenated string x into $\text{enc}(\mathcal{T})$ and z, it proves useful to assume that the encoding enc is *prefix-free*. This means that for any two Turing Machines $\mathcal{T}, \mathcal{T}'$, the string $\text{enc}(\mathcal{T})$ is not a prefix of $\text{enc}(\mathcal{T}')$.

To further allow the unambiguous separation of arbitrary strings $v, w \in \{0,1\}^*$, $[v, w]$ denotes the *self-delimiting encoding* of strings v and w that precedes their concatenation with a string $1^{l(v)}0$ that encodes the length $l(v)$ of v as

$$[v, w] = 1^{l(v)}0vw. \tag{2.8}$$

The aforementioned encodings of δ usually have in common that the length of the encoding scales with the number of internal states $q \in Q$. Consequentially, such encodings provide a natural viewpoint on the information or *descriptive complexity* that is innate to a partial computable function [23]. This notion associates the amount of information that is needed to unambiguously describe a partial computable function f with the shortest encoding $\text{enc}(\mathcal{T})$ of a Turing Machine $\mathcal{T}$ such that $f \equiv f_{\mathcal{T}}$, where equivalence $f \equiv g$ means that f and g are defined on the same definition range $D \subset \{0,1\}^*$ and satisfy $f(x) = g(x)$ for all $x \in D$.

Commemorating one of the pioneers in this field, this quantity is referred to as *Kolmogorov complexity*.

Definition 2.1 (Kolmogorov complexity)
Let f be an arbitrary partial computable function with definition range $D_f \subset \{0,1\}^*$. The Kolmogorov complexity is defined as

$$K_U(f) := \min_{p \in \{0,1\}^*} \{l(p) \mid U(px) = f(x) \text{ for all } x \in D_f \tag{2.9}$$

$$\text{and } U(px) \text{ does not halt for any } x \in \{0,1\}^* \setminus D_f\}. \tag{2.10}$$

In the simplest setting, p is just the string that precedes x with the prefix-free encoding of some Turing machine $\mathcal{T}$, $px = \text{enc}(\mathcal{T})x$. But p could also contain additional parameters y for $\mathcal{T}$, for example $p = \text{enc}(\mathcal{T})[y, x]$.

The learning problems we consider in this work are functions over $\mathbb{N}$. Therefore, we adopt the following convention to unambiguously relate natural numbers to binary strings.

Definition 2.2 (Bijective binary encoding of natural numbers)
For any natural number $n \in \mathbb{N}$, we define $x_n \in \{0,1\}^*$ as the binary string that encodes n in the binary numeral system without leading zeros. 0 is encoded by the empty string ε, 2 by 10, 5 by 101, and so forth. Conversely, if $x \in \{0,1\}^*$ encodes a natural number in this way, we denote this number by n_x.

For cases where we want to separate pieces of information, it further renders useful to consider a *conditional* Kolmogorov complexity.

Definition 2.3 (Conditional Kolmogorov complexity)
Let f be an arbitrary partial computable function with definition range $D_f \subset \{0,1\}^*$. The conditional Kolmogorov complexity is defined as

$$K_U(f \mid z) := \min_{p,p' \in \{0,1\}^*} \{l(p) + l(p') \mid U(p[z, p'x]) = f(x) \text{ for all } x \in D_f \tag{2.11}$$

$$\text{and } U(p[z, p'x]) \text{ does not halt for any } x \in \{0,1\}^* \setminus D_f\}. \tag{2.12}$$

which we define as In the definition of the conditional Kolmogorov complexity of *strings* in [23] merely contains the prefix string p and not the suffix string p'. This restriction was sufficient for their needs as they usually employed this quantity to study the Kolmogorov complexity of strings given their length. Our more general definition allows to condition on pieces of information in *any* part of the string that is provided to U, not only its suffix. While easing the technical construction of Turing Machines, our definition remains equivalent to the restricted definition in [23] up to an additive constant because the piece of information

in question could just be moved to the back of the input string by another Turing Machine that parses the input.

Admittedly, the Kolmogorov complexity of a function depends on the underlying universal Turing Machine U and therefore also the encoding $\text{enc}(\cdot)$. The choice of $\text{enc}(\cdot)$ already entails an inductive bias about what objects are attributed a relatively low and high Kolmogorov complexity. However, any universal Turing Machine U' that corresponds to a different encoding enc' can be described by U by its respective Gödel number. Since invoking U on $\text{enc}(U')\,\text{enc}'(\mathcal{T})x$ produces the same result as directly invoking U' on $\text{enc}'(\mathcal{T})x$, the Kolmogorov complexities K_U,$K_{U'}$ can only differ by an additive constant that is determined by the length of $\text{enc}(U')$. This insight was stated in the Invariance Theorem [23, Section 2.1].

Theorem 2.4 (Invariance Theorem)
For any universal Turing Machines U, U', there is a constant $c_{U,U'}$ such that for any partial computable function f,

$$|K_U(f) - K_{U'}(f)| \leq c_{U,U'}. \tag{2.13}$$

For that reason, we will in the following replace $K(\cdot) = K_U(\cdot)$ and $K(\cdot \mid y) = K_U(\cdot \mid y)$.

When characterizing algorithms with short descriptions that however maintain expressive power, recursion certainly constitutes an essential principle. On the one hand, the expressive power of recursion was precisely demonstrated by the class of *general recursive functions*, whose inductive definition apart from concatenation and minimization also avails itself of recursion [24]. This class was one of the contemporaneous models of computability that were later proved to be equivalent to Turing Machines [21]. On the other hand, recursive operations comprise regularities with short description and therefore usually require little information to describe.

Using the example of such simple recursive functions, the next chapter will demonstrate shortcomings within both the structure of models and the constraints of optimization objectives that impede learning simple functions.

Chapter 3

Models and Optimization that are Mindless of Simplicity

In this chapter, we consider learning problems of the form $h : \mathbb{N}^k \to \mathbb{N}$ for some arbitrary $k \in \mathbb{N}$. To cover the entire range of signed, finite precision floating point numbers in computing architectures and thus all functions that neural network can express in practice, we will later show how our results extend from $\mathbb{N}$ to $\mathbb{Z}$. Figure 3.1 provides a holistic roadmap for the line of reasoning that this chapter trails, which we now explain in more detail. When we construct

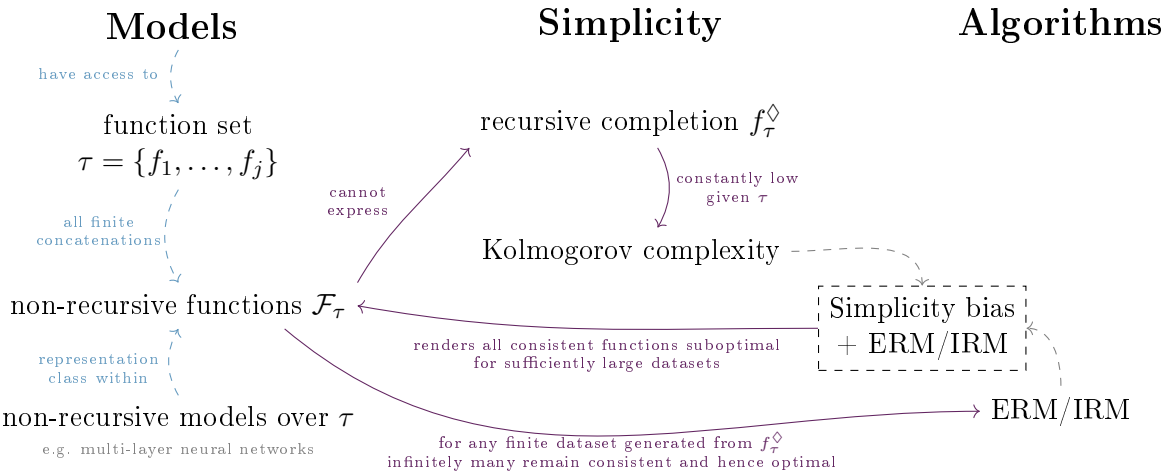


Figure 3.1: Holistic overview of the line of reasoning in Chapter 3.

mathematical models to learn predictors for such functions h, there is an underlying finite set of operations or functions $\tau = \{f_1, \ldots, f_j\}$ that serve as *building blocks* of our model. In practice, these are the functions we eventually use in our source code to express a model in a certain programming language. For neural networks, typical building block functions include arithmetic operations like addition or multiplication, activation functions like the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, ReLU, ELU, or tanh, constants c, and logical expressions like the equality or inequality $=, <$, and logical conjunctions such as $\wedge$ and $\vee$.

In the case of feed-forward neural networks M_θ, our building block functions f_i are merely concatenated in a way that does not depend on the input x. In particular, the number of layers and the values of the parameters are fixed and do not change according to the input x. More formally, the function f_θ that M_θ computes is equivalent to a finite concatenation of the building block functions $f_1, \ldots, f_j$. Such multi-layer neural networks are well-known instances of a model class that we will later rigorously define as *non-recursive models*. Informally, non-recursive models over τ can merely express finite concatenations of the building block functions in τ.

The demonstrations of this chapter are twofold. Firstly, this chapter puts forward that there are simple functions that these non-recursive models are not able to express. To that end, Theorem 3.8 states that models with a non-recursive structure always have a certain class of recursive functions h_τ they are incapable of even expressing, no matter what building block function set τ they were given access to, as long as the functions in τ satisfy conditions that frequently hold in practice. At the same time, Theorem 3.9 asserts the functions h_τ of this very class preserve a constantly low Kolmogorov complexity given τ, which allows us to conclude the aforementioned statement.

On the other hand, the ability of standard optimization objectives to learn the true function also relies on limitations of the model class. In the limit of expressive power where *any* total computable function could be expressed by a model, any finite dataset would have infinitely many functions that remain consistent with it, hindering even optimization objectives that are dedicated to domain generalization like IRM from identifying the true function.

Using the example of the aforementioned class of recursive functions h_τ, Theorem 3.11 however illustrates how incorporating Kolmogorov complexity as a simplicity bias into the optimization objective would ensure that for sufficiently large datasets, every function that a corresponding non-recursive model could possibly express with τ obtains a higher Kolmogorov complexity than the true function h_τ.

By exemplifying the discriminative power of this simplicity bias, these results therefore encourage to study how Kolmogorov complexity paves the way to general, yet nearly optimal learnability conditions for hypothesis classes as large as the class of all partial computable functions, which we undertake later in Chapter 4.

3.1 Non-recursive models

On our way to provide a rigorous definition of non-recursive models, we first characterize what it means for a function to be non-recursive.

Definition 3.1 (Non-recursive functions)
Given a finite set of functions $\tau = \{f_1, \ldots, f_j\}$ such that for each $1 \leq i \leq j$, $f_i : \mathbb{N}^{k_i} \to \mathbb{N}$ is a k_i-ary function over $\mathbb{N}$ for some $k_i \in \mathbb{N}$.

We inductively define the set of non-recursive functions over τ, denoted $\mathcal{F}_\tau$.

Beforehand, take $\mathrm{VAR} := \{n_i \mid i \in \mathbb{N}\}$ *as the set of variable symbols that functions in $\mathcal{F}_\tau$ can take as distinct input variables.*

1. ***Base case:*** *For any variable symbol n_i, the identity function $I \in \mathcal{F}_\tau$, where $I(n_i) = n_i$ for all $n_i \in \mathbb{N}$.*

2. ***Induction step:*** *Let $f \in \tau$ be a function of arity $K \in \mathbb{N}$. Let $g_1, \ldots, g_K \in \mathcal{F}_\tau$ be arbitrary non-recursive functions, where g_i has arity $k_i \in \mathbb{N}$.*

   Let $i_1, \ldots, i_m \in \mathbb{N}$ be the distinct indices of the variable symbols n_i that occur as an input variable symbol in at least one of the functions $g_1, \ldots, g_k$. Abbreviate these m variable symbols as a vector $\mathbf{n} := (n_{i_1}, \ldots, n_{i_m})$. Accordingly, let $\mathbf{n}_i$ be the tuple of the variable symbols $n_{i_j} \in \mathbf{n}$ that occur in g_i, ordered by j.

   Then, the function $h := (f \circ (g_1, \ldots, g_k)) \in \mathcal{F}_\tau$, where $h(\mathbf{n}) := f(g_1(\mathbf{n}_1), \ldots, g_K(\mathbf{n}_K))$ for all $\mathbf{n} \in \mathbb{N}^m$.

Non-recursive functions over τ are hence nothing but finite concatenations of functions in τ. For example, if the function set τ consists out of the constants $f_0(n_i) = 0, f_1(n_i) = 1$, and the addition function $f_+(n_i, n_j) = n_i + n_j$, the non-recursive functions over τ are nothing but the class of linear functions over $\mathbb{N}$, $\mathcal{F}_\tau = \{f : f(n) = an + b \mid a, b \in \mathbb{N}\}$.

As we study the expressive power of models, it suffices to identify models with the representation class $\mathcal{M}$, that contains all functions they can realize. Therefore, we call a model non-recursive on τ if its representation class $\mathcal{M} \subset \mathcal{F}_\tau$. In other words, if a model is non-recursive on τ, any function g that this model can realize is equivalent to a certain finite concatenation of functions in τ.

The inductive definition of $\mathcal{F}_\tau$ furnishes a fit occasion to prove statements over this function class in an analogously inductive manner. For that reason, we establish the notion of the *depth* of a non-recursive function as its maximum concatenation depth.

Definition 3.2 (Depth of non-recursive functions)
We inductively define the depth of non-recursive functions $\mathrm{dep} : \mathcal{F}_\tau \to \mathbb{N}$ *as follows:*

1. ***Base case:*** *The identity function I has depth* $\mathrm{dep}(I) := 0$.

2. ***Induction step:*** *Let $f \in \tau$ be a function of arity $K \in \mathbb{N}$. Let $g_1, \ldots, g_K \in \mathcal{F}_\tau$ be arbitrary non-recursive functions. Then, the function $h := (f \circ (g_1, \ldots, g_k))$ has depth* $\mathrm{dep}(h) := 1 + \max_{1 \leq i \leq k} \mathrm{dep}(g_i)$.

Moving on, the monotonicity of non-recursive functions is defined in the standard way.

Definition 3.3 (Monotonous non-recursive functions)
Let $f \in \mathcal{F}_\tau$ be an arbitrary k-ary non-recursive function with $k \geq 1$.

We call f monotonously increasing if for any $1 \leq i \leq k$, for any $n_1, \ldots, n_k \in \mathbb{N}$, and any $a, b \in \mathbb{N}$ with $a < b$, it holds that

$$f(n_1, \ldots, n_{i-1}, a, n_{i+1}, \ldots, n_k) \leq f(n_1, \ldots, n_{i-1}, b, n_{i+1}, \ldots, n_k).$$

If the inequality holds strictly, we call f strictly monotonously increasing.
The definition of (strictly) monotonously decreasing functions is symmetric.

3.2 Recursive completion

Non-recursive functions as by Definition 3.1 have a concatenation structure that is uniform across all inputs. Conversely, the concatenation structure of recursive functions might certainly depend on the input.

To formally exemplify recursive functions and contrast them to non-recursive ones, we first introduce the *recursive concatenation* of an arbitrary function, which is nothing but the concatenation of a function with itself.

Definition 3.4 (Recursive concatenation)
Let $f : \mathbb{N}^k \to \mathbb{N}$ be an arbitrary function over $\mathbb{N}$ with arity $k \geq 1$. First, denote by $f^-(n) := f(\underbrace{n, \ldots, n}_{k \text{ times}}), n \in \mathbb{N}$ the reduction of f to a unary function that copies the single input n to all k input variables.

We inductively define the m-wise recursive concatenation of f as

$$f^{(0)}(n) := n, \qquad n \in \mathbb{N} \tag{3.1}$$

$$f^{(m+1)}(n) := f^-(f^{(m)}(n)), \qquad n, m \in \mathbb{N}. \tag{3.2}$$

Using the example of a 2-ary function f, Figure 3.2 visualizes the functional composition of the recursive concatenation $f^{(m)}$ by its term tree. If functions f over $\mathbb{N}$ are strictly

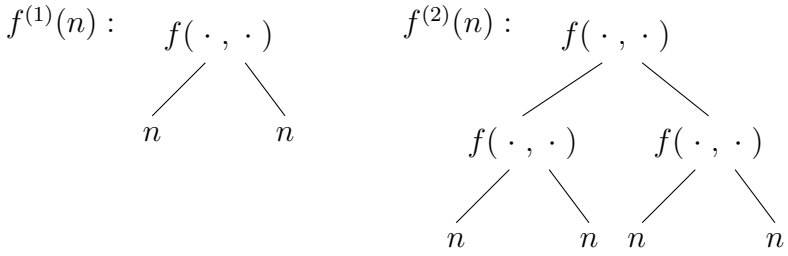


Figure 3.2: Term tree of the recursive concatenation of a 2-ary function $f^{(m)}$ for $m = 1, 2$.

monotonously increasing and have an arity larger than 1, the function values of their recursive concatenations $f^{(m)}(n)$ must grow exponentially fast in the number of concatenations m, as the following Lemma shows.

Lemma 3.5 (Lower bound for recursive concatenation of strictly monotonously increasing functions)
Let $f : \mathbb{N}^k \to \mathbb{N}$ be an arbitrary k-ary function that is strictly monotonously increasing with $k \geq 1$. Let $f^{(m)}$ denote the m-wise recursive concatenation of f as in Definition 3.4.

It holds that $f^{(m)}(n) \geq k^m \cdot n$ for all $n, m \in \mathbb{N}$.

Proof. Fix an arbitrary, strictly monotonously increasing function f of arity k. First of all, we show that $f^-(n) \geq k \cdot n$ for all $n \in \mathbb{N}$.

Exhaustive exploitation of strict monotonicity yields

$$f^-(n) = f(\underbrace{n, \ldots, n}_{k \text{ times}}) \geq f(n-1, n, \ldots, n) + 1 \tag{3.3}$$

$$\overset{\cdots}{\geq} f(0, n, \ldots, n) + n \tag{3.4}$$

$$\overset{\cdots}{\geq} \underbrace{f(0, 0, \ldots, 0)}_{\geq 0} + k \cdot n \geq k \cdot n. \tag{3.5}$$

Now, we prove the general lower bound for $f^{(m)}(n)$ by induction over the number of recursive concatenations $m \in \mathbb{N}$.

The base case $m = 0$ holds by definition since $f^{(0)}(n) = n = k^0 \cdot n$.

For the induction hypothesis (IH), assume that there is some $m \in \mathbb{N}$ such that $f^{(m)}(n) \geq k^m \cdot n$ holds for all $n \in \mathbb{N}$. By the strict monotonicity of f and Equation 3.5, we obtain

$$f^{(m+1)}(n) = f^-(f^{(m)}(n)) \overset{\text{(IH)},(3.5)}{\geq} k \cdot (k^m \cdot n) = k^{m+1} \cdot n. \tag{3.6}$$

Consequently, the overall result follows by the induction principle. □

3.3 Expressive limits of non-recursive models

For the subsequent theorems, we zero in on the recursive concatenation of a specific function, namely the sum over the function set τ, defined as $f_\tau := \left(\sum_{i=1}^{j} f_j\right)$.

To illustrate why it is necessary to take the sum over *all* functions in τ, consider the addition function $f_+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ as an ubiquitous example. If the function set $\tau = \{f_+, f\}$ fortuitously happened to include another function f that is just precisely defined as the recursive concatenation of the addition function $f(m, n) := f_+^{(m)}(n) = 2^m \cdot n$, the recursive concatenation of the addition function alone would be easily expressible by non-recursive models on τ by merely applying f once to the input. But if we instead take the recursive concatenation of their sum $f_\tau(n) = 2n + 2^n n = (2^n + 2) \cdot n$, we can prove that there is no non-recursive function $f \in \mathcal{F}_\tau$ that can keep up with its super-exponential growth, rendering this recursive concatenation inexpressible in $\mathcal{F}_\tau$.

In general however, this inexpressivity statement is hard to prove if we do not make some assumptions on the building block functions $f_i \in \tau$. At the same time, we do not want to restrict τ too much to maintain the pertinence of this result for machine learning models in practice. It turns out that for any function sets τ where each function is either strictly monotonously increasing or bounded, we achieve a sound trade-off between these two conflicting objectives. On the one hand, the formal proofs render quite simple while on the other hand, the representation classes of many machine learning models still fall into this restricted category of function sets $\mathcal{F}_\tau$. Apart from the *ReLU* activation function, most common activation functions such as *ELU*, σ, or tanh are strictly monotonously increasing. At the same time, ubiquitous arithmetic operations like addition or multiplication are strictly monotonously increasing, too. Over and above, learned parameters of a neural network can be represented by constants, which obviously constitute bounded functions. Finally, logical

functions are by definition bounded by 1, and in particular allow to indirectly realise the *ReLU* activation function and other operations that require case distinctions.

For that reason, we restrict the function sets τ to strictly monotonously increasing and bounded functions. As it renders useful in the later theorems, we now show that for any such τ, the recursive concatenation of f_τ is lower bounded by the recursive concatenation of each strictly monotonously increasing function $f_k \in \tau$.

Lemma 3.6 (Lower bound for recursive concatenation of function set sum)
Given an arbitrary function set $\tau = \{f_1, \dots, f_j\}, f_i : \mathbb{N} \to \mathbb{N}$.

For any monotonously increasing function $f_k \in \tau$ *and any* $m, n \in \mathbb{N}$, *it holds that* $f_\tau^{(m)}(n) \geq f_k^{(m)}(n)$.

Proof. Given all as above. We show the statement by induction over $m \in \mathbb{N}$.

For $m = 0$, the result follows from the definition

$$f_\tau^{(0)}(n) = \sum_{i=1}^{j} f_i^{(0)}(n) = \sum_{i=1}^{j} n \geq n = f_k^{(0)}(n).$$

for all $1 \leq k \leq j$ and $n \in \mathbb{N}$.

Now, assume that for some $m \in \mathbb{N}$, the statement $f_\tau^{(m)}(n) \geq f_i^{(m)}(n)$ holds for all monotonously increasing functions $f_i \in \tau$ and all $n \in \mathbb{N}$.

For $m + 1$, any monotonously increasing $f_k \in \tau$ satisfies

$$f_\tau^{(m+1)}(n) = f_\tau^-(f_\tau^{(m)}(n)) = \sum_{i=1}^{j} f_i^-(f_\tau^{(m)}(n)) \geq f_k^-(f_\tau^{(m)}(n)) \tag{3.7}$$

$$\overset{\text{(IH)}}{\geq} f_k^-(f_k^{(m)}(n)) = f_k^{(m+1)}(n), \tag{3.8}$$

where the first inequality draws upon the non-negativity of $\mathbb{N}$. Therefore, the overall statement for all $m \in \mathbb{N}$ follows by induction.

□

In the same spirit as in Definition 3.4, we merge the two variables m, n of the recursive concatenation into one single input n for convenience, and refer to the resulting function as the *recursive completion* of f.

Definition 3.7 (Recursive completion)
Let $f : \mathbb{N}^k \to \mathbb{N}$ *be an arbitrary function with arity* $k \geq 1$.

We define the recursive completion of f, *denoted by* $f^\Diamond : \mathbb{N} \to \mathbb{N}$, *as*

$$f^\Diamond(n) = f^{(n)}(n), \quad n \in \mathbb{N}, \tag{3.9}$$

where $f^{(n)}$ *denotes the n-wise recursive concatenation of* f *as in Definition 3.4.*

The following theorems will be proved for this special case of recursive concatenations. For an arbitrary function set τ that only comprises strictly monotonously increasing and bounded functions, we are going to show that $f_\tau^\Diamond$ — referred to as the *recursive completion over* τ — is not expressible by non-recursive functions over τ, briefly $f_\tau^\Diamond \notin \mathcal{F}_\tau$. This implies that any non-recursive model on τ is not able to express this function.

Theorem 3.8 (Recursive completion over τ is not non-recursively expressible)
Fix an arbitrary function set $\tau = \{f_1, \ldots, f_j\}, f_i : \mathbb{N} \to \mathbb{N}$, where each f_i is strictly monotonously increasing or bounded. Assume that there is at least one $f_i \in \tau$ that is strictly monotonously increasing and has arity $\mathrm{ar}(f_i) > 1$, e.g. the addition function. For any non-recursive function $f \in \mathcal{F}_\tau$, there is an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) \neq f_\tau^{\Diamond}(n)$.

Proof. Fix an arbitrary function set $\tau = \{f_1, \ldots, f_j\}$, where each f_i is strictly monotonously increasing or bounded and at least one $f_i \in \tau$ is strictly monotonously increasing with arity $\mathrm{ar}(f_i) > 1$. Define $f_\tau(n) := \sum_{i=1}^{j} f_i(n)$.

As it facilitates the induction step, we prove a stronger statement by induction over the depth of non-recursive functions $\mathrm{dep}(f) \in \mathbb{N}$. Namely, we are going to prove that for all non-recursive functions $f \in \mathcal{F}_\tau$, there is an $n_0 \in \mathbb{N}$ such that $f(n+a) < f_\tau^{(n)}(n+a)$ for all $n \geq n_0$ and all offsets $a \in \mathbb{N}$.

Beginning with the base case, let $f \in \mathcal{F}_\tau$ be arbitrary with $\mathrm{dep}(f) = 0$. Therefore, f must be equivalent to the identify function, thus $f(n) = n$ for all $n \in \mathbb{N}$.

Choose $n_0 = 1$ and let $n \geq n_0$ and $a \in \mathbb{N}$ be arbitrary.

Since $k = \mathrm{ar}(f_m) > 1$ for at least one strictly monotonously increasing $f_m \in \tau$, Lemma 3.5 guarantees

$$f_m^{(n)}(n+a) \geq k^n \cdot (n+a) \overset{k>1}{>} n + a. \tag{3.10}$$

Since f_m is monotonously increasing, we conclude by means of Lemma 3.6 that

$$f_\tau^{(n)}(n+a) \overset{3.6}{\geq} f_m^{(n)}(n+a) \overset{(3.10)}{>} n + a = f(n+a). \tag{3.11}$$

Proceeding with the induction hypothesis (IH), assume that there is some $p \in \mathbb{N}$ such that for every non-recursive function $f \in \mathcal{F}_\tau$ with $\mathrm{dep}(f) \leq p$, there exists an $n_0 \in \mathbb{N}$ such that $f_\tau^{(n)}(n+a) > f(n+a)$ for all $n \geq n_0$ and $a \in \mathbb{N}$.

Let $a \in \mathbb{N}$ be arbitrary in the following. Consider an arbitrary non-recursive function $f \in \mathcal{F}_\tau$ with $\mathrm{dep}(f) = p + 1$.

As by Definition 3.1, we have $f(n) = f_m(g_1(n), \ldots, g_k(n))$ for some $f_m \in \tau$ with arity $k \in \mathbb{N}$ and non-recursive functions $g_i \in \mathcal{F}_\tau$ with depth $\mathrm{dep}(g_i) \leq p$.

To begin with, we consider the case where f_m is a bounded function. Denote by c_m the bound of f_m. By Equation 3.11, we have

$$f_\tau^{(n)}(n+a) > n + a \geq c_m \geq f_m(g_1(n+a), \ldots, g_k(n+a)) = f(n+a) \tag{3.12}$$

for any $n \geq c_m$.

Otherwise, consider the case where f_m is strictly monotonously increasing. By the induction hypothesis, for every non-recursive function g_i, there is an $n_0(g_i)$ such that $f_\tau^{(n)}(n+a) > g_i(n+a)$ for all $n \geq n_0(g_i)$. Since there are only k non-recursive functions g_i, we directly take n_0 as the maximum $n_0 = \max_{1 \leq i \leq k} n_0(g_i)$.

Now, let $n \geq n_0$ be arbitrary. Subsequently, we assume that $a \geq 1$, and substitute $b = a - 1$ and $n' = n + 1$. By the monotonicity of f_m, we conclude

$$\begin{aligned} f(n'+b) = f(n+a) &= f_m(g_1(n+a), \ldots, g_k(n+a)) && (3.13)\\ &\overset{(IH)}{<} f_m(f_\tau^{(n)}(n+a), \ldots, f_\tau^{(n)}(n+a)) = f_m^-(f_\tau^{(n)}(n+a)) && (3.14)\\ &\leq f_\tau^-(f_\tau^{(n)}(n+a)) = f_\tau^{(n+1)}(n+a) = f_\tau^{(n')}(n'+b). && (3.15) \end{aligned}$$

Since $a \in \mathbb{N}_{\geq 1}$ was chosen arbitrarily, the above argument holds for all $b \in \mathbb{N}$ and all $n' \geq n_0 + 1 =: n_0(f)$. To conclude with, we have shown that for any non-recursive function $f \in \mathcal{F}_\tau$ with $\text{dep}(f) = p + 1$, there exists an $n_0(f) \in \mathbb{N}$ such that $f(n + a) < f_\tau^{(n)}(n + a)$ for all $n \geq n_0(f), a \in \mathbb{N}$.

Consequently, the overall result for non-recursive functions of arbitrary depth draws upon the induction principle.

As a special case of the above result, we obtain that for any non-recursive function $f \in \mathcal{F}_\tau$, there is an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) < f_\tau^\Diamond(n)$.

□

Before we continue, let us make a few comments on the proof of the theorem. In the induction step for non-recursive functions f of depth $p + 1$, the threshold n_0 only increased by 1 when the outermost function f_m was strictly monotonously increasing. Otherwise, if f_m was a bounded function, the threshold n_0 could just be taken as the bound c_m of f_m, irrespective of the depth of f. Therefore, for non-recursive functions f of sufficiently large depth p, the threshold n_0 can be upper bounded by a term $a \cdot p$, where $a \to 1$ for $p \to \infty$.

Figure 3.3 visualizes the eventual dominance of $f_\tau^\Diamond$ over non-recursive functions f. f_τ covers every possible *composition path* that the non-recursive functions in τ could possibly choose. Eventually, n will exceed both the depth of f and the largest constant in f. By the the strict monotonicity of the involved functions, $f_\tau^\Diamond$ will thus excel f.

An extension of this result to $\mathbb{Z}$ allows to apply this result directly to neural networks which usually operate over signed floating point numbers of finite precision. The adjustments we need to make are minimal. On the one hand, the inequality in Equation (3.7) in the proof of the lower bound of the sum $f_\tau := \sum_{i=1}^{j} f_i$ in Lemma 3.6 relied on the non-negativity of each f_i. To generalize this statement to functions over $\mathbb{Z}$, we simply wrap each function f_i by the *magnitude* function before summing them up. Juxtaposing this adjusted construction f_τ to non-recursive functions over τ is still perfectly fair, since both the addition function and the magnitude function will still fall within the scope of functions that still satisfy our assumptions on τ.

On the other hand, the assumptions on the functions in τ are weakened a little to still express a broad range of functions that are used to design neural networks and other models in practice. While the definition of boundedness directly carries over to negative numbers, the condition of strict monotonicity is slightly alleviated.

Arithmetic operations such as multiplication and addition, logical expressions, constants, and activation functions that are usually employed in practice such as ReLU, ELU, tanh, or sigmoid still fall within the scope of these assumptions.

Although the proofs require only slight changes, they yet affect the ease of exposition. For this reason, the argument was presented here only for functions over $\mathbb{N}$, while the more general result for functions over $\mathbb{Z}$ is laid out in Appendix A.3.

This inexpressivity result is preserved if we scale the input n of $f_\tau^\Diamond$ by an arbitrary positive parameter $a \in \mathbb{N}$. The proof is analogous to Theorem 3.8 and therefore deferred to Appendix A.2.

Corollary 3.8.1 (Scaled Recursive Completion is not non-recursively expressible)
Let τ be as in Theorem 3.8.

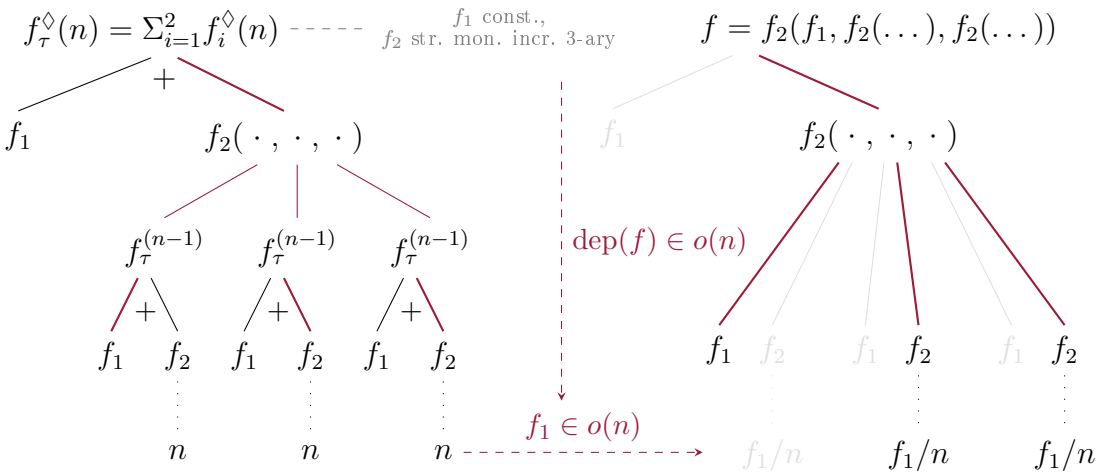


Figure 3.3: Term tree juxtaposing the recursive completion $f_\tau^\Diamond$ and an exemplary non-recursive function $f \in \mathcal{F}_\tau$ of fixed depth. Eventually, n will excel both the depth of f and the largest constant in f, which ensures that $f_\tau^\Diamond(n) > f(n)$.

For any $a \in \mathbb{N}$, define the scaled recursive completion over τ as $(f_\tau)_a^\Diamond(n) := f_\tau^{(n)}(a \cdot n)$.

Then, $(f_\tau)_a^\Diamond \notin \mathcal{F}_\tau$ for any $a \geq 1$. In particular, let $f \in \mathcal{F}_\tau$ be an arbitrary non-recursive function over τ. Then, for the same $n_0(f)$ as in Theorem 3.8, we have that for all $a \in \mathbb{N}$ with $a \geq 1$ and $n \geq n_0$, $f(n) \neq (f_\tau)_a^\Diamond(n)$ for all $n \geq n_0(f)$.

On the one hand, we proved that the recursive completion over τ is inexpressible by $\mathcal{F}_\tau$ under feasible assumptions. On the other hand, it remains to argue that $f_\tau^\Diamond$ maintains a low Kolmogorov complexity when the description of the underlying function set τ is already given. In other words, a program to compute $f_\tau^\Diamond$ is simple to express with the functions $f_i \in \tau$. It turns out that there is a constant $c \in \mathbb{N}$ across all finite sets of partial computable functions τ that bounds the description length of such a program given a description of τ.

3.4 The uniform simplicity of recursive completion

We cannot demand from our models to be able to learn functions that are arbitrarily complex to describe with the underlying function set τ.

For the recursive completion $f_\tau^\Diamond$ however, we will show that given a description of τ, the Kolmogorov complexity of $f_\tau^\Diamond$ is upper bounded by a constant that is independent of the function set τ.

To establish this statement, we will construct a Turing Machine $\mathcal{T}_\Diamond$ that first receives a description of the functions $f_i \in \tau$ and the scale $a \in \mathbb{N}$ as parameters, and then computes $(f_\tau)_a^\Diamond(n)$ for any $n \in \mathbb{N}$ that is appended as input.

If $f_i \in \tau$ is partial computable, there exists a Turing Machine $\mathcal{T}_i$ that computes f_i. Providing the encoding $\mathrm{enc}(\mathcal{T}_i)$ of such a Turing Machine and the arity of f_i will constitute a sufficient description of f_i. The encoding $\mathrm{enc}(\tau)$ can therefore be thought of as a self-delimiting encoding of the *sequence* $\big[\mathrm{enc}(\mathcal{T}_1), \mathrm{ar}(f_1), \dots, \mathrm{enc}(\mathcal{T}_j), \mathrm{ar}(f_j)\big]$. In the spirit of

Equation 2.8, the self-delimiting encoding of a sequence $[z_1, \ldots, z_n]$ of binary strings z_i is defined as

$$[z_1, \ldots, z_n] := [z_1, \varepsilon][z_2, \varepsilon] \ldots [z_{n-1}, \varepsilon] z_n \in \{0,1\}^*, \tag{3.16}$$

where $\varepsilon \in \{0,1\}^*$ again is the empty string as introduced in Definition 2.2.

Given such an encoding, $\mathcal{T}_\Diamond$ can then simulate the Turing Machines $\mathcal{T}_i$ just as the universal Turing Machine U from Equation 2.7, and sum up their outputs to obtain f_τ.

Of course, we have to make the assumption that every function $f_i \in \tau$ is partial computable. But any function that can be computed by a machine must necessarily be partial computable according to the Church-Turing thesis. Therefore, this assumption is perfectly feasible in practice.

Using the *unconditional* Kolmogorov complexity $K(\cdot)$ to quantify how the descriptive complexity $K\left(f_\tau^\Diamond\right)$ evolves across different τ would however be deceptive for our argument, since it interfuses the expressive power of the *model structure* with the expressive power of the underlying building block function set τ. The non-recursivity of models is however only property of the former, while the latter can be thought of as the prior knowledge the model can assume to express patterns. Therefore, we are instead going to use the *conditional* Kolmogorov complexity $K\big((f_\tau)_a^\Diamond \mid \mathrm{enc}(\tau)\big)$. In this way, our argument substantiates that such non-recursive *structure* hinders models from expressing functions that yet would have been simple to describe with the underlying function set τ.

Theorem 3.9 (Uniform Kolmogorov Complexity Bound for Recursive Completion)
There exists a constant $c \in \mathbb{N}$ such that the following holds:

Given an arbitrary finite set of partial computable functions $\tau = \{f_1, \ldots, f_j\}$. Let $\mathcal{T}_i$ be a Turing Machine that computes f_i. Denote by $\mathrm{enc}(\tau)$ the self-delimiting encoding of the sequence $\big[\mathrm{enc}(\mathcal{T}_1), \mathrm{ar}(f_1), \ldots, \mathrm{enc}(\mathcal{T}_j), \mathrm{ar}(f_j)\big]$ as in Equation 3.16.

Then, $K((f_\tau)_a^\Diamond \mid \mathrm{enc}(\tau)) \leq c + 2 \cdot \log_2(a)$.

Proof. Given τ and $\mathrm{enc}(\tau)$ as above.

We define a Turing Machine $\mathcal{T}_\Diamond$ that computes the scaled recursive completion in the following way: $\mathcal{T}_\Diamond$ expects its input in the self-delimiting format $\big[\mathrm{enc}(\tau), [x_a, x_n]\big]$, where x_a and x_n are the binary encodings of the parameter a and the argument n.

Initially, $\mathcal{T}_\Diamond$ verifies the syntactical correctness of its input. In particular, it segments the input string according to the delimiters and checks whether the string segments for n,a, or the arities $\mathrm{ar}(f_i)$ truly encode an integer in the correct format. Moreover, it checks whether the string segments for $\mathcal{T}_i$ actually encode valid Gödel numbers. If any of the above checks fail, $\mathcal{T}_\Diamond$ halts.

Besides the input tape and the output tape, $\mathcal{T}_\Diamond$ also uses three auxiliary tapes, the *counting tape*, the *computation tape*, and the *accumulation tape*. First, $\mathcal{T}_\Diamond$ writes the integer input n both on the counting tape and the output tape. On the output tape, it further multiplies this integer by a.

Now, $\mathcal{T}_\Diamond$ repeats the ensuing procedure until the counting tape is eradicated blank.

1. Clean the computation tape and the accumulation tape from any non-blank symbols. Write a 0 on the accumulation tape.

2. Interpret the binary sequence x on the output tape as a natural number $x_\ell, \ell \in \mathbb{N}$.

3. From $i = 1$ to $i = j$, fetch the Gödel number of $\mathcal{T}_i$ and its input arity $k := \mathrm{ar}(f_i)$ from the input tape. Invoke $\mathcal{T}_i$ on the k-wise self-delimited concatenation of x_ℓ and add its output to the encoded integer on the accumulation tape.

4. After having added the output of all j Turing Machines, copy the encoded string from the accumulation tape to the output tape.

5. Subtract the integer encoded on the counting tape by 1.

After this iterative procedure terminates, $\mathcal{T}_\Diamond$ halts immediately.

In each iteration, if ℓ is the integer encoded on the output tape by x_ℓ, the accumulation tape with contain the value $f_\tau^-(\ell)$. Since $\ell = a \cdot n$ at the very beginning, the string x written to the output tape after the mth iteration will encode $f_\tau^{(m)}(a \cdot n)$. As the initial argument n is written on the counting tape too, exactly n iterations will be performed. Therefore, the final output x encodes $f_\tau^{(n)}(a \cdot n) = (f_\tau)_a^\Diamond(n)$. Denote by $f_\Diamond = f_{\mathcal{T}_\Diamond}$ the partial computable function that $\mathcal{T}_\Diamond$ computes. Then, $f_\Diamond\big(\big[\mathrm{enc}(\tau), [x_a, x_n]\big]\big) = (f_\tau)_a^\Diamond(n)$ for all $a, n \in \mathbb{N}$.

To execute $\mathcal{T}_\Diamond$ on the universal Turing Machine U, we need to provide $\mathrm{enc}(\mathcal{T}_\Diamond)\big[\mathrm{enc}(\tau), [x_a, x_n]\big]$ Let c_0 denote the length of the Gödel number $\mathrm{enc}(\mathcal{T}_\Diamond)$. Comparing the self-delimiting encoding $[x_a, x_n] = [x_a, \varepsilon]x_n = yx_n$, we find that $y = [x_a, \varepsilon]$ comprises $2 \cdot l(x_a) + 1 = 2 \cdot \lceil \log_2(a) \rceil + 1 \leq 3 + 2 \cdot \log_2(a)$ bits.

With $c := c_0 + 3$, we therefore finally obtain

$$K((f_\tau)_a^\Diamond \mid \mathrm{enc}(\tau)) = \min_{p,p' \in \{0,1\}^*} \{l(p) + l(p') \mid U\big(p[\mathrm{enc}(\tau), p'x_n]\big) = (f_\tau)_a^\Diamond(x_n) \text{ for all } n \in \mathbb{N}\} \tag{3.17}$$

$$\leq l(\mathrm{enc}(\mathcal{T}_\Diamond)) + l([x_a, \varepsilon]) \tag{3.18}$$

$$= c + 2 \cdot \log_2(a). \tag{3.19}$$

□

Figure 3.4 sketches the interplay of the different tapes that $\mathcal{T}_\Diamond$ orchestrates to compute $f_\tau^\Diamond$. Because the Turing Machine $\mathcal{T}_\Diamond$ regards τ merely as one of its inputs, it has a constant description length across all possible function sets τ. Together with Corollary 3.8.1, this Theorem demonstrates that the structural limitation of non-recursive models can not be arbitrarily mended by appropriate choices of the underlying function set, including activation functions. No matter how expressive the underlying function set is, the conditional Kolmogorov complexity of functions that cannot even be expressed by such models is upper bounded by a constant.

The corollary below joins these two results together.

Corollary 3.9.1 (Simple Functions that non-recursive Models cannot Express)
There exists a constant $c \in \mathbb{N}$ such that the following holds:

Given an arbitrary function set $\tau = \{f_1, \ldots, f_j\}$, such that each f_i is firstly partial computable and secondly strictly monotonously increasing or bounded. Assume that there is at least one $f_i \in \tau$ that is strictly monotonously increasing and has arity $\mathrm{ar}(f_i) > 1$, e.g. the addition function. Denote the encoding $\mathrm{enc}(\tau)$ just as in Theorem 3.9.

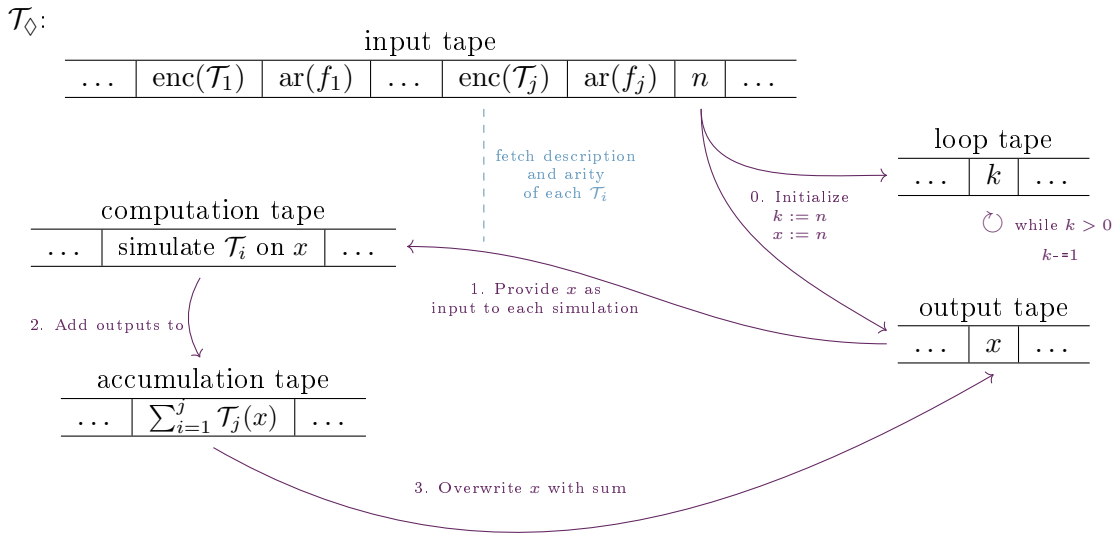


Figure 3.4: Uniform Turing Machine that computes the recursive completion.

For any $m \in \mathbb{N}$, there are $\sqrt{2}^{m-c} - 1$ functions $h : \mathbb{N} \to \mathbb{N}$ with a conditional Kolmogorov complexity

$$K\big(h \mid \mathrm{enc}(\tau)\big) \leq m$$

that cannot be expressed by any non-recursive model over τ.

Proof. By Corollary 3.8.1, the function $(f_\tau)_a^\Diamond \notin \mathcal{F}_\tau$ for any $a \geq 1$. At the same time, Theorem 3.9 ensures that there is a constant c such that $K((f_\tau)_a^\Diamond \mid \mathrm{enc}(\tau)) \leq c + 2 \cdot \log_2(a)$ for any $a \in \mathbb{N}$.

Let $m \in \mathbb{N}$ be arbitrary. For any $a \in \mathbb{N}$ with $a \leq \sqrt{2}^{m-c}$, $c + 2 \cdot \log_2(a) \leq c + 2 \cdot \frac{1}{2}(m-c) \leq m$. Excluding $a = 0$ yields our result. □

If we want to enable our models to express functions up to a certain simplicity, we must extend the structure of our model classes by recursive elements. But even if our models were expressive enough to capture recursive functions like the recursive completion $(f_\tau)_a^\Diamond$, they are yet provided no incentive to favour them over other, possibly non-recursive hypotheses h.

In the next subsection, we will see that for any finite dataset, there are infinitely many non-recursive functions $f \in \mathcal{F}_\tau$ that remain perfectly consistent with $(f_\tau)_a^\Diamond$ and therefore achieve the optimal score in the IRM objective when the expected risks R are accordingly replaced by the empirical risks $\hat{R}$ as is usually done in practice. Consequently, non-recursive models over τ can indeed receive an optimal score although

Although no finite dataset can eliminate these optimal yet non-recursive functions, it will be shown that beyond some size threshold n_0, any function $f \in \mathcal{F}_\tau$ that still remains consistent with a finite dataset S of size $|S| \geq n_0$ must have a higher conditional Kolmogorov complexity than the true function $(f_\tau)_a^\Diamond$. For that reason, when optimization objectives

additionally evaluated functions by their Kolmogorov complexity, sufficiently large finite datasets would in fact distinguish the true function $(f_\tau)_a^\Diamond$ from non-recursive functions.

3.5 Optimization objectives lack a simplicity bias

With full access to the underlying distribution P by which problem instances $n \in \mathbb{N}$ occur, the expected risk R can certainly identify the true function $(f_\tau)_a^\Diamond$ as its unique minimizer. If any $n \in \mathbb{N}$ had a non-zero probability according to P, then any function that disagrees on at least one instance with $(f_\tau)_a^\Diamond$ would obtain a non-zero risk R. In such a case of perfect knowledge of P, even the standard ERM principle would suffice to identify the true function. In practice however, datasets are bound to finiteness and the empirical risk $\hat{R}$ hence leaves functions infinitely many degrees of freedom as it merely comes to fitting the samples. These degrees of freedom are either effaced by restrictions on the model class or further inductive biases in the optimization formulation.

After the last subsections addressed the former, this subsection demonstrates that inductive biases such as the simplicity bias towards minimal Kolmogorov complexity can further filter out a huge fraction of alternative hypotheses that would otherwise receive the same optimal optimization function value.

Firstly, for realistic assumptions on what functions the function set τ comprises, any finite dataset S leaves infinitely many functions that non-recursive models could express to basically memorize the samples in S and hence achieve optimal risk. Even domain generalization optimization objectives like IRM can by itself not distinguish the recursive completion $(f_\tau)_a^\Diamond$ from these non-recursive functions $f \in \mathcal{F}_\tau$.

Complementing this result, we prove that all non-recursive functions eventually render suboptimal when we further take into account their Kolmogorov complexity. Figure 3.5 underscores the discriminative power that Kolmogorov complexity or any similar simplicity bias exerts on the functions that remain consistent with the dataset. Because the mere requirement of fitting samples leaves infinitely many degrees of freedom, optimization objectives like ERM assign infinitely many non-recursive functions $f \in \mathcal{F}_\tau$ the same optimal score for any finite dataset generated by $f_\tau^\Diamond$ (cf. Figure 3.5a). On the other hand, only finitely many non-recursive functions $f \in \mathcal{F}_\tau$ remain at least as optimal as the true function if functions are additionally evaluated by a simplicity bias like their Kolmogorov complexity (cf. Figure 3.5b).

Over and above, there exists a threshold n_0 such that for *any* dataset S of size $|S| \geq n_0$, even these finitely many more optimal alternatives are eliminated, because any non-recursive function $f \in \mathcal{F}_\tau$ that still remains consistent with S must have a higher conditional Kolmogorov complexity $K\big(f \mid \mathrm{enc}(\tau)\big)$ than the true function $(f_\tau)_a^\Diamond$.

Lemma 3.10 (IRM does not Identify Recursive Completion)
Let $\tau := \{f_1, \ldots, f_j\}$ be an arbitrary function set. Let $a \in \mathbb{N}$ with $a \geq 1$ be arbitrary. Assume that the functions below are included in τ:

1. *The addition function $f_+(x, y) = x + y$,*
2. *The multiplication function $f_\times(x, y) = x \cdot y$,*

3. *The equality function* $f_{=}(x,y) = \begin{cases} 1, & x = y \\ 0, & x \neq y \end{cases}$,

4. *The constants* $c_1 = 1$, *and* $c_0 = 0$.

For every finite dataset $S = \left\{\left(x_1, (f_\tau)_a^\Diamond(x_1)\right), \ldots, \left(x_k, (f_\tau)_a^\Diamond(x_k)\right)\right\}$, *there is an infinite subset* $F_S \subset \mathcal{F}_\tau$ *of functions* $f \in F_S$ *that achieve the optimal score according to the IRM objective from Equation 2.3 with the 0-1-loss and empirical risks* $\hat{R}$, *but are not equivalent to* $(f_\tau)_a^\Diamond$.

Proof. Let τ be as above. Fix an arbitrary $a \in \mathbb{N}$ with $a \geq 1$.

We show that we can construct functions that memorize arbitrarily many samples.

To begin with, any $n \in \mathbb{N}$ can be expressed by a constant function $c_n \in \mathcal{F}_\tau$. This is proved in a straightforward inductive manner. For the base case, we already have $c_0 = 0 \in \tau$. Now, assume that there is an $n \in \mathbb{N}$ such that there exists a function $c_n \in \mathcal{F}_\tau$ with $c_n = n$. Then, we construct $c_{n+1} = f_+(c_n, c_1) = n+1$. By the inductive definition of non-recursive functions in 3.1, it holds that $c_{n+1} \in \mathcal{F}_\tau$, and the overall statement draws on the induction principle. In the following, we will therefore directly write n instead of c_n for notational simplicity.

Thereby, for any $m \in \mathbb{N}$, we can construct a function $f_m \in \mathcal{F}_\tau$ that memorizes the values of $(f_\tau)_a^\Diamond$ on $\{0, \ldots, m\}$. This statement is analogously proved by induction over $m \in \mathbb{N}$. For the base case $m = 0$, we define $f_0(n) := f_\times\left(f_=(0,n), (f_\tau)_a^\Diamond(0)\right) = \begin{cases} (f_\tau)_a^\Diamond(0), & n = 0 \\ 0, & n \neq 0 \end{cases}$.

As f_0 is composed of building block functions in τ, it holds that $f_0 \in \mathcal{F}_\tau$.

But since $f_+ \in \tau$ is strictly monotonously increasing with arity $\mathrm{ar}(f_+) > 1$, Lemmas 3.5 and 3.6 ensure that

$$(f_\tau)_a^\Diamond(n) \overset{3.6}{\geq} f_+^{(n)}(n) \overset{3.5}{\geq} n > 0 \qquad \text{for all } n \geq 1. \tag{3.20}$$

For that reason, $f_0(n) = (f_\tau)_a^\Diamond(n)$ only holds for $n = 0$.

Moving on with the induction hypothesis (IH), assume that there is an $m \in \mathbb{N}$ and a function $f_m \in \mathcal{F}_\tau$ such that $f_m(n) = (f_\tau)_a^\Diamond(n)$ holds for all $n \leq m$, but $f_m(n) = 0$ for $n > m$.

Then, we define

$$f_{m+1}(n) := f_+\Big(\tag{3.21}$$

$$f_\times\left(f_=(m+1, n), (f_\tau)_a^\Diamond(m+1)\right), \tag{3.22}$$

$$f_m(n)\Big). \tag{3.23}$$

Again, f_{m+1} is a valid composition of functions in $\mathcal{F}_\tau$, thus $f_{m+1} \in \mathcal{F}_\tau$.

By the induction hypothesis, it holds that

$$f_{m+1}(n) = \begin{cases} (f_\tau)_a^\Diamond(m+1), & n = m+1 \\ f_m(n), & n \neq m+1 \end{cases} \overset{\text{(IH)}}{=} \begin{cases} (f_\tau)_a^\Diamond(m+1), & n = m+1 \\ (f_\tau)_a^\Diamond(n), & n \leq m \\ 0, & n > m+1 \end{cases}. \tag{3.24}$$

Together with Equation 3.20, we have $f_{m+1}(n) \neq (f_\tau)_a^\Diamond(n)$ for all $n > m+1$.

By the induction principle, we therefore conclude that for any $m \in \mathbb{N}$ a function $f_m \in \mathcal{F}_\tau$ such that $f_m(n) = (f_\tau)_a^\Diamond(n)$ if and only if $n \leq m$. Collect these functions into $F := \{f_m \mid m \in \mathbb{N}\}$. Because $f_m(m) = (f_\tau)_a^\Diamond(m) \overset{(3.20)}{\neq} 0 = f_k(m)$ for all $k < m$, all these functions are pairwise different, and $|F| = \infty$.

Now, let $\mathcal{P}_{tr} := \{P_\varepsilon \mid \varepsilon \in \mathcal{E}_{tr}\}$ be an arbitrary set of training distributions, $\mathcal{E}_{tr} \subset \mathcal{E}$. Let $S = \{(x_1, (f_\tau)_a^\Diamond(x_1)), \ldots, (x_k, (f_\tau)_a^\Diamond(x_k))\}$ be an arbitrary dataset of some size $k \in \mathbb{N}$ that summarizes all observed samples from $\mathcal{P}_{tr}$. The 0-1-loss is defined as

$$\ell(x, y) = \begin{cases} 1, & x = y \\ 0, & x \neq y. \end{cases} \tag{3.25}$$

Let $m_0 := \max_{1 \leq i \leq k} x_i \in \mathbb{N}$ be the largest instance in S. For any $m \geq m_0$, it holds that $f_m(x_i) = (f_\tau)_a^\Diamond(x_i)$ for all x_i in S. Therefore, the risks $\hat{R}_\varepsilon(f_m) = 0$ are optimal for any training distribution $P_\varepsilon \in \mathcal{P}_{tr}$. There can be no function f with $\hat{R}_\varepsilon(f) < \hat{R}_\varepsilon(f_m)$ for any $P_\varepsilon \in \mathcal{P}_{tr}$. For that reason, f_m is an optimal solution to Equation 2.3.

Choosing $F_S := \{f_m \in F \mid m \geq m_0\}$ concludes our proof. □

Note that the function set $\tau' := \{f_+, f_\times, c_1, c_0, f_=\}$ that was assumed in Lemma 3.10 already satisfies the conditions of Theorem 3.8 for itself, since each function is either bounded or strictly monotonously increasing, and the addition function f_+ has arity $\text{ar}(f_+) > 1$. Although any non-recursive model can therefore not express $(f_\tau)_a^\Diamond$, there are arbitrarily many non-recursive functions such models could draw upon to achieve an optimal risk by merely memorizing information.

Theorem 3.8 even paves the way to a stronger result that assures that *any* non-recursive functions $f \in \mathcal{F}_\tau$ — not only the ones explicitly constructed in the proof of Lemma 3.10 — that agree with $(f_\tau)_a^\Diamond$ on sufficiently many points must obtain a higher Kolmogorov complexity than $(f_\tau)_a^\Diamond$.

If optimization objectives were to take into account this kind of simplicity bias, they could hence already filter out a huge amount of hypotheses despite the quantitative incompleteness of the training dataset.

Theorem 3.11 (Surpassing the Kolmogorov Complexity Threshold)
Let the function set τ be as in Theorem 3.8. There is an $n_0 \in \mathbb{N}$ such that for every dataset S with size $|S| > n_0$, all functions $f \in \mathcal{F}_\tau$ that remain consistent with S have a higher conditional Kolmogorov complexity than $(f_\tau)_a^\Diamond$,

$$K\big(f \mid \text{enc}(\tau)\big) > K\big((f_\tau)_a^\Diamond \mid \text{enc}(\tau)\big). \tag{3.26}$$

Proof. Let $a \in \mathbb{N}$ with $a \geq 1$ be arbitrary. By Theorem 3.9, the conditional Kolmogorov complexity $K\big((f_\tau)_a^\Diamond \mid \text{enc}(\tau)\big) \leq c + \log_2(a)$ for some $c \in \mathbb{N}$.

Denote $c_0 := c + \log_2(a)$ in the following. By the geometric sum, there are only $2^{c_0+1} - 1$ binary strings with a length shorter than or equal to c_0.

Therefore, the set

$$F := \big\{f \in \mathcal{F}_\tau \mid K\big(f \mid \text{enc}(\tau)\big) \leq K\big((f_\tau)_a^\Diamond \mid \text{enc}(\tau)\big)\big\} \tag{3.27}$$

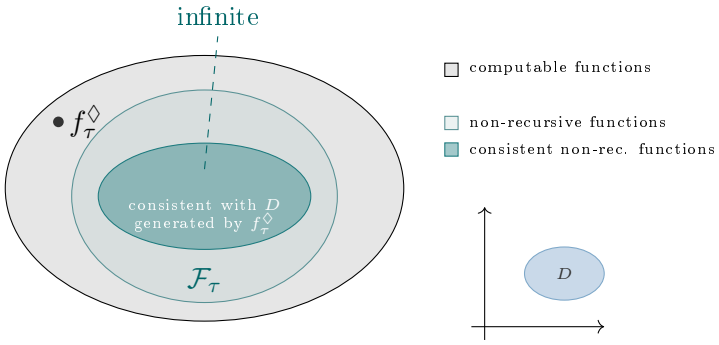
must be finite.

Because $F \subset \mathcal{F}_\tau$, Corollary 3.8.1 ensures that for each $f \in F$, there is an $n_0(f)$ such that $f(n) \neq (f_\tau)_a^\Diamond(n)$ for all $n \geq n_0(f)$.

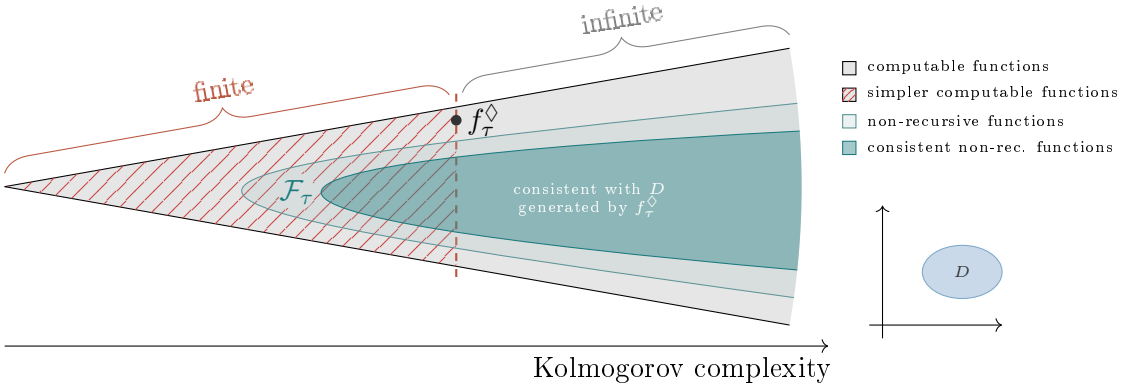
Let $n_0 := \max_{f \in F} n_0(f)$ be the maximum of these indices.

Now, let $S = \{(x_1, (f_\tau)_a^\Diamond(x_1)), \ldots, (x_k, (f_\tau)_a^\Diamond(x_k))\}$ be an arbitrary dataset of size $k > n_0$. Denote by $m_1 := \max_{1 \leq i \leq k} x_i$ the largest natural number instance in S. Since $|S| \geq n_0+1$, $m_1 \geq n_0$. By the choice of n_0, we have $f(m_1) \neq (f_\tau)_a^\Diamond(m_1)$ for all $f \in F$. For any $f \in \mathcal{F}_\tau$ that is consistent with the functional pairs in S, it particularly holds that $f(m_1) = (f_\tau)_a^\Diamond(m_1)$. For that reason, $f \notin F$ and hence $K(f \mid \mathrm{enc}(\tau)) > K((f_\tau)_a^\Diamond \mid \mathrm{enc}(\tau))$. □

As soon as the dataset D contains a sufficiently large instance $x_i \geq m_1$, any non-recursive function with a lower Kolmogorov complexity than $f_\tau^\Diamond$ must become inconsistent with D, since it maps x_i to a smaller value than $f_\tau^\Diamond$ as was shown in the inexpressivity result of Theorem 3.8. Figure 3.6 visualizes this condition in the spirit of Figure 3.5.



(a) Without any inductive bias beyond merely fitting the dataset, infinitely many non-recursive functions remain consistent and hence optimal according to optimization objectives like ERM.



(b) Additionally exerting Kolmogorov complexity as a simplicity bias on the consistent functions induces an ordering on these consistent functions, leaving at most finitely many non-recursive functions with a better score than $f_\tau^\Diamond$.

Figure 3.5: Juxtaposing the discriminate power of a Kolmogorov complexity bias beyond fitting a dataset with the indifference of standard optimization objectives like ERM and IRM to functions that perfectly fit the dataset.

The final message of this chapter can therefore be summarized as follows. If we want to enable algorithms to learn simple functions, we not only need to bestow our machine learning models with recursive structure, but also need to extend our optimization objectives to favour simpler functions over more complex ones.

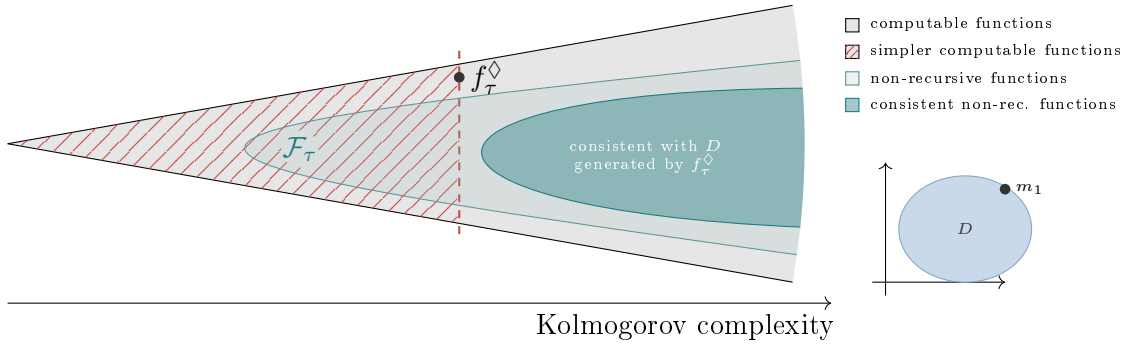


Figure 3.6: One sufficiently large sample renders all consistent non-recursive functions sub-optimal. Here, m_1 is the threshold in the proof of Theorem 3.11, above which every non-recursive functions with a lower Kolmogorov complexity than $f_\tau^\Diamond$ must become inconsistent by Theorem 3.8.

It remains to ask how such measures could alleviate the strong information-theoretic conditions that training distributions still have to satisfy to fall within the scope of generalization guarantees. In the next chapter, we demonstrate how Kolmogorov complexity paves the way to formulate such sufficient conditions in a general, but still nearly optimal manner. To that end, it first exhibits that it is not meaningful to condition learnability on the number of samples in dataset. Instead, it proposes to measure the information that a dataset conveys about the functions that could have generated it in terms of Kolmogorov complexity, and demonstrates how this measure yields not only generalization guarantees, but could realise simplicity biases in practice.

Chapter 4

Sufficient Information for Out-of-Distribution Learnability

In Chapter 3, we argued that the reasonable maximum demand we can pose on generalizing observations in a dataset D is to infer a *simplest consistent function* with D. With simplest consistent function, we refer to a function f with minimal Kolmogorov complexity $K(f)$ that still coincides with every functional tuple $(x, y) \in D$. Accordingly, Theorem 3.11 demonstrated how optimization objectives that exerted the discriminative power of Kolmogorov complexity to determine such simplest consistent functions could already filter out a huge fraction of false hypotheses f'. Particularly, it stated that for any function f from a certain class of recursive functions, there exists a threshold n_0 such that for all datasets D comprised of more than n_0 instances, all non-recursive functions f' that still remain consistent with D must have a higher Kolmogorov complexity than the true function f, rendering f distinguishably optimal. This learnability condition is *distribution-free* to the extent that it merely requires the $n \geq n_0$ instances in D to be pairwise different. Apart from that, the training distribution these instances stem from can be arbitrary.

Such statistical conditions that suffice to guarantee learnability even out-of-distribution are of essential interest in the domain generalization setting, where the training distribution can theoretically differ arbitrarily from the distribution during testing or deployment. But so far, such conditions are either quite conservative, or restricted to a specific class of functions [1], [2].

Similarly, the aforementioned theorem has two critical drawbacks:

1. The scope of functions for which this learnability guarantee applies is drastically limited. It only holds for functions $f = f^{\Diamond}_{\tau}$ from a particular class of recursive functions (see Definition 3.7).

2. Secondly, the hypothesis class $\mathcal{H}$ is limited, since the statement only targets the suboptimality of non-recursive functions $f \in \mathcal{F}_{\tau}$. Even for datasets with more than n_0 instances, there could still be other partial computable functions with a lower Kolmogorov complexity than $f^{\Diamond}_{\tau}$.

As will be shown later, there are realistic cases in which even datasets of infinite size could still fail to render the true function f the simplest consistent function. In general, it is

therefore delusive to condition learnability on the *number* of samples in D. Large datasets do not necessarily contain much information when their samples result from monotonously applied regularities with small description length.

Instead, we will see that it is more meaningful to directly quantify the *information* that D conveys about the functions that could have *generated* the samples. In our noise-free setting, generating a dataset means that $f(x_i) = y_i$ for each sample (x_i, y_i) in D. In particular, this chapter puts forward how Kolmogorov complexity smooths the way to a holistic information-theoretic condition for the learnability of *any* partial computable function $f : D \to \{0,1\}^*, D \subseteq \{0,1\}^*$, thus any function we could reasonably desire to learn computationally. This condition quantifies both the data-generating function f and the dataset D in terms of their Kolmogorov complexity and does not require further information on f or D. Despite the generality of this condition, it still remains optimal in terms of the informational resources that are necessary to learn f. This result hence substantiates that small, but informative datasets can suffice to correctly infer the underlying function, even if the hypothesis class is as large as the class of all partial computable functions.

Beyond this condition, given *any* dataset D generated by some f, this chapter provides an upper bound on the number of consistent yet inequivalent partial computable functions f' that do not have a higher Kolmogorov complexity than f and hence still stand in the way of inferring f.

4.1 Sample size does not avail learnability

We first state the underlying question this chapter seeks to answer in the first instance:

> Given an arbitrary partial computable function $f : \{0,1\}^* \to \{0,1\}^*$ with Kolmogorov complexity $K(f)$, what conditions does a dataset D comprised of samples $(x, f(x))$ need to satisfy to render f the partial computable function with the lowest Kolmogorov complexity that remains consistent with D?

Theorem 3.11 merely required D to contain sufficiently many samples. However, merely conditioning learnability on the sample size is not possible if we can not rest on the *i.i.d.* assumption.

We demonstrate on two realistic examples why such a condition would be vacuous. Firstly, there are functions f with low Kolmogorov complexity such that even some infinite datasets would still not render f the simplest consistent function. Secondly, we prove that there are functions with arbitrarily large Kolmogorov complexity that can be identified by a single sample. In both cases, we point up that the paradox resolves if we instead regarded datasets by a quantity based on Kolmogorov complexity.

1. Infinite datasets despite low Kolmogorov complexity

Consider the constant functions $f_0(x) = 0, f_1(x) = 1$ for all $x \in \{0,1\}^*$. On the opposite site, take the mod_2 function $\mathrm{mod}_2(x) = x \mod 2$. f_0 and f_1 arguably both have a lower Kolmogorov complexity than mod_2 because they directly write a symbol on the output tape and terminate while a Turing Machine that computes mod_2 would intuitively require two states to represent whether the last encountered symbol was 0 or 1. For that reason, we assume $K(f_0), K(f_1) < K(\mathrm{mod}_2)$ without loss of generality. If this inequality would hold

conversely for f_1 or f_0, we can conduct the same argument symmetrically. The two datasets $D_0^n := \{(2k, 0) \mid k \in \mathbb{N}, k \leq n\}$ and $D_1^n := \{(2k+1, 0) \mid k \in \mathbb{N}, k \leq n\}$ remain consistent with f_0 and f_1 respectively for any sample size n. This consistency even holds for the infinite datasets $D_0 := \{(2k, 0) \mid k \in \mathbb{N}\}$ and $D_1 := \{(2k+1, 0) \mid k \in \mathbb{N}\}$. A guarantee that conditioned the learnability of mod_2 merely on the sample size does therefore not exist. This paradox fades away if we consider the Kolmogorov complexity of the datasets D_0^n and D_1^n instead. Provided the instances $x_{2k}, k \leq n$ in a self-delimiting format, a Turing Machine with a similar simplicity as the one that computes the constant function f_0 could generate the dataset D_0^n. As we prove later in a more general scope, the *conditional Kolmogorov complexity* $K\big((y_0, y_2, \ldots, y_{2n}) \mid (x_0, x_2, \ldots, x_{2n})\big)$, where $y_i = f_0(x_i)$, does not exceed the Kolmogorov complexity $K(f_0)$ by a constant.

2. Single samples despite arbitrarily high Kolmogorov complexity

On the other hand, there are functions with arbitrarily high Kolmogorov complexity that can be learned by datasets $D = \{(x_1, y_1)\}$ with only a single sample. For any $y \in \{0,1\}^*$, consider the dataset $D_y = \{(0, y)\}$. There must exist a partial computable function f_y that is consistent with D and achieves the lowest Kolmogorov complexity among all such f. If there are multiple ones with the same lowest complexity, take any of them. As an intuitive example, this function f_y could be the constant function $f_y(x) = y, x \in \{0,1\}^*$. For any $y \neq y'$, the functions $f_y, f_{y'}$ are pairwise different, since $f_y(0) \neq f_{y'}(0)$. Consequently, the set of functions $\{f_y \mid y \in \{0,1\}^*\}$ is infinite, and there hence exists no upper bound on the Kolmogorov complexity of the functions comprised. Again, let us take a look at this paradox from the viewpoint of Kolmogorov complexity. Although each D_y comprised solely one sample, an algorithm that produces D_y on input 0 has to generate y from scratch. These algorithms therefore have a Kolmogorov complexity that similarly grows to infinity as $y \to \infty$, by the same counting argument as in the Incompressibility Lemma 4.3. Likewise will the conditional Kolmogorov complexity $K\big(y \mid 0\big)$ scale to infinity.

These examples expose the discrepancy between the mere number of samples in a dataset and the actual *information* these samples convey about the functions that could have generated it. In the next section, *functional information* is proposed as a quantity that measures this kind of information within a dataset. It will be shown that functional information has a tight relation with the Kolmogorov complexity of the function that could have generated this dataset and hence allows us to culminate in an learnability condition for the overarching hypothesis class of arbitrary partial computable functions.

4.2 Quantifying the functional information in a dataset

To quantify how much information a dataset D conveys about the functions that could have generated it, it is delusive to directly consider the *unconditional* Kolmogorov complexity of D, because it also incorporates the information that is required to generate the instances x_i. These x_i are however independent from the actual function f, but the y_i are not. Therefore, it is more accurate to consider how much information is *added* after the instances $x_1, \ldots, x_n$ are completed by their labels $y_1, \ldots, y_n$. At first glance, this notion of information could be formalised by the Kolmogorov complexity of $[y_1, \ldots, y_n]$ given $[x_1, \ldots, x_n]$, where $[z_1, z_2, \ldots, z_n]$ denominates the self-delimiting encoding of the string sequence $z_1, z_2, \ldots, z_n$ just as in Equa-

tion 3.16. Aggregating all instances and labels into one long self-delimited string respectively however entails cumbersome peculiarities that spoil some desirable properties. Although the following definition does hence not ultimately identify the aforementioned notion of information, its treatment elucidates the necessity of the definition we eventually end up with. To that end, we first formalise the aforementioned idea as the *joint functional information* in a dataset.

Definition 4.1 (Joint functional information in a dataset)
Let $D = \{(x_1, y_1), \dots, (x_n, y_n)\}, n \in \mathbb{N}$ *be an arbitrary finite, ordered dataset, where* $x_i, y_i \in \{0,1\}^*$. *We denominate by* $[z_1, z_2, \dots, z_n]$ *the self-delimiting encoding of the string sequence* $z_1, z_2, \dots, z_n$ *just as in Equation 3.16. Then, the joint functional information in* D *is defined as*

$$K_{JF}(D) := K\big([y_1, y_2, \dots, y_n] \mid [x_1, x_2, \dots, x_n]\big). \tag{4.1}$$

To some extent, the joint functional information in a dataset is already closely related to the information within the functions that could have produced it, which is formally established in the Theorem below.

Theorem 4.2 (Joint functional information bound of datasets)
Let enc *be an arbitrary prefix-free encoding of Turing Machines. There is an additive constant* $c \in \mathbb{N}$ *such that the following holds:*

Let $D = \{(x_1, y_1), \dots, (x_n, y_n)\}, n \in \mathbb{N}$ *be an arbitrary finite, ordered dataset. For any partial computable function* f *that satisfies* $f(x_i) = y_i$ *for all* $1 \leq i \leq n$, *we have*

$$K\big([y_1, \dots, y_n] \mid [x_1, \dots, x_n]\big) \leq K(f) + c. \tag{4.2}$$

Proof. We construct a Turing Machine V that generates datasets in the following way.

V expects as input a string that begins with the prefix-free encoding $\mathrm{enc}(\mathcal{T})$ of a Turing Machine T. Thereon follows the self-delimited encoding of the self-delimited sequence $\big[[x_1, \dots, x_n], \varepsilon\big]$, where the enclosing self-delimitation arises from the convention in the Definition 2.3 of conditional Kolmogorov complexity.

For each input x_i, V simulates $\mathcal{T}$ on x_i. After the simulation terminates with output y_i, V appends $[y_i, \varepsilon]$ to the string on the output tape. If x_i was the last string, $i = n$, which V recognizes by the self-delimiting encoding, it merely appends y_i to the output tape instead. Denote the encoding length of this Turing Machine by $c := l(\mathrm{enc}(V))$.

Now, let f be an arbitrary partial computable function and $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be an arbitrary finite, ordered dataset such that $f(x_i) = y_i$. Let $\mathcal{T}_f$ be the Turing Machine with the shortest encoding that computes f. It thence holds that $K(f) = l(\mathrm{enc}(\mathcal{T}_f))$. Because $f(x_i) = y_i$ by assumption, the instances x_i in D all belong to the definition range of f. For that reason, $\mathcal{T}_f$ will certainly terminate on every input x_i.

Therefore, invoking the universal Turing Machine U on the string $\mathrm{enc}(V)\,\mathrm{enc}(\mathcal{T}_f)[x_1, \dots, x_n]$ yields the output $[y_1, \dots, y_n]$, and we conclude that

$$K([y_1, \dots, y_n] \mid [x_1, \dots, x_n]) = K\Big(U\big(\mathrm{enc}(V)\,\mathrm{enc}(\mathcal{T}_f)\big[[x_1, \dots, x_n], \varepsilon\big]\big)\Big) \tag{4.3}$$

$$\leq l(\mathrm{enc}(V)) + l(\mathrm{enc}(\mathcal{T}_f)) = K(f) + c. \tag{4.4}$$

□

Equivalently, this Theorem lower bounds the Kolmogorov complexity of partial computable functions that could have generated a dataset. For that reason, it sets the stage to a general, sufficient condition for the simplest consistent function to coincide with the true function f. Specifically, all datasets D generated by f satisfy $K_{JF}(D) \leq K(f)+c$. The closer the functional information in D approaches this upper bound, the more simpler functions must have been rendered inconsistent with some functional tuples (x, y) in D. Therefore, knowledge of the functional information in D enables to determine an upper bound on the number of alternative functions f' that do not have a higher Kolmogorov complexity than f but still remain consistent with a dataset D. Nonetheless, this leaves an essential question open. For any partial computable function f, *are* there actually datasets D generated by f that reach this functional information upper bound $K(f) + c$ and ergo narrow down the number of alternative simplest consistent functions? Or in greater detail, what is the maximum functional information that the class of ordered datasets generated by f, hereinafter denoted by $\mathcal{D}_f$, exhibits? Accordingly, we denote this maximum by

$$M_f := \max_{D \in \mathcal{D}_f} \{K_{JF}(D)\}. \tag{4.5}$$

This maximum is well-defined for each f, since any dataset D that was generated by f satisfies $K_{JF}(D) \leq K(f) + c$. But can we guarantee that M_f remains close to $K(f)$, say up to an additive constant? Or to go one step further:

> if D is inconsistent with all partial computable functions with $K(f) < k$, is the joint functional information $K_{JF}(D) \geq k$?

4.2.1 The inconveniences of joint functional information

For our definition of joint functional information, such a desirable lower bound for M_f is unfortunately impossible if we consider the class of partial computable functions as a whole. The datasets above comprise merely tuples over the *definition range* of a partial computable function f. On inputs where f is undefined, the Turing Machines that compute f will never halt. But this halting behaviour also contributes to the information within a Turing Machine. Turing Machines $\mathcal{T}$ that output 0 for all inputs x not longer than m but never halt when invoked on longer inputs arguably require more information to describe than Turing Machines $\mathcal{T}'$ that simply output 0 on any input. Consequently, the partial computable function f such that $f(w)$ is constantly 0 for all inputs that are not longer than m and undefined otherwise has a higher Kolmogorov complexity than the constant function $f'(w) = 0$ that is defined for all inputs. If we however want to learn the partial computable function f and not the total computable function f', any dataset over the definition range of f will not render f' inconsistent.

Even when the above question is only addressed for *total* computable functions, this restriction introduces the next quandary, because the halting problem is not computable [25].

Therefore, there exists no universal Turing Machine that accepts exactly the encodings of total computable functions. Any universal Turing Machine that accepts the encodings of total computable functions must necessarily also accept some non-total, partial computable

functions. Without making particular assumptions on the encoding, such lower bounds on the Kolmogorov complexity of a class of elements usually rest on the pigeonhole principle and the limitations of the encoding space, as we will later employ in Lemma 4.3. In this case however, the total computable functions could constitute only a tiny fraction of this encoding space, rendering such lower bounds vacuous.

On the other hand, for any *total* computable f there certainly exist datasets D such that any *partial* computable $f' \not\equiv f$ with $K(f') \leq K(f)$ is rendered inconsistent with D. For any such f', there must namely either exist an instance x with $f(x) \neq f'(x)$, or an instance x that is out of the definition range of f'. Gathering all these x together into one dataset D^* will certainly render f the unique simplest consistent function. But proving that the joint functional information in this dataset $K_{JF}(D^*)$ is also close to $K(f)$ is non-trivial, if not impossible.

The reason for that is that joint functional information does not satisfy some elementary mathematical properties that would render useful in such an argument. We provide two examples. Firstly, joint functional information is not monotonous with growing datasets. That is, it does not necessarily hold that $K_{JF}(D) \leq K_{JF}(D')$ for $D \subset D'$. Secondly, joint functional information is not invariant under permutation, it is sensitive to the order of the elements in D. Both of these peculiarities substantiate that joint functional information is yet inconsistent with the notion of information that we desire to quantify. But to advance these arguments, we first need to become aware of the existence of *incompressible* strings, which cannot be compressed beyond their own length.

Lemma 4.3 (Incompressible Strings)
For any $n \in \mathbb{N}$ and any $x \in \{0,1\}^*$, there exists a string $v_n \in \{0,1\}^n$ with $K(v_n \mid x) \geq n$.

Proof. Let $n \in \mathbb{N}$ be arbitrary. There are 2^n strings of length n, but only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ strings that are shorter than n. By the pigeonhole principle, there must be at least one $v \in \{0,1\}^n$ such that there exists no program p of length $l(p) < n$ with $U(p) = v$. For this v, we have $K(v \mid x) \geq n$. □

With this Lemma at hand, let us begin with the monotonicity for growing subsets. Our initial objective was to quantify the information that functions that could have generated a dataset exhibit. But as introduced in the beginning, *generating* a dataset D refers to generating each sample *in isolation*, not generating the joint label sequence $[y_1, \ldots, y_n]$ when given the joint instance sequence $[x_1, \ldots, x_n]$. However, the latter more loose definition allows to leak information across different samples. That is, some instance x_j might convey information about a different label y_i, and hence artificially simplify the production of $[y_1, \ldots, y_n]$. For this very reason, producing larger datasets must not necessarily require algorithms with more information that producing smaller datasets, as the following Lemma illustrates.

Lemma 4.4 (Joint functional information is not monotonous with growing subsets)
There exist subsets D, D' such that $D \subset D'$ but irrespective of the ordering of D and D', $K_{JF}(D) > K_{JF}(D')$.

Proof. Fix an arbitrary $x \in \{0,1\}^*$. For any $n \in \mathbb{N}$, Lemma 4.3 guarantees the existence of a string $y^{(n)} \in \{0,1\}^*$ with $K(y^{(n)} \mid x) \geq n$. Now, consider the three ordered datasets $D_1^{(n)} = \{(x, y^{(n)})\}, D_2^{(n)} = \{(y^{(n)}, 0), (x, y^{(n)})\}, D_3^{(n)} = \{(x, y^{(n)}), (y^{(n)}, 0)\}$. Obviously, $D_1^{(n)} \subset$

$D_2^{(n)}, D_3^{(n)}$. We will prove that $D_2^{(n)}$ and $D_3^{(n)}$ have a lower joint functional information than $D_1^{(n)}$ for sufficiently large n.

First, we construct two Turing Machines $\mathcal{T}_2, \mathcal{T}_3$ that can generate datasets of the kind 2 and 3 for arbitrary strings x and $y^{(n)}$. The Turing Machine $\mathcal{T}_2$ expects its input as a self-delimited sequence of two strings $\big[[y, x], \varepsilon\big]$. $\mathcal{T}_2$ just reads y and returns $[0, y]$. Similarly, $\mathcal{T}_3$ also expects its input as a self-delimited sequence of two strings $\big[[x, y], \varepsilon\big]$. But $\mathcal{T}_3$ now ignores x, reads y and returns $[y, 0]$. Hereinafter, we denote the encoding lengths of $\mathcal{T}_2$ and $\mathcal{T}_3$ as c_2 and c_3, respectively. For that reason, $K_{JF}(D_2^{(n)}) = K([0, y^{(n)}] \mid [y^{(n)}, x]) \leq c_2$. Symmetrically, $K_{JF}(D_3^{(n)}) \leq c_3$. However, $K_{JF}(D_1^{(n)}) \geq n$ by the definition of $y^{(n)}$. For $n > c_2, c_3$, we therefore conclude $K_{JF}(D_1^{(n)}) > K_{JF}(D_2^{(n)}), K_{JF}(D_3^{(n)})$, although $D_1^{(n)} \subset D_2^{(n)}, D_3^{(n)}$. □

A realistic example that illustrates this inconsistency is a poorly designed exam, where the second question already hints at the true answer to the first question. The above proof already elicits the second inconsistency, namely that joint functional information is sensitive to the order in the dataset. The ordering of the samples might allow Turing Machines to draw on benign regularities within the output, which the subsequent Lemma exemplifies.

Lemma 4.5 (Joint functional information is not ordering-invariant)
There exists an ordered dataset $D := \{(x_1, y_1), \ldots, (x_n, y_n)\}$ *and a permutation* π *over* $\{1, \ldots, n\}$ *such that*

$$K([y_1, \ldots, y_n] \mid [x_1, \ldots, x_n]) \neq K([y_{\pi(1)}, \ldots, y_{\pi(n)}] \mid [x_{\pi(1)}, \ldots, x_{\pi(n)}]). \tag{4.6}$$

Proof. Fix an arbitrary, pairwise different binary strings $x_1, x_2, \cdots \in \{0,1\}^*$. Hereinafter, we fix an arbitrary $n \in \mathbb{N}$. For notational clarify, we avoid superscripts (n) in the definition of the ensuing objects, although all of them are different for different n. For the example of the dataset D, we explicitly write $D^{(n)}$ if we want to clarify n.

Lemma 4.3 guarantees the existence of a string $y \in \{0,1\}^n$ with $K(y \mid x) \geq n$. We construct the ordered dataset $D := \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where y_i is the ith bit in the binary string y.

Without loss of generality, y contains not more 0s than 1s. (Otherwise, the argument is symmetric.) There exists a permutation π over $\{1, \ldots, n\}$ that rearranges the bits in y in such a way that $y_{\pi(1)} \cdots y_{\pi(n)} = 0^j 1^{n-j}$ for some $j \leq \frac{n}{2}$. Accordingly, denote the resulting ordered dataset by

$$D_\pi := \{(x_{\pi(1)}, y_{\pi(1)}), \ldots, (x_{\pi(n)}, y_{\pi(n)})\}. \tag{4.7}$$

First, we show that $K_{JF}(D) \geq n - c_0$ for some constant c_0.

Let $\mathcal{T}_{con}$ be the concatenation Turing Machine that on input of a self-delimited sequence $[z_1, \ldots, z_n]$ outputs the concatenated string $z_1 \cdots z_n$ without delimiters.

Moreover, let V be the Turing Machine that concatenates two functions. On input $\mathrm{enc}(\mathcal{T})p$, V first simulates the universal Turing Machine U on p. After the simulation terminated with output y, V simulates $\mathcal{T}$ on y and returns the output from $\mathcal{T}$.

Let c_{con} and c_v denominate the encoding lengths of these Turing Machines $\mathcal{T}_{con}$ and V. For any string p, p' such that $U(p\big[[x_1, \ldots, x_n], \varepsilon\big]p') = [y_1, \ldots, y_n]$, If there are strings

p, p' that outputs $[y_1, \ldots, y_n]$ when provided to the universal Turing Machine U along with $[x_1, \ldots, x_n]$, then we also have

$$U(\text{enc}(V)\,\text{enc}(\mathcal{T}_{con})p\big[[x_1, \ldots, x_n], \varepsilon\big]p') = y_1 \cdots y_n = y. \tag{4.8}$$

But as $K(y \mid [x_1, \ldots, x_n]) \geq n$, it must also hold that

$$n \leq l(\text{enc}(V)) + l(\text{enc}(\mathcal{T}_{con})) + l(p) + l(p') = c_v + c_{con} + l(p) + l(p'). \tag{4.9}$$

By the definition of conditional Kolmogorov complexity (see Definition 2.3), we conclude that

$$K_{JF}(D) = K([y_1, \ldots, y_n] \mid [x_1, \ldots, x_n]) \geq n - c_0 \tag{4.10}$$

for the constant $c_0 := c_v + c_{con}$.

On the other hand, we now show that $K_{JF}(D_\pi) \leq \log_2(n) + c_1$ for some constant c_1.

To that end, we construct the Turing Machine $\mathcal{T}_1$. $\mathcal{T}_1$ expects its input as $\big[[z_1, \ldots, z_n], x_j\big]$, where x_j is the binary string encoding the natural number $j \in \mathbb{N}$. Using the outer delimiters, $\mathcal{T}_1$ counts the number of samples and saves this number n on an auxiliary tape. Subsequent to that, $\mathcal{T}_1$ checks whether $j \leq n$, and writes $[y_1, \ldots, y_n]$ on the output tape, where the first j y_i are 0 and the remaining y_i are 1.

Let c_1 be the encoding length $l(\text{enc}(\mathcal{T}_1))$. Since $j \leq \frac{n}{2}$, we have

$$\begin{aligned}
K_{JF}(D_\pi) &= K\big([y_{\pi(1)}, \ldots, y_{\pi(n)}] \mid [x_{\pi(1)}, \ldots, x_{\pi(n)}]\big) && (4.11)\\
&\leq l(\text{enc}(\mathcal{T}_1)) + \lceil \log_2(j) \rceil && (4.12)\\
&\leq c_1 + \left\lceil \log_2\left(\frac{n}{2}\right) \right\rceil && (4.13)\\
&\leq c_1 + \log_2(n) + 1 + \underbrace{\log_2\left(\frac{1}{2}\right)}_{=-1} = c_1 + \log_2(n). && (4.14)
\end{aligned}$$

For sufficiently large n, we thus eventually have $n - c_0 > \log_2(n) + c_1$, and therefore $K_{JF}(D^{(n)}) \neq K_{JF}(D^{(n)}_{\pi^{(n)}})$. □

Both these drawbacks suggests that reconciling joint functional information with our notion of *generating* a dataset necessitates to *isolate* the samples. In this manner, we hinder Turing Machines from availing themselves of hardly controllable information leaks *across* samples. The refined definition in the next subsection satisfies not only the two desirable properties above, but allows general, yet nearly optimal, sufficient conditions for learnability of *any* partial computable function.

4.2.2 Isolating functional information

The insight from the last subsection leads our formalisation of the information in a dataset D back to the roots of Kolmogorov complexity. The functional information in a dataset is equivalent to the least information required to describe an algorithm that produces the correct label y_i for each instance x_i in D.

Definition 4.6 (Functional information in a dataset)
Let $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}, n \in \mathbb{N}$ *be an arbitrary finite (unordered) dataset, where* $x_i, y_i \in \{0,1\}^*$. *Then, the functional information in* D *is defined as*

$$K_F(D) := \min_{p \in \{0,1\}^*} \{l(p) \mid U(px_i) = y_i \text{ for all } (x_i, y_i) \in D\}. \tag{4.15}$$

In this vein, the functional information in D bears resemblance to the Kolmogorov complexity of a function $K(f)$ with the difference that in latter, the constraints $U(px) = f(x)$ are imposed for all $x \in \{0,1\}^*$, while the constraints $U(px_i) = y_i$ in the former are merely imposed of the set of instances in D. Beyond D, any such function encoded by the string p may behave arbitrarily.

The definition regards D as an unordered set and is hence not sensitive to the order of the enumeration of samples in D. Moreover, functional information is also monotonously increasing for larger datasets, since larger datasets add new constraints to the string p, while the prior ones remain unaltered. Clearly, this definition also sharpens the bound from Theorem 4.2, as it removes the additive constant. Since we will often refer to this result, it is stated as a Lemma.

Lemma 4.7 (Functional information is bounded by generating function)
Let $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}, n \in \mathbb{N}$ *be an arbitrary finite dataset, where* $x_i, y_i \in \{0,1\}^*$. *For any partial computable function* f *that satisfies* $f(x_i) = y_i$ *for all* $1 \leq i \leq n$, *we have* $K_D(F) \leq K(f)$.

Proof. Given D and f be as above.

The string p with $l(p) = K(f)$ that satisfies the constraints $U(px) = f(x)$ in the Kolmogorov complexity will also satisfy the constraints $U(px_i) = y_i$, because $f(x_i) = y_i$ for all $(x_i, y_i) \in D$. □

At the same time, inconsistency with D also implies a lower bound on $K_F(D)$. If all computable functions f with $K(f) < k$ are inconsistent with D, then we certainly have $K_F(D) \geq k$. By directly availing itself of the definition of Kolmogorov complexity, the functional information $K_F(D)$ hence closely relates to the Kolmogorov complexity of the functions that could have generated D. Figure 4.1 summarizes these benefits of K_F over the joint functional information K_{JF} we started from earlier.

	$K_F(D)$	$K_{JF}(D)$
▷ Consistency implies upper bound	✓	$K_F(D) \leq K(f) + c$
▷ Inconsistency implies lower bound	✓	✗
▷ Monotonicity for supersets	✓	✗
▷ Invariance under sample permutation	✓	✗

Figure 4.1: While functional information satisfies properties that agree with our notion of information, the relaxation in the definition of joint functional information spoils these desirable properties.

Last but not least, this definition also resolves the conundrum that was posed just before the beginning of Subsection 4.2.1 about the maximal possible functional information that datasets generated by f can exhibit. At least for *total* computable functions f, there certainly exist datasets D that remain consistent with f and satisfy $K_F(D) = K(f)$, thus paving the way to sharp learnability conditions later. First, we briefly state this result formally.

Lemma 4.8 (Maximal functional information)
For any total computable function f, there exists a dataset D that remains consistent with f and satisfies $K_F(D) = K(f)$.

Proof. Fix an arbitrary total computable function f. Let $p \in \{0,1\}^*$ be a string with $K(f) = l(p)$ such that $U(px) = f(x)$ for all $x \in \{0,1\}^*$. For any string $\hat{p}$ with $l(\hat{p}) < l(p)$, there exists an $x \in \{0,1\}^*$ such that $U(\hat{p}x) \neq f(x)$. We collect all these samples $(x, f(x))$ into one dataset D^* just as in Subsection 4.2.1. This dataset has a functional information $K_F(D) \geq K(f)$. But by Lemma 4.7, it also holds that $K_F(D) \leq K(f)$, which yields the desired result. □

For partial computable functions however, such a dataset does not exist in general. Consider the example from beginning of Subsection 4.2.1 that juxtaposed the total computable constant function $f'(x) = 0$ with the partial computable function f that is constantly 0 for all strings not longer than m but undefined for all other strings. Since datasets are restricted to the definition range of the data-generating function, any dataset generated by f remains consistent with f'. However, f' arguably has a lower Kolmogorov complexity than f, since this threshold m can be chosen arbitrarily large. Therefore, there is no dataset that renders f its simplest consistent function. But admittedly, such synthetic examples do not matter much in practical cases since we only care about that the machine executes the desired function correctly within its definition range.

4.2.3 Sufficient learnability conditions for any computable function

As announced earlier, the functional information in a dataset D directly allows to upper bound the number of simpler consistent functions that still stand in the way of rendering the true function f the simplest consistent function. This upper bound holds across all prefix-free encodings of Turing Machines and can be understood as a worst-case guarantee for arbitrarily dense encodings.

Theorem 4.9 (Maximum number of alternative simpler consistent functions)
Let $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}, n \in \mathbb{N}$ be an arbitrary, finite dataset with functional information $K_F(D)$. Given an arbitrary partial computable function $f : \{0,1\}^* \to \{0,1\}^*$ that is consistent with D. Then, there are at most $2^{K(f)+1} - 2^{K_F(D)}$ different partial computable functions f' with $K(f') \leq K(f)$ that are consistent with D but are not equivalent to f, $f' \not\equiv f$.

Proof. Given an arbitrary, not necessarily prefix-free, encoding $\text{enc}(\cdot)$ of Turing Machines. Given D with functional information $K_F(D)$ and a consistent partial computable function f as above. By Lemma 4.7, any partial computable function f' that is consistent with D satisfies $K(f') \geq K_F(D)$. By the geometric sum, the number of strings v with length

$K_F(D) \leq v \leq K(f)$ are $2^{K(f)+1}-1-(2^{K_F(D)}-1) = 2^{K(f)+1}-2^{K_F(D)}$. This also upper bounds the number of different partial computable functions f' with $K_F(D) \leq K(f') \leq K(f)$. □

After all, the above condition sheds light on a blind spot that the principle of inferring a simplest consistent function yet exhibits.

In general, even prefix-free encodings enc$(\cdot)$ of Turing Machines can be quite dense in the sense that for each length n, 2^n-1 strings could be theoretically used as the encoding enc$(\mathcal{T})$ of some Turing Machine $\mathcal{T}$ if all shorter strings remain unused. For any partial computable function f with Kolmogorov complexity $K(f)$, there could be theoretically $\Theta(2^{K(f)})$ different partial computable functions f' with the same Kolmogorov complexity $K(f') = K(f)$. If such functions would also remain consistent with a dataset D generated by f, then the choice of a simplest consistent function is grossly ambiguous. In theory, this drawback can in fact be resolved easily since each encoding enc of Turing Machine can be transformed into a sparse encoding enc$'$ that preserves the order of enc and admits only one Turing Machine $\mathcal{T}$ with encoding length m for each $m \in \mathbb{N}$. To corroborate this argument, we order binary strings primarily by their length and secondly by their lexicographic order. The binary strings are thus ordered as $\varepsilon, 0, 1, 00, 01, 10, \ldots$. Given an arbitrary encoding enc and an according universal Turing Machine U, we can construct a prefix-free encoding enc$'$ and a universal Turing Machine U' that works as follows. U' restricts inputs to take the format $1^i0, i \in \mathbb{N}$. Invoked on 1^i0, U' replaces 1^i0 by the i-th binary string in the aforementioned order and simulates U on this string. This way, we are guaranteed that for each length there is only one input string that produces a well-defined output on U'. Therefrom, we could guarantee that $K(f) < K(f')$ for all partial computable functions with $f \not\equiv f'$ and the upper bound in Theorem 4.9 would shrink to a *constant*, as the following Corollary illustrates.

Corollary 4.9.1 (Constant maximum number of alternative simpler consistent functions) *Assume an encoding of Turing Machines such that there are no partial computable functions with $f \not\equiv f'$ and $K(f) = K(f')$. Let D be an arbitrary finite dataset, and f an arbitrary consistent partial computable function f just as in Theorem 4.9. If D has functional information $K_F(D)$, there are at most $K(f) - K_F(D)$ inequivalent partial computable functions $f' \not\equiv f$ with $K(f') \leq K(f)$ that are still consistent with D.*

Proof. Given D and f as above. Any partial computable function $f' \not\equiv f$ with $K(f') \leq K(f)$ must satisfy $K(f') < K(f)$ by the assumption on the encoding. Moreover, any f' that is consistent with D satisfies $K(f') \geq K_F(D)$ because of Lemma 4.7. Since $K(f') \neq K(f'')$ for all $f' \not\equiv f''$, there can be only $K(f) - 1 - (K_F(D) - 1) = K(f) - K_F(D)$ such functions f'. □

Admittedly, such encodings with unambiguous length come a the cost of using only an exponential fraction of the available encoding space. Accordingly, the Kolmogorov complexity of functions $K(f)$ would be exponentially higher than in dense encodings. But in the case of dense encodings, representing the information in a dataset with functional information $K_F(D)$ still has blind spots. To see this, consider a dataset D with maximum functional information $K_F(D) = K(f)$ that renders all other partial computable functions f' with $K(f') < K(f)$ inconsistent achieves the same functional information as an even more discriminative dataset that excludes all f' with $K(f') \leq K(f)$. The above Corollary merely

shows that this blind spot of the simplest consistent function inference principle is not a problem in general, but mainly arises as a trade-off with the density of the encoding space. Moreover, the number of different functions with equal Kolmogorov complexity that remain consistent with a dataset might be low in practice, albeit nothing but a conjecture of the author. Before we proceed, we first compare these generalization conditions in the light of prior works in statistical learning theory, domain generalization, and compression theory.

4.2.4 Comparison to prior learnability conditions

One of the earliest frameworks in statistical learning theory is the probably approximately correct (PAC) learnability of a hypothesis class $\mathcal{H}$ [6, Chapter 3]. Just as in our case, the PAC model relies on the *realizability assumption*, i.e. the true function h is in fact a member of $\mathcal{H}$. Moreover, the classical PAC model assumes that the labels y_i in the dataset D were generated *perfectly* devoid of noise or other corruption by this true function, $y_i = h(x_i)$. This is an assumption we must make equally, since we merely consider functions that are perfectly *consistent* with the samples in D. Finally, all instances x_i are supposed to arise *i.i.d.* from a uniform, unadulterated distribution P. PAC learnability considers how many samples a learning algorithm A like ERM requires to output a hypothesis $\hat{h}$ such that the expected risk $R(\hat{h}) \leq \varepsilon$ with probability $1 - \delta$, where the latter probability both incorporates randomness in the algorithm A and the marginal distribution P from which the instances x_i arise. If there exists such a sample number threshold $m_0(\varepsilon, \delta)$ that is independent of the underlying marginal distribution P and the true function $h \in \mathcal{H}$, then the hypothesis class $\mathcal{H}$ is said to be PAC-learnable. While PAC learnability guarantees an probabilistic upper bound on the expected risk $R(\hat{h}) \leq \varepsilon$, our setting is per se *distribution-free*, and hence does not assume a distribution P to quantify the expected risk function over. Instead, our conditions guarantee the ensuing statement:

> The function $\hat{h}$ with the lowest Kolmogorov complexity that is consistent with the dataset D coincides with the true function h that generated D.

For the PAC model, our framework can therefore only yield sufficient information-theoretic conditions for the *perfect* case $R(\hat{h}) = 0$.

However, the consequence of our statement is also stronger than merely $R(\hat{h}) = 0$. Such a $\hat{h}$ is guaranteed to not only attain optimal expected risk with regard to some distribution P, but to coincide with the true function h on its *entire* definition range. Likewise, the conditions in our statement are stronger than in the PAC model too. Not only must $\hat{h}$ be fully consistent with D, hence achieve empirical risk $\hat{R}(\hat{h}) = 0$, but also achieve the *lowest* Kolmogorov complexity among such consistent functions. As we now show for the hypothesis class of *parity functions*, these additional discriminative constraints can enable learnability guarantees with less samples than in the usual PAC model.

Lemma 4.10 (Sufficient sample size for simplest consistent parity function)
Let $\mathcal{H} = \left\{ f_\beta : \{0,1\}^d \to \{0,1\}, f(x) = \langle \beta, x \rangle \mod 2 \mid \beta \in \{0,1\}^d \right\}$ *be the class of parity functions over d-dimensional binary inputs. Let* $P = \mathrm{Ber}\left(\frac{1}{2}\right)^{\otimes d}$ *be the distribution over strings in* $\{0,1\}^d$ *where each component is distributed independently according to* $\mathrm{Ber}\left(\frac{1}{2}\right)$.

For any $f_\beta \in \mathcal{H}$ and any $0 \leq k \leq d$, if there are $0 \leq i \leq 2^d - 1$ other $f'_\beta \in \mathcal{H}$ with $K(f'_\beta) \leq K(f_\beta)$, then with probability at least $1 - i \cdot \left(\frac{1}{2^k}\right)$, datasets D comprised of k samples $x_i \overset{i.i.d}{\sim} P$, $y_i = f(x_i)$, suffice to render f_β the simplest consistent function in $\mathcal{H}$ with D.

Proof. Given $\mathcal{H}$ and P as above. Let $f_\beta \in \mathcal{H}$ be an arbitrary parity function such that i other parity functions $f'_\beta \in \mathcal{H}$ achieve $K(f'_\beta) \leq K(f_\beta)$, $0 \leq i \leq 2^d - 1$.

For any $f_\beta, f'_\beta \in \mathcal{H}$ with $f_\beta \not\equiv f'_\beta$, it is well-known (cf. [26, Section 1.3]) that

$$\Pr_{x \sim P}\left[f_\beta(x) = f'_\beta(x)\right] = \frac{1}{2}. \tag{4.16}$$

Let $0 \leq k \leq d$ be arbitrary.

Hereinafter, we denominate the dataset $D := \{(x_i, f_\beta(x_i)) \mid i = 1, \ldots, k\}$.

Then, by union bound, the failure probability β satisfies

$$\begin{aligned}
\beta :&= \Pr_{x_i \overset{i.i.d}{\sim} P}\left[f_\beta \text{ is not the simplest consistent function with } D\right] && (4.17)\\
&= \Pr_{x_i \overset{i.i.d}{\sim} P}\left[\text{There exists another consistent } f'_\beta \in \mathcal{H} \text{ with } K(f'_\beta) \leq K(f_\beta)\right] && (4.18)\\
&\leq \sum_{\substack{f'_\beta \in \mathcal{H}, f_\beta \not\equiv f'_\beta \\ K(f'_\beta) \leq K(f_\beta)}} \Pr_{x_i \overset{i.i.d}{\sim} P}\left[f_\beta(x_i) = f'_\beta(x_i) \text{ for all } i = 1, \ldots, k\right] && (4.19)\\
&\overset{(4.16)}{=} \sum_{\substack{f'_\beta \in \mathcal{H}, f_\beta \not\equiv f'_\beta \\ K(f'_\beta) \leq K(f_\beta)}} \left(\frac{1}{2}\right)^j. && (4.20)\\
&= i \cdot \left(\frac{1}{2}\right)^j, && (4.21)
\end{aligned}$$

which concludes our proof. □

The required sample size therefore now additionally depends on the Kolmogorov complexity $K(h)$ of the true function h, and can undercut the typical required sample thresholds if $K(h)$ is low.

Vice versa, $R(\hat{h}) = 0$ alone does not necessarily imply that $\hat{h}$ is the simplest consistent function with the dataset D. Obviously, there might be simpler consistent partial computable functions outside of $\mathcal{H}$. But even within $\mathcal{H}$, there could be simpler consistent functions if the support of P does not cover the entire definition range of functions in $\mathcal{H}$.

Let us now exemplify how the discriminative power of Kolmogorov complexity carves the way to learn functions for which no learnability guarantees have been formulated in the PAC model so far. We consider the problem of learning the *prime number* decision function

$$f_{\mathbb{P}} : \mathbb{N} \to \{0, 1\}, f_{\mathbb{P}}(n) := \begin{cases} 1, & n \in \mathbb{P} \\ 0, & n \notin \mathbb{P} \end{cases} \tag{4.22}$$

in the hypothesis class of *all* computable decision functions over the natural numbers. No matter how large the finite dataset is, there will always be infinitely many computable decision functions that perfectly fit the dataset and hence achieve the optimal empirical risk.

However, enumerating the first m prime numbers will eventually guarantee that the simplest consistent decision function is indeed $\mathbb{1}_{\mathbb{P}}$.

Lemma 4.11 (Teaching prime numbers by enumerating them)
Let $\mathbb{1}_{\mathbb{P}}$ denominate the prime number decision function as in Equation 4.22. There exists an m_0 such that any dataset D that contains $(n, \mathbb{1}_{\mathbb{P}}(n))$ for all $n \leq m_0$ renders $\mathbb{1}_{\mathbb{P}}$ the simplest consistent function with D among all computable decision functions over $\mathbb{N}$.

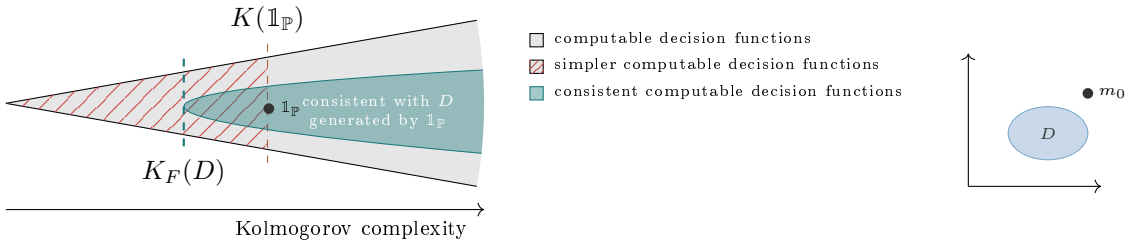
Proof. Functions $f : \mathbb{N} \to \{0,1\}$ relate to functions over binary strings $f : \{0,1\}^* : \{0,1\}$ in the natural way (cf. Definition 2.2). For any decision function $f' : \mathbb{N} \to \{0,1\}$ that is not equivalent to $\mathbb{1}_{\mathbb{P}}$, there exists an $n_{f'} \in \mathbb{N}$ with $\mathbb{1}_{\mathbb{P}}(n_{f'}) \neq f'(n_{f'})$. Since $\mathbb{1}_{\mathbb{P}}$ has a finite Kolmogorov complexity, there are only finitely many such functions f' with $K(f') \leq K(f_{\mathbb{P}})$. The maximum $m_0 := \max_{K(f') \leq K(\mathbb{1}_{\mathbb{P}})} \left(n_{f'}\right)$ is therefore well-defined.

Any dataset D that contains $(n, \mathbb{1}_{\mathbb{P}}(n))$ for all $n \leq m_0$ will hence render each such f' inconsistent, leaving $\mathbb{1}_{\mathbb{P}}$ as the unique simplest consistent function with $D^{(m)}$. □

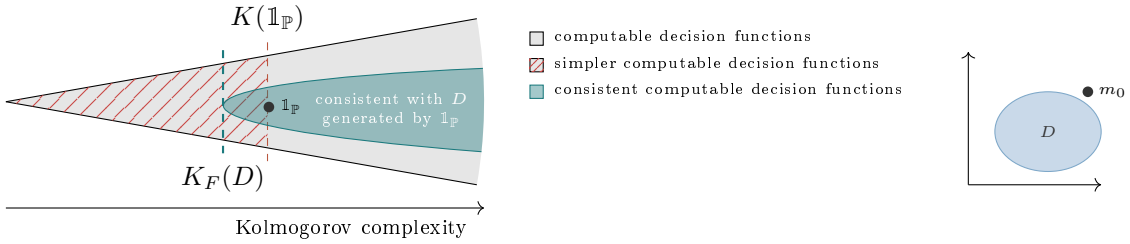
Figure 4.2 illustrates the above learnability result for prime numbers by means of the functional information in D. The more positive and negative examples of prime numbers D contains, the higher is its functional information $K_F(D)$. When all instances below some threshold m_0 are covered (cf. Figure 4.2c), the functional information coincides with the Kolmogorov complexity of $\mathbb{1}_{\mathbb{P}}$, rendering $\mathbb{1}_{\mathbb{P}}$ the unique simplest consistent function.

Since our setup assumes noise-free samples, it is however not applicable to the extended settings of the *agnostic* PAC model or uniform convergence where the labels are not strictly generated from the true function h but follow a distribution that can integrate noise or corruptions [6, Chapter 4]. In classical learning theory, VC dimension or the Rademacher complexity are usually adopted to derive sufficient sample sizes for the PAC learnability or uniform convergence of hypothesis classes [6, Chapters 6, 26]. In terms of both VC dimension and Rademacher complexity, the class of partial computable functions does however not obtain a finite complexity. Even under the *i.i.d.* assumption, these complexity quantities hence do not yield learnability conditions for this overarching hypothesis class, while Kolmogorov complexity assures that at least each total computable function can be learned with finitely many samples.

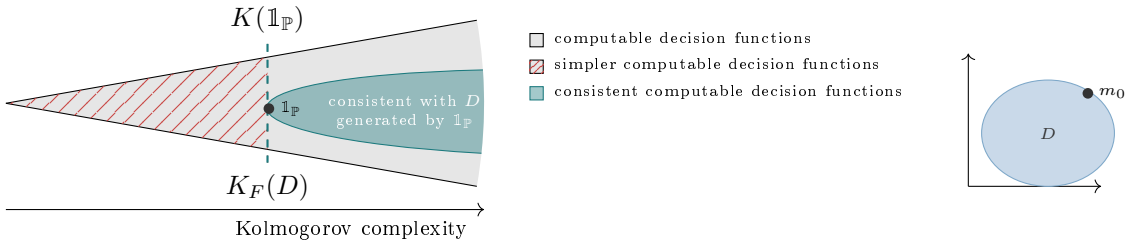
Moving on with compression theory, a recent advance first provided tight generalization guarantees for the probability that a dataset generated by *i.i.d.* samples *changes* its optimal compression [4]. In their work, compression schemes were algorithms that compressed larger datasets into smaller ones. Sufficiently informative datasets D in [4] might at some point be compressed to the same smaller dataset as any of their supersets, because only redundant information is added. At such a point, guarantees about the in-distribution generalization of predictors that were trained on D by appropriate learning algorithms might become quite tight, as was shown later in [3]. This directly relates to the implications of the functional information in D. If a dataset D generated by some function f already contains maximal information about f and hence satisfies $K_D(F) = K(f)$, larger datasets $\hat{D} \supset D$ will not achieve a longer compression. First of all, the difference with this work lies in our more general notion of compression by virtue of Kolmogorov complexity. But as for PAC learning, our statements are distribution-free and do not rest on the *i.i.d.* assumption.



(a) Yet too uninformative dataset D that leaves more simple decision functions consistent.



(b) More informative samples were included into D, such that more simpler decision functions are rendered inconsistent. Accordingly, the functional information $K_F(D)$ increased.



(c) Sufficiently many samples are included in D. The functional information has attained its maximum $K_F(D) = K\big(\mathbb{1}_{\mathbb{P}}\big)$, rendering any simpler decision function inconsistent.

Figure 4.2: Enumerating the first k prime numbers up to the threshold m_0 from the proof of Lemma 4.11 eventually renders the true prime number decision function the simplest consistent function among all computable decision functions over $\mathbb{N}$.

Our work therefore adopts the problem setting of domain generalization, where theoretical guarantees were however usually conducted for exemplary classes such as linear predictors or classifiers, and still draw upon strong assumptions about the support of the training distributions [1], [2].

All in all, the functional information of a dataset accurately captures how much information it conveys about the functions that could have generated it. Not only does it allow to specify information-theoretic learnability conditions for any partial computable function in hypothesis classes as large as the partial computable functions. The next section demonstrates how functional information can also realise a simplicity bias in practice.

4.3 Functional information as a simplicity bias in practice

Because it culminates in the same paradox as the halting problem, Kolmogorov complexity and thus also functional information are incomputable in general. However, there are computable alternatives such as compression algorithms that approximate Kolmogorov complexity. For that reason, this section first presents how functional information can realise a simplicity bias within *any* hypothesis class. Later, we will then address how functional information could be efficiently approximated in practice.

For any hypothesis class $\mathcal{H}$, any dataset D yields some simplest consistent functions $f \in \mathcal{H}$. Lemma 4.7 guarantees $K_F(D) \leq K(f)$, no matter how much D is extended by samples from such a function f. If we therefore generate some pseudo-samples $(x'_i, f(x'_i))$ by evaluating f on some unlabeled instances x'_i that do not occur in D, and extend D by these samples to a larger dataset $\hat{D}$, we can reevaluate $K_F(\hat{D})$ to see how much the functional information has *increased*. The more informative $\hat{D}$, the more will $K_F(\hat{D})$ approach $K(f)$. Even for the simplest consistent functions $f \in \mathcal{H}$, $K_F(\hat{D})$ might hence increase a little because there might be simpler consistent computable functions *outside* of $\mathcal{H}$. But if f instead is not a simplest consistent function but rather $K(f) \gg K_F(D)$, $K_F\hat{D}$ will arguably increase sharply.

For that reason, the closer $K_F(\hat{D})$ remains to $K_F(D)$, the closer might $K(f)$ be to the simplest consistent function. The simplest consistent functions f within $\mathcal{H}$ will arguably receive the smallest increase and thus the strongest favour.

Algorithm 1 Simplicity bias with functional information $K_F(D)$.

Given:

- A labeled dataset $D := \{(x_1, y_1), \ldots, (x_n, y_n)\}$,
- A set of unlabeled instances $D' := \{x'_1, \ldots, x'_m\}$,
- A function f consistent with D,
- An oracle for the functional information $K_F(\cdot)$ of datasets.

1: Compute $K_F(D)$.
2: Evaluate unlabeled instances on h:

   (a) Select some instances $x'_{i_1}, \ldots, x'_{i_K}$ from D'.

   (b) Compute $y'_{i_j} = f(x'_{i_j})$ for $1 \leq j \leq K$.

   (c) Collect them together with the true samples into $\hat{D} := D \cup \{(x'_{i_1}, y'_{i_1}), \ldots, (x'_{i_K}, y'_{i_K})\}$.

3: Compute $K_F(\hat{D})$. {The closer $K_F(\hat{D})$ is to $K_F(D)$, the closer is f to the simplest consistent functions.)}
4: **return** $K_F(D), K_F(\hat{D})$

Before we discuss the drawbacks of this algorithm, let us first consider some benefits.

Apart from the oracle for $K_F(D)$, the above procedure merely requires labeled and unlabeled samples and a hypothesis h. It can therefore be applied to any model class. Moreover, it takes advantage of the massive amount of unlabeled data in the internet, while labeled data is costly to generate. The number of unlabeled instance m for the dataset $\hat{D}$ can therefore be chosen quite high, since the evaluation of f is mostly computationally feasible too. The more samples $\hat{D}$ incorporates, the more significant is $K_F(\hat{D})$ as an approximation of $K(f)$.

However, there are still some fundamental hindrances that obstruct the way to implement this algorithm in practice. First and foremost, Kolmogorov complexity is incomputable in a hard sense. That is, for any infinite set of strings, there is no partial computable function that correctly outputs their Kolmogorov complexity [27]. Because its incomputability revolves around the halting problem, Levin devised an extended version of Kolmogorov complexity which does not only take into account the length of a program, but also its running time [19]. That way, extremely short programs that yet never halt are not attributed a finite value anymore. A Turing Machine could hence incrementally simulate the first p strings $x_1, \ldots, x_p$ on the universal Turing Machine for m time steps, where m starts at $\log_2(p)$ and decreases linearly with the length of x_i, and check if one of them returns the desired string. The running time of this exhaustive procedure is however far from feasible.

As an alternative, lossless compression algorithms such as the widely employed Lempel-Ziv-Welch algorithm [28] have been proposed to approximate the Kolmogorov complexity of strings in practice [23, p. 696]. Given a string x, such algorithms yield both a compression p and a method to reconstruct x from p. In the next, final chapter, we however substantiate that the Lempel-Ziv-Welch algorithm can not yield even approximate guarantees about the order between the Kolmogorov complexity $K(v), K(w)$ of two strings v, w, no matter what exponential, multiplicative, and additive approximation factors we allow. Such approximation ratios would however be necessary if we desire a realisation of a simplicity bias like in Algorithm 1 that also yields some guarantees about the approximation accuracy of the functional information $K_F(D)$. Therefore, the final chapter corroborates that practical realisations of the above simplicity bias are yet out of reach. But at the same time, it points up how compression and learning are interrelated, and encourages advancements in the study of algorithms that detect and compress regularities in data for the sake of learning simple functions.

Chapter 5

Compression Algorithms against Kolmogorov Complexity

In Chapter 4, we formally established how Kolmogorov complexity smooths the way to quantify the information that a dataset of samples conveys about the functions that could have generated it. Coined *functional information*, this quantity allowed to derive general, yet nearly optimal, sufficient information-theoretic learnability conditions for virtually any partial computable function. Moreover, we discussed how functional information not only helps to determine the relevant information content in a dataset, but also how it could realise a simplicity bias in practice that favours functions with lower Kolmogorov complexity.

But because it would evoke the same paradox as the halting problem, Kolmogorov complexity is not uniformly computable in general [25]. Notwithstanding that, heuristics could yield feasible approximations of this measure in practice. Because we want to employ such a approximation algorithm to *compare* the functional information of different objects, a necessary requirement for such an algorithm A would be to yield at least some rough guarantee about the order between two strings v, w in terms of their Kolmogorov complexity. That is, there should be some exponential, multiplicative, and additive approximation offsets a, b, k such that for any strings $v, w \in \{0,1\}^*$, if $A(v) \geq \exp_2^{(k)}\big(a \cdot A(w) + b\big)$, then $K(v) \geq K(w)$. Here, $\exp_2^{(k)}$ designates the k-wise concatenation of the exponential function with basis 2, just as in Definition 3.4.

To date, compression algorithms are typically suggested to yield such a feasible approximation of Kolmogorov complexity. Most importantly, several compression algorithms based on the classic Lempel-Ziv-Welch (LZW) algorithm [28] have been employed to cluster data of various formats [5]. However, this chapter proves that for any encoding of Turing Machines, the Lempel-Ziv-Welch algorithm can not yield any approximate guarantee about the order of the Kolmogorov complexity of two strings, rendering it at least theoretically undesirable for our purpose of approximating functional information. In Theorem 5.7, we prove that such approximation offsets a, b, k as formulated above do not exist for the LZW algorithm. This chapter therefore points up that practical implementations of a simplicity bias that yield at least approximate guarantees still need advances in compression algorithms.

But to that end, we first introduce the LZW algorithm in Section 5.1, and state some required results about the compressibility of strings in Section 5.2. In Section 5.3, we eventually culminate in the aforementioned Theorem.

5.1 The Lempel-Ziv-Welch algorithm

However, we substantiate that such methods still exhibit two essential shortcomings for our purpose to approximate functional information. To that end, we first present how the Lempel-Ziv-Welch algorithm would actually compress binary strings. Some adjustments have been made for the ease of exposition, but they do not affect our later argument. The

Algorithm 2 Lempel-Ziv-Welch algorithm [28].

Given: The binary alphabet $\Sigma = \{0, 1\}$ and a non-empty string $x \in \{0, 1\}^*$

1: Instantiate a bidirectional key-value directory T to save patterns.

(a) T saves tuples (id, v), where $v \in \{0, 1\}^*$ is a pattern, and $\mathrm{id} \in \{0, 1\}^*$ is its identifier.

(b) We write $T(v) = \begin{cases} \mathrm{id}, & (\mathrm{id}, v) \in T \\ \perp, & (\mathrm{id}, v) \notin T \text{ for any } \mathrm{id} \in \mathbb{N}. \end{cases}$

(c) When a pattern v is *added* to T, the tuple $([|T|, \varepsilon], v)$ is added to T, where $|T|$ is the length of the directory T and $[|T|, \varepsilon]$ its self-delimiting encoding. Once added, elements are not deleted. This way, the identifier is always unique.

(d) We add the symbols $0, 1 \in \Sigma$ as the elementary patterns to T. Therefore, it now holds that $(0, 100), (1, 101) \in T$, and $|T| = 2$.

2: Instantiate the empty output string $p := \varepsilon$ that represents the compression of x.

3: **While** x is not in T empty:

(i) Incrementally determine the shortest prefix $x_1 x_2 \cdots x_k$ of x, $x_i \in \{0, 1\}$, that is not in T.

(ii) Append the identifier $T(x_1 x_2 \cdots x_{k-1})$ to p.

(iii) Add the new pattern $x_1 x_2 \cdots x_k$ to T.

(iv) Remove the prefix $x_1 x_2 \cdots x_{k-1}$ from x.

4: Append the identifier $T(x)$ to p.

5: **return** p, T

algorithm above merely looks for repetitions in the symbols sequences of a string. The more often certain sequences of symbols occur in x, the smaller will the compression p be. However, this algorithm does not go further and consider the patterns *across* the stored strings themselves. Strings that are generated from quite simple recursive algorithm do however exhibit precisely such patterns, which the algorithm does yet not draw upon.

For this reason, the maximum compression ratio — namely the ratio between the length of n and its compression p — is bounded by a polynomial in the length of n, as the following Lemma ascertains.

Lemma 5.1 (Bounded compression ratio Lempel-Ziv-Welch)
Let $x \in \{0, 1\}^*$ be an arbitrary string. Then the Lempel-Ziv-Welch compression p that is output from Algorithm 2 satisfies $l(p) \geq \sqrt{l(x)}$.

Proof. For an arbitrary $n \in \mathbb{N}$, consider the string 1^n, that is the concatenation of n 1s.

We show that this string achieves the optimal compression length that strings of length n could possibly achieve.

In each iteration, the prefix $x_1x_2\cdots x_k$ that is added to T can at most be one symbol longer than the longest string in T. For if that was not the case, it holds that $k \geq 3$, because 0 and 1 are in T from the beginning. But then $x_1x_2\cdots x_{k-1}$ is already not in T, which contradicts the assumption. For that reason, the optimal compression length is achieved if in each iteration, the prefix that is added achieves this maximal length. Because if such an $x_1x_2\cdots x_k$ is added to T, then $x_1x_2\cdots x_{k-1}$ is logarithmically compressed to $T(x_1x_2\cdots x_{k-1})$ and appended to p. The longer this compressed sequence $x_1x_2\cdots x_{k-1}$ is, the shorter will p be.

Now, we show that $x = 1^n$ achieves this optimal compression property. If $n = 1$, then $p = 1^1 = 1$, which is optimal. Below, we hence assume $n \geq 2$. In the first iteration $i = 1$, 1 is added to p and 11 is added to T. Therefore, T contains a pattern string v of length $i + 1$ after iteration $i = 1$. Now, assume that after some iteration $i = j$, T contains all strings $1, 11, \ldots, 1^{j+1}$. Then, in iteration $j + 1$, if the remaining string x is not longer than $j + 1$, $T(x)$ is added to p, which is optimal since all prefixes $x_1x_2\cdots x_{k-1}$ that were added to p had maximum length. Otherwise, if the remaining string x is longer than $j + 1$, then 1^{j+2} is the shortest prefix that is not in T, which is also optimal because there could be no such prefix that is longer.

Now, let us define a lower bound on the length of p. Let k be the smallest natural number such that $\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \geq n$. Then, Algorithm 2 terminates after k iterations, because in each iteration i that is not the last one, i symbols are removed from x.

The string $T(x_1x_2\cdots x_{k-1})$ that is appended to p in each such iteration obviously has a length of at least 1. Because $\frac{(k+1)^2}{2} \geq n$, we also have $k \geq \sqrt{2n-1}$. Finally, since $n \geq 2$, $2n - 1 = n + (n - 1) \geq n$. By the monotonicity of the square root function, we therefore conclude

$$|p| \geq k \geq \sqrt{2n-1} \geq \sqrt{n} = \sqrt{(l(x))}. \tag{5.1}$$

□

On the other hand, there is of course also an upper bound on the length of p. Although the following upper bound is grossly conservative, it is sufficient for our needs and all the more simple to show.

Lemma 5.2 (LZW Compression length upper bound)
Let $x \in \{0,1\}^*$ be an arbitrary string. Then the Lempel-Ziv-Welch compression p that is output from Algorithm 2 satisfies $l(p) \leq 3 \cdot \big(l(x)\big)^2$.

Proof. Let x be an arbitrary string of length n. Because 0 and 1 are in T from the beginning, the shortest prefix $x_1x_2\cdots x_k$ that is not in T must have length $k \geq 2$. Therefore, at least one symbol is removed from x in each iteration. Consequently, the LZW algorithm runs for at most $n - 1$ iterations, since after $n - 1$ iterations, the remaining x will at most comprise one symbol, which is certainly in T. By the same argument as in the proof of Lemma 5.1, T can contain at most $n + 1$ strings after iteration $n - 1$, since 0 and 1 are added from the beginning. All in all, p can at most contain n compressions of strings whose identifying

natural number is at most $n + 1 - 1 = n$. By the additional length of the self-delimiting encoding, we therefore conclude

$$l(p) = n \cdot (2\lceil \log_2(n) \rceil + 1) \leq n \cdot (2n + 1) = 2n^2 + n \leq 3n^2. \tag{5.2}$$

□

In contrast to the quadratic upper bound on the compression ratio of the Lempel-Ziv-Welch algorithm, the compression ratio that Turing Machines achieve are arbitrarily large, as the next Section illustrates.

5.2 Compressibility of strings

In the following, we fix $\Sigma = \{0, 1\}$ as before. This section ascertains both the existence of incompressible strings and arbitrarily compressible strings, which smooths the way to Theorem 5.7 in the next section. In the same vein as Lemma 4.3, we first argue that there must exist an incompressible string for each length n.

Lemma 5.3 (Incompressible Strings)
For any $n \in \mathbb{N}$, there exists a string $v_n \in \{0,1\}^n$ with $K(v_n) \geq n$.

Proof. Let $n \in \mathbb{N}$ be arbitrary. There are 2^n strings of length n, but only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ strings that are shorter than n. By the pigeonhole principle, there must be at least one $v \in \{0,1\}^n$ such that there exists no program p of length $l(p) < n$ with $U(p) = v$. For this v, we have $K(v) \geq n$. □

In general, since the number of strings grows exponentially in their length, most strings are almost completely incompressible [23]. Conversely, there are however also a few strings that are arbitrarily compressible. While the Lempel-Ziv-Welch algorithm could not compress strings $z_n = 1^n$ beyond a quadratic factor, there is a Turing Machine that achieves a compression ratio that scales faster than any polynomial, and even faster than any fixed concatenation of the exponential function, as the following Lemma certifies.

Lemma 5.4 (Arbitrarily Compressible Strings)
Let $\mathrm{enc}(\cdot)$ *be an arbitrary, prefix-free encoding of Turing Machines. Let* $\exp_2^{(m)}$ *denote the m-wise recursive concatenation of* $\exp_2(x) := 2^x$ *as in Definition 3.4. There exists a constant $c \in \mathbb{N}$ such that: For every $n \in \mathbb{N}$, there exists a string $w_n \in \{0,1\}$ of length $l(w_n) = \exp_2^{(n)}(1)$ with Kolmogorov complexity $K(w_n) \leq \log_2(n) + c$.*

Proof. Given everything as above, we construct a Turing Machine $\mathcal{T}$ that on input x_n outputs the the string $w_n = (1)^{\exp_2^{(n)}(1)}$, which is the $\exp_2^{(n)}(1)$-wise concatenation of 1. Besides an input tape and an output tape, $\mathcal{T}$ uses one auxiliary tape.

First of all, $\mathcal{T}$ asserts that the input $x \in \{0,1\}^*$ correctly encodes a natural number n_x as by Definition 2.2. Then, $\mathcal{T}$ writes a 1 on the output tape and repeats the following procedure until the input tape only holds the empty string ε.

- Remove all non-blank symbols on the auxiliary tape.

- Copy the non-blank symbol sequence x on the output tape to the auxiliary tape.
- Interpret x as the natural number $n_x \in \mathbb{N}$. Compute $x_{2^{n_x}}$ and write it on the output tape.
- Subtract the number on the in put tape by 1.

After this procedure, $\mathcal{T}$ copies x once more to the auxiliary tape and eradicates the output tape blank. Now, it repeats the following procedure as long as the string on the auxiliary tape is not blank, and terminates afterwards.

- Write a 1 on the output tape.
- Subtract the string on the auxiliary tape by 1.

If $\mathcal{T}$ is invoked on x_n for any $n \in \mathbb{N}$, the string on the output tape after the first iterative procedure encodes $\exp_2^{(n)}(1)$. After the second iterative procedure, we therefore have $(1)^{\exp_2^{(n)}(1)}$ written to the output tape as desired, and the universal Turing Machine U from Section 2.2 equally yields $(1)^{\exp_2^{(n)}(1)}$ when invoked on $\mathrm{enc}(\mathcal{T})x_n$.

For that reason, choosing the constant as $c = l(\mathrm{enc}(\mathcal{T})) + 1$, the Kolmogorov complexity of this string is bounded by

$$K((1)^{\exp_2^{(n)}(1)}) = K(\mathcal{T}(x_n)) = K\big(U(\mathrm{enc}(\mathcal{T})x_n)\big) \leq l(\mathrm{enc}(\mathcal{T})) + l(x_n) \leq c + \log_2(n). \quad (5.3)$$

Finally, this string has length $\exp_2^{(n)}(1)$, which concludes our proof. □

These compressibility results already reveal the fundamental reason why the Lempel-Ziv-Welch algorithm and Kolmogorov complexity induce disparate orders on binary strings $v \in \{0,1\}^*$. For sufficiently large n, there is a string w_n of length $\exp_2^{(n)}(1)$ whose Kolmogorov complexity is smaller than that of an incompressible string v_n of length n. However, the compression length of the LZW algorithm cannot exceed a polynomial factor, rendering the compression of w_n far larger than that of v_n. Since the exponential tower function $\exp_2^{(n)}(1)$ grows faster than any fixed concatenation of exponential functions, Theorem 5.7 in the next section will demonstrate that this inconsistency even holds if we make arbitrarily large allowances for exponential, multiplicative, and additive approximation offsets.

5.3 Unbounded order inconsistencies

On the one hand, Lemma 5.1 demonstrated that the maximum compression ratio of the LZW algorithm merely scales quadratically with growing length. Conversely, Lemma 5.4 demonstrated that strings such as $(1)^{\exp_2^{(n)}(1)}$ are arbitrarily compressible for increasing n, because their Kolmogorov complexity scales only logarithmically for increasing n. No matter that prefix-free encoding of Turing Machines we choose, this discrepancy allows us to conclude that for any additive and multiplicative, and even exponential offset constants $a, b, k \in \mathbb{N}$, there are infinitely many strings v, w such that

$$l\big(A(v)\big) \geq \exp_2^{(k)}\big(a \cdot l\big(A(w)\big) + b\big), \text{ but } K(v) < K(w), \quad (5.4)$$

where A is the LZW compression p from Algorithm 2.

But first and foremost, we concretize the dominance of exponential growth over polynomial growth by means of elementary calculus. The proofs of the following two results are omitted here and laid out in Appendix A.1 instead.

Lemma 5.5 (Log-Linear Inequality with Additive Constant)
Let $a \geq 1, b \geq 0$ be arbitrary real numbers. For any real number $x > 2^{4(a+b)}$, it holds that $a \log_2(x) + b < x$.

The second Lemma ascertains the growth of the recursive exponentiation of a constant over linear functions.

Lemma 5.6 (Special Cascade Exponential Equality)
Let $\exp_2^{(m)}$ denote the m-wise recursive concatenation of $\exp_2(x) := 2^x$ as in Definition 3.4. Let a, b be an arbitrary real scalars with $a \geq 1, b \geq 0$, and $k \in \mathbb{N}$. For any natural number $n \in \mathbb{N}$ with $n > 2^{k+2} \cdot (a + b)$, it holds that

$$\exp_2^{(n-k)}(1) > a \cdot n + b. \tag{5.5}$$

The aforementioned order inconsistencies more generally hold for any compression algorithm A whose compression ratio is bounded by a fixed concatenation of exponential functions. Therefore, even the aforementioned suggested proposal to recursively compress the pattern string dictionary would not resolve this issue, since the length of p would still scale at least logarithmically with the length of x. Because it even streamlines some technical details in the proof, we therefore prove this more general result in the following Theorem.

Theorem 5.7 (Unbounded compression order inconsistencies)
Let $\mathrm{enc}(\cdot)$ be an arbitrary, prefix-free encoding of Turing Machines. Let A be an arbitrary compression algorithm such that there exist a length threshold n_0, an exponential compression ratio offset $m_0 \in \mathbb{N}$, and a polynomial compression upper bound $p(x) = \sum_{i=0}^{k_0} a_i x^i$ such that for all inputs x with length $l(x) \geq n_0$, it holds that $\log_2^{(m_0)}\big(l(x)\big) \leq l\big(A(x)\big) \leq p\big(l(x)\big)$.

Then, for any real numbers $a \geq 1, b \geq 0$, and any $k \in \mathbb{N}$, there are strings $v, w \in \{0,1\}^*$, such that

$$l\big(A(v)\big) \geq \exp_2^{(k)}\big(a \cdot l\big(A(w)\big) + b\big), \text{ but } K(v) < K(w). \tag{5.6}$$

Proof. Given $\mathrm{enc}(\cdot)$, the compression algorithm A with $n_0, m_0, p(x) = \sum_{i=0}^{k_0} a_i x^i$, and the constants a, b, k as above. Let c be the constant from Lemma 5.4. Fix an arbitrary $n \in \mathbb{N}$ with $n > \max\Big(2^{4(1+c)}, n_0, 2^{k+m_0+3} \cdot \big(k_0 + a \cdot \sum_{i=0}^{k_0} a_i + b\big)\Big)$.

On the one hand, Lemma 5.3 guarantees the existence of an incompressible string $v_n \in \{0,1\}^*$ with $K(v_n) \geq n$. On the other hand, we consider the compressible string $w_n = 1^{\exp_2^{(n)}(1)}$ from Lemma 5.4 with $K(z_n) \leq \log_2(n) + c$.

1. Kolmogorov Complexity Inequality

Because $n > 2^{4(1+c)}$, Lemma 5.5 asserts that

$$K(v_n) \overset{5.3}{\geq} n \overset{5.5}{>} \log_2(n) + c \overset{5.4}{\geq} K(w_n). \tag{5.7}$$

2. Compression Inequality

Since $n > n_0$, the compression ratio bound of A ensures that

$$l\big(A(v_n)\big) \geq \log_2^{(m_0)}\big(l(v_n)\big) = \log_2^{(m_0)}\big(\exp_2^{(n)}(1)\big) = \exp_2^{(n-m_0)}(1). \tag{5.8}$$

Because $2^n > n$ and $n \geq 1$, we also have

$$2^{k_0 n + a\cdot\sum_{i=0}^{k_0} a_i + b} = \big(2^{a\cdot\sum_{i=0}^{k_0} a_i + b}\big)\big(2^n\big)^{k_0} > \Big(a\cdot\sum_{i=0}^{k_0} a_i + b\Big)n^{k_0} \geq a\cdot\Big(\sum_{i=0}^{k_0} a_i n^i\Big) + b. \tag{5.9}$$

Because $n > 2^{k+m_0+3}\cdot\big(k_0 + a\cdot\sum_{i=0}^{k_0} a_i + b\big)$, Lemma 5.6 also guarantees that

$$\exp_2^{(n-k-m_0-1)}(1) > k_0 n + a\cdot\sum_{i=0}^{k_0} a_i + b. \tag{5.10}$$

By the strict monotonicity of $\exp_2$, we therefore conclude our proof with

$$\begin{aligned}
l\big(A(v_n)\big) &\overset{(5.8)}{\geq} \exp_2^{(n-m_0)}(1) && (5.11)\\
&= \exp_2^{(k)}\big(\exp_2^{(n-k-m_0)}(1)\big) && (5.12)\\
&= \exp_2^{(k)}\Big(\exp_2\big(\exp_2^{(n-k-m_0-1)}(1)\big)\Big) && (5.13)\\
&\overset{(5.10)}{>} \exp_2^{(k)}\Big(\exp_2\big(k_0 n + a\cdot\sum_{i=0}^{k_0} a_i + b\big)\Big) && (5.14)\\
&\overset{(5.9)}{>} \exp_2^{(k)}\Big(a\cdot\big(\sum_{i=0}^{k_0} a_i n^i\big) + b\Big) && (5.15)\\
&\geq \exp_2^{(k)}\Big(a\cdot l\big(A(w_n)\big) + b\Big). && (5.16)
\end{aligned}$$

□

The Lempel-Ziv-Welch compression from Algorithm 2 certainly satisfies these bounded compression ratio conditions. On the one side, Lemma 5.2 allows us to choose $p(x) = 3x^2$ as an upper bound on the compression length. On the other side, since any $n > 2^{16}$ satisfies $\sqrt{n} > \log_2(n)$ by virtue of Lemma 5.5, the compression length lower bound in Lemma 5.1 renders $\log_2(n)$ a sufficient lower bound with $n_0 = 2^{16}$. But even more modern compression algorithms like the DEFLATE algorithm [29], which is widely adopted in file compression, or the Brotli compression [30], which recently exhibited experimental superiority over the so far unchallenged DEFLATE algorithm [31], would not yield satisfactory approximations of the Kolmogorov complexity order between two strings. Furthermore, although it sufficed to neglect it for the above proof, the length of the additional pattern dictionary T must also be taken into account when quantifying the compression ratio, because x cannot be restored from p without T.

But even if compression algorithms were to yield such rough guarantees, the implementation of functional information would still require some further adjustments to such algorithms. While compression algorithms usually compress data ad hoc without prior knowledge, functional information instead conditions on the information of the given instances x_i.

In this vein, we merely require to find the shortest prefix p such that $U(px_i) = y_i$. Moreover, functional information imposes *multiple* constraints on p, because p needs to satisfy $U(px_i) = y_i$ for all samples (x_i, y_i) in the dataset D. The larger D becomes, the harder will this constrained optimization problem appear too.

If the second requirements renders too difficult to realise, it might hence also be acceptable to work with the *joint* functional information $K_{JF}(D)$, as this definition simply concatenates the labels and instances to one long string. Although this comes at the cost of spoiling theoretical guarantees, it might yield satisfactory approximations in practice.

This discussion stresses how the problem of learning a simplest consistent function is deeply intertwined with compression. An algorithm that would be able to compress the causal mechanism that determines y_i given x_i into a small string p directly tackles the learning problem itself. The recent advances in compression theory [4] could therefore impact both the theory of generalization and practical algorithms in machine learning.

Chapter 6

Conclusion

Our limitations to let machines learn simple functions are yet manifold. First of all, recursion is a powerful, yet simple computational concept that constitutes a multitude of simple functions. However, feed-forward neural networks and other models with non-recursive structure can not even express some simple recursive functions, and hence disregard a broad range of relevant patterns in nature. At the same time, learnability conditions for standard optimization objectives like ERM or IRM rely on some constraints what the representation class can express. Incorporating the discriminative power of Kolmogorov complexity as a simplicity bias into the optimization objective would however guarantee that at least all total computable functions are learnable with finite statistical information, even in the overarching setting where all partial computable functions are admitted to the representation class, a scenario for which classical PAC learnability based on the VC dimension or Rademacher complexity bounds could not yield any guarantee. Moreover, this additional discriminative power can also reduce the number of samples that are required to information-theoretically guarantee learnability below the typical thresholds in the PAC model, such as was shown for d-dimensional parity functions, where less than d samples can suffice to render all hypotheses with a lower Kolmogorov complexity suboptimal.

Our distribution-free setting where learnability is merely conditioned on the functional information in the dataset substantiates that limited training distributions that do not cover the entire support of possible instances can certainly suffice to learn any computable function, as long as the samples from this limited distribution are sufficiently informative that they cannot be losslessly compressed below the shortest description length of the true function.

Learning is therefore inseparably linked to compression, because learning the true function can be regarded as identifying the shortest compression that encompasses sufficient information to generate the labels when given the samples. However, compression algorithms do not keep up with the compression power of Turing Machines, and therefore can not even yield approximate guarantees on the *order* between the Kolmogorov complexity of two strings.

Explicitly realising such an approximation of at least the ordering that Kolmogorov complexity induces seems hard and directly related to the learning problem itself.

6.1 Future work

In practice, viable heuristics that indirectly pursue the simplicity bias of Kolmogorov complexity might however already take advantage from its aforementioned benefits. The statements in Chapter 4 hold for any possible prefix-free encoding of Turing Machines, since they only rest on the property that there are only finitely many functions with a lower complexity. Although the heuristic simplicity biases we derive in practice might not precisely agree with the intuitive information-theoretic notion of simplicity that Kolmogorov complexity over the standard Turing Machine encodings expresses, such heuristics might advance our theoretical toolbox to derive out-of-distribution generalization guarantees under weaker statistical assumptions. Instead of encoding Turing Machines, such simplicity biases could also be directly defined over model classes such as neural networks. While the complexity of multi-layer neural networks is usually merely quantified in terms of their layer width, depth, or the norm of its weights, desirable simplicity biases could further consider regularities in the computation graph of the neural network. As this thesis put forward the general benefit of simplicity biases independent of the exact order between functions, having such a specific simplicity bias at hand would allow to quantify more precise information-theoretic learnability guarantees.

Moreover, the role of recursion in empowering model classes to express simple but omnipresent patterns should not be neglected. However, the problems that bar the way from learning recursive algorithms still remain uncharted waters. Firstly, the error backpropagation on which gradient descent draws upon can not be directly transferred to models where the structure of the computation graph is not fixed but also subject to optimization. On the contrary, recurrent neural networks have a *fixed* recursive structure. When they are trained to fit a time series, they are provided with a learning signal at each step in time to optimize parameters within this fixed structure. But it is the recursive structure itself that needs to be learned when models shall express a more powerful range of simple recursive patterns. However, it is hard to parametrize this model structure in a way that a function that expresses this model class is differentiable over these parameters. Therefore, it is yet unclear how alterations in the structure shall be guided by the value of the loss function.

Although these challenges are hard, they seem impossible to bypass on our quest for empowering machines to learn simple functions and establishing guarantees about their generalization behaviour.

Appendix A

Secondary Theory and Proofs

A.1 Elementary calculus

Lemma A.1 (Log-Linear Inequality)
Let $a \geq 1$ be an arbitrary real number. For any real number $x > 2^{4a}$, $a \log_2(x) < x$.

Proof. Fix an arbitrary real number $a \geq 1$.

First of all, we show that $x < 2^x$ for all $x \in \mathbb{R}$ with $x \geq 1$ by elementary calculus. Define the real functions $g(x) = x, h(x) = 2^x$. Both g and h are continuous differentiable with derivatives $g'(x) = 1, h'(x) = \ln(2)2^x$. For any $x \geq 1$, the strict monotonicity of the exponential function yields

$$h'(x) = \ln(2)2^x \geq 2\ln(2) = \ln(4) \geq \ln(e) = 1 = g'(x). \tag{A.1}$$

But we also have $h(1) = 2 > 1 = g(1)$. Therefore, for any $x \geq 1$, it holds that

$$2^x = h(x) = h(1) + \int_1^x h'(x')dx' \overset{(A.1)}{>} g(1) + \int_1^x g'(x')dx' = g(1) + g(x) - g(1) = g(x). \tag{A.2}$$

In particular, the above result implies $\log_2(x) < x$ for all $x \geq 1$ by substituting $x = 2^z$ into Equation A.2. To generalize this argument to $a\log_2(n)$, we conduct an analogous argument for the real functions $g_a(x) = a\log_2(x)$ and $h(x) = x$.

By $\log_2(x) = \log_2(e)\ln(x)$ for all $x > 0$, we obtain the derivative of g_a as $g_a'(x) = \log_2(e)a\frac{1}{x}$. Since $\log_2(e) < 2$, any $x \geq 2a$ satisfies

$$g_a'(x) = \log_2(e)a\frac{1}{x} < 2a\frac{1}{2a} = 1 = h'(x). \tag{A.3}$$

Now, take $x = 2^{4a}$. Since $2^{2a} \geq 2^2 = 4$ by the assumption $a \geq 1$, it holds that

$$g_a(x) = a\log_2(x) = a\log_2(2^{4a}) = 4 \cdot a \cdot a \overset{(A.2)}{<} 2^{2a} \cdot 2^a \cdot 2^a = 2^{4a} = x = h(x). \tag{A.4}$$

As $2^{4a} \geq 2^{2a} \geq 2a$, for all $x > 2^{4a}$ this eventually yields

$$x = h(x) = h(2^{4a}) + \int_{2^{4a}}^{x} h'(x')dx' \tag{A.5}$$

$$\overset{(A.3)}{>} g_a(2^{4a}) + \int_{2^{4a}}^{x} g_a'(x')dx' \tag{A.6}$$

$$= g_a(2^{4a}) + g_a(x) - g_a(2^{4a}) = g_a(x) = a\log_2(x). \tag{A.7}$$

□

Corollary A.1.1 (Log-Linear Inequality with Additive Constant)
Let $a \geq 1, b \geq 0$ be arbitrary real numbers. For any real number $x > 2^{4(a+b)}$, $a\log_2(x)+b < x$.

Proof. Because $a \geq 1$, it is guaranteed that $x \geq 2^4 \geq 2$, and hence $\log_2(x) \geq 1$. Since $a + b \geq 1$, we employ Lemma A.1 to conclude $a\log_2(x) + b < (a + b)\log_2(x) \leq x$. □

Lemma A.2 (General Cascade Exponential Inequality)
Let $\exp_2^{(m)}$ denote the m-wise recursive concatenation of $\exp_2(x) := 2^x$ as in Definition 3.4. Let a be an arbitrary real scalar with $a \geq 1$, and $k \in \mathbb{N}$. For any natural number $n \in \mathbb{N}$ with $n > 4 \cdot 2^k \cdot a$ and any real number $x \geq 2$, it holds that

$$\exp_2^{(n-k)}(x) > a \cdot n \cdot x. \tag{A.8}$$

Proof. Denote by g_2 the real function $g_2(x) = 2x$. A similarly elementary argument as in the proof of Lemma A.1 yields $\exp_2(x) = 2^x \geq 2x = g_2(x)$ for any $x \geq 2$. Repeated application of this inequality thence yields

$$\exp_2^{(n)}(x) \geq g_2^{(n)}(x) = 2^n \cdot x \tag{A.9}$$

for any $n \in \mathbb{N}$. In particular, this states that $\exp_2^{(n-k)}(x) \geq 2^{n-k} \cdot x$ for any $n \geq k$.

Now, assume that $n > 4 \cdot 2^k \cdot a$. By Lemma A.1, it holds that $2^n > 2^k \cdot a \cdot n$. For that reason, we obtain

$$\exp_2^{(n-k)}(x) \overset{(A.9)}{\geq} 2^{n-k} \cdot x \overset{A.1}{>} 2^{-k} \cdot 2^k \cdot a \cdot n \cdot x = a \cdot n \cdot x. \tag{A.10}$$

□

Corollary A.2.1 (Special Cascade Exponential Equality)
Let $a \geq 1, b \geq 0$ be arbitrary real numbers. For any $k \in \mathbb{N}$ and $n \in \mathbb{N}$ with $n > 4 \cdot 2^k \cdot (a+b)$, it holds that

$$\exp_2^{(n-k)}(1) > a \cdot n + b. \tag{A.11}$$

Proof. Let $a \geq 1, b \geq 0$ be arbitrary real numbers. Let $n > 4 \cdot 2^k \cdot (a + b)$ be arbitrary. First of all, Equation A.2 asserts that $2^k > k$. With $a + b \geq 1$, this implies $n > 4 \cdot k \geq k + 1$ for $k \geq 1$. For $k = 0$, we similarly have $n > 4 \cdot 2^k = 4 > k + 1$. Therefore, we may reduce $\exp_2^{(n-k)}(1) = \exp_2^{(n-k-1)}(2)$ and conclude in the same fashion as Lemma A.2:

$$\exp_2^{(n-k-1)}(2) \overset{(A.9)}{\geq} 2^{-k-1} \cdot 2^n \cdot 2 \tag{A.12}$$

$$\overset{(A.10)}{>} 2^{-k-1} \cdot 2^k \cdot (a + b) \cdot n \cdot 2 = (a + b) \cdot n \overset{n \geq 1}{\geq} an + b. \tag{A.13}$$

□

A.2 Inexpressivity of scaled recursive completion

Corollary A.2.2 (Scaled Recursive Completion is not non-recursively expressible)
Let τ be as in Theorem 3.8.

For any $a \in \mathbb{N}$, define the scaled recursive completion over τ as $(f_\tau)_a^\Diamond(n) := f_\tau^{(n)}(a \cdot n)$.

Then, $(f_\tau)_a^\Diamond \notin \mathcal{F}_\tau$ for any $a \geq 1$. Let $f \in \mathcal{F}_\tau$ be an arbitrary non-recursive function over τ. Then, for the same $n_0(f)$ as in Theorem 3.8, we have that for all $a \in \mathbb{N}$ with $a \geq 1$ and $n \geq n_0$, $f(n) \neq (f_\tau)_a^\Diamond(n)$ for all $n \geq n_0(f)$.

Proof. We show that the thresholds n_0 in the proof of Theorem 3.8 maintain to hold for $(f_\tau)_s^\Diamond$ for any scale $s \in \mathbb{N}_{\geq 1}$. In the following, let $s \in \mathbb{N}$ be arbitrary with $s \geq 1$.

Analogously to Theorem 3.8, we prove a stronger argument by induction over the depth of non-recursive functions. That is, we show that for all non-recursive functions $f \in \mathcal{F}_\tau$, there is an $n_0 \in \mathbb{N}$ such that $f(n+a) < f_\tau^{(n)}(s \cdot (n+a))$ for all $n \geq n_0$ and all offsets $a \in \mathbb{N}$. Note that this statement was already proved for $s = 1$ in the original theorem.

Since the assumptions on τ are the same as in Theorem 3.8, we can directly avail to its equations. For depth 0 and any $n \geq n_0 = 1, a \in \mathbb{N}$, we extend Equations 3.10 and 3.11 to

$$f_\tau^{(n)}(s \cdot (n+a)) \overset{(3.11)}{\geq} f_m^{(n)}(s \cdot (n+a)) \geq f_m^{(n)}(n+a) \overset{(3.10)}{>} n + a = f(n+a). \tag{A.14}$$

Proceeding with the induction hypothesis (IH), assume that there is some $p \in \mathbb{N}$ such that for every non-recursive function $f \in \mathcal{F}_\tau$ with $\mathrm{dep}(f) \leq p$, there exists an $n_0 \in \mathbb{N}$ such that $f_\tau^{(n)}(s \cdot (n+a)) > f(n+a)$ for all $n \geq n_0$ and $a \in \mathbb{N}$. As in Theorem 3.8, we expand any $f \in \mathcal{F}_\tau$ with depth $p+1 \geq 1$ as $f(n) = f_m(g_1(n), \ldots, g_k(n))$ for some $f_m \in \tau$ with arity $k \in \mathbb{N}$ and non-recursive functions $g_i \in \mathcal{F}_\tau$ with depth $\mathrm{dep}(g_i) \leq p$ and distinguish by the cases where f_m is bounded or strictly monotonously increasing.

In the case that f_m is bounded by some c_m, the same $n_0 = c_m$ as in Equation A.35 in the proof of the original theorem satisfies

$$f_\tau^{(n)}(s \cdot (n+a)) \overset{(A.14)}{>} n + a \geq c_m \geq f(n+a). \tag{A.15}$$

In the other case where f_m is strictly monotonously increasing, we adopt $n_0 = \max_{1 \leq i \leq k} n_0(g_i)$ as it stands from the proof of Theorem 3.8. This n_0 is hence independent of s. Similarly, let $n \geq n_0$ and $a \geq 1$ be arbitrary and substitute $n' = n+1$ and $b = a-1$. Then we analogously have

$$\begin{aligned}
f(n'+b) = f(n+a) &= f_m(g_1(n+a), \ldots, g_k(n+a)) && \text{(A.16)}\\
&\overset{(IH)}{<} f_m(f_\tau^{(n)}(s \cdot (n+a)), \ldots, f_\tau^{(n)}(s \cdot (n+a))) && \text{(A.17)}\\
&= f_m^-(f_\tau^{(n)}(s \cdot (n+a))) && \text{(A.18)}\\
&\overset{3.6}{\leq} f_\tau^-(f_\tau^{(n)}(s \cdot (n+a))) && \text{(A.19)}\\
&= f_\tau^{(n+1)}(s \cdot (n+a)) = f_\tau^{(n')}(s \cdot (n'+b)). && \text{(A.20)}
\end{aligned}$$

The overall statement consequently follows by the induction principle, and as a special case, we obtain that for any non-recursive function $f \in \mathcal{F}_\tau$, there exists an $n_0(f) \in \mathbb{N}$ such that $f(n) < f_\tau^{(n)}(s \cdot n) = (f_\tau)_s^\Diamond(n)$ for all $n \geq n_0$.

□

A.3 Extension of non-recursive inexpressivity results to integer functions

To directly apply our results to neural networks with finite precision floating numbers, we extend the inexpressivity of non-recursive functions from Section 3.3 to functions sets over $\mathbb{Z}$.

The necessary alterations are only small and the assumptions on the properties of the functions in the function set τ are still realistic, such that they are directly applicable to multi-layer neural networks and other non-recursive models on common computing architectures.

In particular, the inequality in Equation (3.7) in the proof of the lower bound of the sum $f_\tau := \sum_{i=1}^{j} f_i$ in Lemma 3.6 relied on the non-negativity of each f_i. To generalize this statement to functions over $\mathbb{Z}$, we simply take the *magnitude* of each function f_i before summing them up. Therefore, we redefine the magnitude sum function $f_\tau(n) := \sum_{i=1}^{j} |f_i(n)|$. Juxtaposing this construction f_τ with non-recursive functions over τ is still perfectly fair, since both the addition function and the magnitude function fall within the scope of functions that still satisfy our assumptions on τ. The uniform Turing Machine $\mathcal{T}_\Diamond$ from 3.9 only needs a slight adjustment to correctly compute the recursive completion over τ $f_\tau^\Diamond$ according to this new definition. Representing integers $n \in \mathbb{Z}$ in the two's complement as is usually done in floating point arithmetic on computing architectures, $\mathcal{T}_\Diamond$ only has to compute the magnitude of the output after each simulation on the computation tape before it accumulates this output on the accumulation tape.

Secondly, we adapt our assumptions on the functions in τ. Originally, we assumed that each function $f_i : \mathbb{N} \to \mathbb{N}$ in τ is either bounded or strictly monotonously increasing. Moreover, we required that at least one strictly monotonously increasing function f_i has an arity $\mathrm{ar}(f_i) > 1$. The definition of bounded function directly transfers to functions $f_i : \mathbb{Z} \to \mathbb{Z}$. On the other hand, sign flips in functions over $\mathbb{Z}$ sometimes invert the monotonic behaviour of a function, as is the case for the multiplication function. To reconcile the constraint of strict monotonicity with this behaviour, it is viable to merely require strict monotonicity over natural numbers and bound the magnitude of the function values for non-negative and possibly negative inputs by the function value for the respective non-negative inputs. We formalise the latter condition in the following vein

Definition A.3 (Dominance on $\mathbb{N}$)
Let $f : \mathbb{Z}^k \to \mathbb{Z}$ be an arbitrary function. We say that f is dominant on $\mathbb{N}$ if for all $x_1, \ldots, x_k \in \mathbb{Z}$,

$$|f(x_1, \ldots, x_k)| \leq f(|x_1|, \ldots, |x_k|). \tag{A.21}$$

For at least one strictly monotonously increasing function $f_i \in \tau$, we however still require arity $\mathrm{ar}(f_i) > 1$. Additionally, the strict monotonicity of f_i must hold entirely over $\mathbb{Z}$,

not only over $\mathbb{N}$ as for the other strictly monotonously increasing functions. The addition function f_+ however still satisfies these conditions. For such functions, we slightly adjust the self-lower bound from Lemma 3.5 to take into account the possibly negative function value at 0. For completeness, we still provide the full proof.

Lemma A.4 (Lower bound for recursive concatenation of strictly monotonously increasing functions)
Let $f : \mathbb{Z}^k \to \mathbb{Z}$ be an arbitrary k-ary function that is strictly monotonously increasing with $k \geq 1$. Let $f^{(m)}$ denote the m-wise recursive concatenation of f as in Definition 3.4.

It holds that $f^{(m)}(n) \geq k^m \cdot n + f^-(0) \cdot \sum_{i=0}^{m-1} k^i$ for all $n, m \in \mathbb{N}$.

Proof. Fix an arbitrary, strictly monotonously increasing function f of arity k. First of all, we show that $f^-(n) \geq k \cdot n + f^-(0)$ for all $n \in \mathbb{N}$.

Exhaustive exploitation of strict monotonicity yields

$$f^-(n) = f(\underbrace{n, \ldots, n}_{k \text{ times}}) \geq f(n-1, n, \ldots, n) + 1 \tag{A.22}$$

$$\overset{\cdots}{\geq} f(0, n, \ldots, n) + n \tag{A.23}$$

$$\overset{\cdots}{\geq} f(0, 0, \ldots, 0) + k \cdot n = f^-(0) + k \cdot n. \tag{A.24}$$

Now, we prove the general lower bound for $f^{(m)}(n)$ by induction over the number of recursive concatenations $m \in \mathbb{N}$.

The base case $m = 0$ holds by definition since $f^{(0)}(n) = n = k^0 \cdot n + f^-(0) \cdot \underbrace{\sum_{i=0}^{-1} k^i}_{=0}$.

For the induction hypothesis (IH), assume that there is some $m \in \mathbb{N}$ such that $f^{(m)}(n) \geq k^m \cdot n + f^-(0) \cdot \sum_{i=0}^{m-1} k^i$ holds for all $n \in \mathbb{N}$. By the strict monotonicity of f and Equation 3.5, we obtain

$$f^{(m+1)}(n) = f^-(f^{(m)}(n)) \overset{\text{(IH)},(3.5)}{\geq} k \cdot f^-(0) \cdot \sum_{i=0}^{m-1} k^i + f^-(0) = k^{m+1} \cdot n + \sum_{i=0}^{(m+1)-1} k^i. \tag{A.25}$$

Consequently, the overall result follows by the induction principle. □

With the new definition of f_τ that considers the *magnitude* of each function f_i, lower bound of the recursive concatenation $f_\tau^{(m)}$ may accordingly include the magnitude too.

Lemma A.5 (Lower bound for recursive concatenation of function set sum)
Given an arbitrary function set $\tau = \{f_1, \ldots, f_j\}, f_i : \mathbb{Z} \to \mathbb{Z}$. Denote by f_τ the magnitude sum $f_\tau(n) = \sum_{i=1}^{j} \left|f_i(n)\right|$.

For any monotonously increasing function $f_k \in \tau$ and any $m, n \in \mathbb{N}$, it holds that $f_\tau^{(m)}(n) \geq \left|f_k^{(m)}(n)\right|$.

Proof. Given all as above. We show the statement by induction over $m \in \mathbb{N}$.

For $m = 0$, the result follows from the definition

$$f_\tau^{(0)}(n) = \sum_{i=1}^{j} \left| f_i^{(0)}(n) \right| = \sum_{i=1}^{j} n \geq n = f_k^{(0)}(n).$$

for all $1 \leq k \leq j$ and $n \in \mathbb{N}$.

Now, assume that for some $m \in \mathbb{N}$, the statement $f_\tau^{(m)}(n) \geq |f_i^{(m)}(n)|$ holds for all monotonously increasing functions $f_i \in \tau$ and all $n \in \mathbb{N}$.

For $m+1$, any monotonously increasing $f_k \in \tau$ satisfies

$$f_\tau^{(m+1)}(n) = f_\tau^-(f_\tau^{(m)}(n)) = \sum_{i=1}^{j} \left| f_i^-(f_\tau^{(m)}(n)) \right| \geq \left| f_k^-(f_\tau^{(m)}(n)) \right| \tag{A.26}$$

$$\overset{\text{(IH)}}{\geq} \left| f_k^-(f_k^{(m)}(n)) \right| = \left| f_k^{(m+1)}(n) \right|, \tag{A.27}$$

where the first inequality draws upon the non-negativity of each summand $\left| f_i^-(f_\tau^{(m)}(n)) \right|$. Therefore, the overall statement for all $m \in \mathbb{N}$ follows by induction.

□

With these adjusted bounds at hand, we can obtain the same inexpressivity result as in the original Theorem 3.8. Our requirement that each function in τ is either bounded, or dominant and strictly monotonously increasing on $\mathbb{N}$ is still satisfied by arithmetic operations such as addition and multiplication, as well as logical expressions and constants. Moreover, any activation functions that are usually employed in practice can be expressed as finite concatenations of functions that satisfy the above conditions.

It therefore covers the range of functions that are used to construct multi-layer neural networks like MLPs in practice.

Theorem A.6 (Recursive sum completion is not non-recursively expressible)
Fix an arbitrary function set $\tau = \{f_1, \ldots, f_j\}$, $f_i : \mathbb{Z}^k \to \mathbb{Z}$, where each f_i is either

- *bounded, or*
- *dominant and strictly monotonously increasing on $\mathbb{N}$.*

Assume that there is at least one $f_i \in \tau$ that is strictly monotonously increasing on $\mathbb{Z}$ and has arity $\mathrm{ar}(f_i) > 1$, e.g. the addition function. For any non-recursive function $f \in \mathcal{F}_\tau$, there is an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) \neq f_\tau^\Diamond(n)$.

Proof. Given an arbitrary function set $\tau = \{f_1, \ldots, f_j\}$ as above. Define $f_\tau(n) := \sum_{i=1}^{j} |f_i(n)|$ as in Lemma A.5.

As it facilitates the induction step, we prove a stronger statement by induction over the depth of non-recursive functions $\mathrm{dep}(f) \in \mathbb{N}$. Namely, we are going to prove that for all non-recursive functions $f \in \mathcal{F}_\tau$, there is an $n_0 \in \mathbb{N}$ such that $|f(n+a)| < f_\tau^{(n)}(n+a)$ for all $n \geq n_0$ and all offsets $a \in \mathbb{N}$.

Beginning with the base case, let $f \in \mathcal{F}_\tau$ be arbitrary with $\mathrm{dep}(f) = 0$. Therefore, f must be equivalent to the identify function, thus $f(n) = n$ for all $n \in \mathbb{Z}$.

Let $a \in \mathbb{N}$ be arbitrary.

Since $k = \mathrm{ar}(f_m) > 1$ for at least one strictly monotonously increasing $f_\ell \in \tau$, Lemma A.4 guarantees that

$$f_\ell^{(n)}(n+a) \geq k^n \cdot (n+a) + f_\ell^-(0) \cdot \sum_{i=0}^{n-1} k^i. \tag{A.28}$$

for any $n \in \mathbb{N}$. Because $k > 1$, it further holds by the geometric sum that

$$f_\ell^-(0) \cdot \sum_{i=0}^{n-1} k^i \geq -\big|f_\ell^-(0)\big| \cdot \frac{k^n - 1}{k-1} \geq -\big|f_\ell^-(0)\big| \cdot (k^n - 1). \tag{A.29}$$

Choose $n_0 := |f_\ell^-(0)| + 1$ and let $n \geq n_0$ be arbitrary. Joining the above inequalities, we obtain

$$\begin{aligned}
f_\ell^{(n)}(n+a) &\overset{(A.28)}{\geq} k^n \cdot (n+a) + f_\ell^-(0) \cdot \sum_{i=0}^{n-1} k^i && \text{(A.30)}\\
&\overset{(A.29)}{\geq} k^n \cdot (n+a) - (k^n - 1) \cdot \big|f_\ell^-(0)\big| && \text{(A.31)}\\
&= (k^n - 1) \cdot \underbrace{\big(n + a - \big|f_\ell^-(0)\big|\big)}_{>0} + n + a && \text{(A.32)}\\
&> n + a = |n + a|. && \text{(A.33)}
\end{aligned}$$

Since f_ℓ is monotonously increasing, we conclude by means of Lemma A.5 that

$$f_\tau^{(n)}(n+a) \overset{A.5}{\geq} f_\ell^{(n)}(n+a) \overset{(A.30)}{>} |n+a| = |f(n+a)|. \tag{A.34}$$

Proceeding with the induction hypothesis (IH), assume that there is some $p \in \mathbb{N}$ such that for every non-recursive function $f \in \mathcal{F}_\tau$ with $\mathrm{dep}(f) \leq p$, there exists an $n_0 \in \mathbb{N}$ such that $f_\tau^{(n)}(n+a) > f(n+a)$ for all $n \geq n_0$ and $a \in \mathbb{N}$.

Let $a \in \mathbb{N}$ be arbitrary in the following. Consider an arbitrary non-recursive function $f \in \mathcal{F}_\tau$ with $\mathrm{dep}(f) = p + 1$.

As by Definition 3.1, we have $f(n) = f_m(g_1(n), \ldots, g_k(n))$ for some $f_m \in \tau$ with arity $k \in \mathbb{N}$ and non-recursive functions $g_i \in \mathcal{F}_\tau$ with depth $\mathrm{dep}(g_i) \leq p$.

To begin with, we consider the case where f_m is a bounded function. Denote by c_m the bound of f_m, that is $|f_m(n)| \leq c_m$ for all $n \in \mathbb{Z}$. For any $n \geq \max\big(|f_\ell^-(0)| + 1, c_m\big)$, where f_ℓ is the strictly monotonously increasing function from before, Equation 3.11 assures that

$$f_\tau^{(n)}(n+a) \overset{(3.11)}{>} n + a \geq c_m \geq \big|f_m\big(g_1(n+a), \ldots, g_k(n+a)\big)\big| = |f(n+a)|. \tag{A.35}$$

Otherwise, consider the case where f_m is dominant and strictly monotonously increasing on $\mathbb{N}$. By the induction hypothesis, for every non-recursive function g_i, there is an $n_0(g_i)$ such that $f_\tau^{(n)}(n+a) > |g_i(n+a)|$ for all $n \geq n_0(g_i)$. We hence directly take the threshold as the maximum $n_0 = \max_{1 \leq i \leq k} n_0(g_i)$.

Now, let $n \geq n_0$ be arbitrary. Subsequently, we assume that $a \geq 1$, and substitute $b = a - 1$ and $n' = n + 1$. By the strict monotonicity of f_m on $\mathbb{N}$, we conclude

$$|f(n'+b)| = |f(n+a)| = \big|f_m\big(g_1(n+a), \ldots, g_k(n+a)\big)\big| \tag{A.36}$$

$$\overset{A.3}{\leq} f_m\big(|g_1(n+a)|, \ldots, |g_k(n+a)|\big) \tag{A.37}$$

$$\overset{(IH)}{<} f_m(f_\tau^{(n)}(n+a), \ldots, f_\tau^{(n)}(n+a)) \tag{A.38}$$

$$\leq \big|f_m^-(f_\tau^{(n)}(n+a))\big| \tag{A.39}$$

$$\leq f_\tau^-(f_\tau^{(n)}(n+a)) = f_\tau^{(n+1)}(n+a) = f_\tau^{(n')}(n'+b). \tag{A.40}$$

Since $a \in \mathbb{N}_{\geq 1}$ was chosen arbitrarily, the above argument holds for all $b \in \mathbb{N}$ and all $n' \geq n_0 + 1 =: n_0(f)$. To conclude with, we have shown that for any non-recursive function $f \in \mathcal{F}_\tau$ with $\mathrm{dep}(f) = p+1$, there exists an $n_0(f) \in \mathbb{N}$ such that $|f(n+a)| < f_\tau^{(n)}(n+a)$ for all $n \geq n_0(f), a \in \mathbb{N}$.

Consequently, the overall result for non-recursive functions of arbitrary depth draws upon the induction principle.

As a special case of the above result, we obtain that for any non-recursive function $f \in \mathcal{F}_\tau$, there is an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) < f_\tau^{\Diamond}(n)$.

□

References

[1] K. Ahuja, E. Caballero, D. Zhang, J.-C. Gagnon-Audet, Y. Bengio, I. Mitliagkas, and I. Rish, "Invariance principle meets information bottleneck for out-of-distribution generalization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3438–3450, 2021.

[2] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, "Invariant risk minimization," *arXiv preprint arXiv:1907.02893*, 2019.

[3] D. Paccagnan, M. Campi, and S. Garatti, "The pick-to-learn algorithm: Empowering compression for tight generalization bounds and improved post-training performance," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[4] M. C. Campi and S. Garatti, "Compression, generalization and learning," *Journal of Machine Learning Research*, vol. 24, no. 339, pp. 1–74, 2023.

[5] R. Cilibrasi and P. M. Vitányi, "Clustering by compression," *IEEE Transactions on Information theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[6] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[7] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and S. Y. Philip, "Generalizing to unseen domains: A survey on domain generalization," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 8, pp. 8052–8072, 2022.

[8] D. Krueger, E. Caballero, J.-H. Jacobsen, A. Zhang, J. Binas, D. Zhang, R. Le Priol, and A. Courville, "Out-of-distribution generalization via risk extrapolation (rex)," in *International Conference on Machine Learning*, PMLR, 2021, pp. 5815–5826.

[9] V. Vapnik, "Principles of risk minimization for learning theory," *Advances in neural information processing systems*, vol. 4, 1991.

[10] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, "Toward causal representation learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 612–634, 2021.

[11] P. Kamath, A. Tangella, D. Sutherland, and N. Srebro, "Does invariant risk minimization capture invariance?" In *International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 4069–4077.

[12] E. Rosenfeld, P. Ravikumar, and A. Risteski, "The risks of invariant risk minimization," *arXiv preprint arXiv:2010.05761*, 2020.

[13] R. J. Solomonoff, "A formal theory of inductive inference. Part I," *Information and control*, vol. 7, no. 1, pp. 1–22, 1964.

[14] F. Rosenblatt *et al.*, *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan books Washington, DC, 1962, vol. 55.

[15] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[16] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. URL: https://www.sciencedirect.com/science/article/pii/S0893608005801315.

[17] P. J. Haley and D. Soloway, "Extrapolation limitations of multilayer feedforward neural networks," in *[Proceedings 1992] IJCNN international joint conference on neural networks*, IEEE, vol. 4, 1992, pp. 25–30.

[18] K. Xu, J. Li, M. Zhang, S. S. Du, K.-i. Kawarabayashi, and S. Jegelka, "What can neural networks reason about?" *arXiv preprint arXiv:1905.13211*, 2019.

[19] L. A. Levin, "Universal sequential search problems," *Problemy peredachi informatsii*, vol. 9, no. 3, pp. 115–116, 1973.

[20] A. M. Turing *et al.*, "On computable numbers, with an application to the entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[21] A. M. Turing, "Computability and λ-definability," *The Journal of Symbolic Logic*, vol. 2, no. 4, pp. 153–163, 1937.

[22] A. M. Turing, "Intelligent machinery (1948)," *B. Jack Copeland*, p. 395, 2004.

[23] M. Li, P. Vitányi, *et al.*, *An introduction to Kolmogorov complexity and its applications*. Springer, 2008, vol. 3.

[24] K. Gödel, "Über formal unentscheidbare sätze der principia mathematica und verwandter systeme I," *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173–198, 1931.

[25] G. J. Chaitin, "Information-theoretic limitations of formal systems," *Journal of the ACM (JACM)*, vol. 21, no. 3, pp. 403–424, 1974.

[26] R. O'Donnell, *Analysis of boolean functions*. Cambridge University Press, 2014.

[27] P. M. Vitányi, "How incomputable is kolmogorov complexity?" *Entropy*, vol. 22, no. 4, p. 408, 2020.

[28] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 06, pp. 8–19, 1984.

[29] P. Deutsch, *Rfc1951: Deflate compressed data format specification version 1.3*, 1996.

[30] J. Alakuijala, A. Farruggia, P. Ferragina, E. Kliuchnikov, R. Obryk, Z. Szabadka, and L. Vandevenne, "Brotli: A general-purpose data compressor," *ACM Transactions on Information Systems (TOIS)*, vol. 37, no. 1, pp. 1–30, 2018.

[31] J. Alakuijala, E. Kliuchnikov, Z. Szabadka, and L. Vandevenne, "Comparison of brotli, deflate, zopfli, lzma, lzham and bzip2 compression algorithms," *Google Inc*, pp. 1–6, 2015.

[32] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.