PAC-learning gains of Turing machines over circuits and neural networks

Brieuc Pinon*, Raphaël Jungers, Jean-Charles Delvenne

ICTEAM/INMA
UCLouvain
Louvain-la-Neuve, Belgium

Abstract

A caveat to many applications of the current Deep Learning approach is the need for large-scale data. One improvement suggested by Kolmogorov Complexity results is to apply the minimum description length principle with computationally universal models. We study the potential gains in sample efficiency that this approach can bring in principle. We use polynomial-time Turing machines to represent computationally universal models and Boolean circuits to represent Artificial Neural Networks (ANNs) acting on finite-precision digits.

Our analysis unravels direct links between our question and Computational Complexity results. We provide lower and upper bounds on the potential gains in sample efficiency between the MDL applied with Turing machines instead of ANNs. Our bounds depend on the bit-size of the input of the Boolean function to be learned. Furthermore, we highlight close relationships between classical open problems in Circuit Complexity and the tightness of these bounds.

Keywords: Kolmogorov Complexity, minimum description length, PAC-learning, Computational Complexity, Deep Learning, Program Induction

*Corresponding author, Avenue Georges Lemaître 4-6/L4.05.01, 1348 Louvain-la-Neuve, Belgium

Email addresses: brieuc.pinon@uclouvain.be (Brieuc Pinon), raphael.jungers@uclouvain.be (Raphaël Jungers), jean-charles.delvenne@uclouvain.be (Jean-Charles Delvenne)

1. Introduction

Recent years have seen a renew of interest in the development of methods to make inductive inferences in computationally universal programming languages. This work assesses the gain in sample efficiency that can be attained in principle from these methods in comparison to ANNs under simplifying algorithmic assumptions.

This theoretical investigation is similar to comparisons that have been made between ANNs and other classical Machine Learning algorithms to explain the experimental success of the former.

We present those two branches of Machine Learning research. The branch that develops methods to make inductive inferences in computationally universal programming languages; and the branch that compares encodings for hypotheses based on the induced minimal description sizes to express functions.

Inductive inference in computationally universal programming languages. An important aspect of Deep Learning (DL) research is the development of new architectures. Some of these architectures can efficiently address particular problems, like Convolutional Neural Network (CNN) for Computer Vision, Recurrent Neural Networks (RNN) for natural language processing, and Graph Neural Networks (GNN) that can be adapted to a wide variety of applications, see [1].

A common critical point in the development of these architectures is the exploitation of prior knowledge about the task at hand. This exploitation is done by imposing on the model some factorized representation. The structure of the representation enforces the a priori known symmetries in the underlying function to learn. This leads to improvement in the sample efficiency, see [2] for a paper taking this perspective for GNN with associated PAC-learning results.

For CNN, RNN, and GNN, the factorization's structure is classically fixed a priori with respect to prior information about the symmetries in the function to learn. Alternatively, the structure could be learned from the data.

It is thus natural to invest efforts in the construction of learning algorithms with the flexibility to define and use potential abstract structures found in the data. In this line of work, we mention two approaches: *Differentiable programming* which consists in learning with DL architectures close to Turing-complete systems such as in [3, 4, 5, 6, 7, 8, 9, 10, 11]; and learning in non-differentiable programs written in universal languages for which combinatorial

optimization methods such as genetic programming must be used, see [12]. This latter approach is usually referred to as *Inductive Programming* or *Program synthesis* from examples, see [13] and [14].

Our work is a theoretical investigation of the potential sample efficiency gains that could be observed from Inductive Programming or learning with these new expressive DL architectures in comparison to classical Artificial Neural Networks.

Models: the necessary sizes to represent functions. This last decade DL algorithms allowed to tackle problems that had been impossible to solve until then. These successes opened the question: Why DL is more efficient on these complex problems than other classical Machine Learning algorithms, such as shallow neural networks?

A theoretical answer is that depth in Artificial Neural Networks (ANNs) allows us to efficiently encode some classes of functions. More precisely there exists a sequence of functions for which low-depth ANNs need an exponential number of neurons to approximate it and, in comparison, higher-depth ANNs only need a polynomial number of neurons to achieve the same approximation. Examples of references on the subject are [15, 16, 17], and, from the perspective of Boolean circuits, [18].

The expressive power of depth in ANNs to efficiently represent functions in terms of the sizes of the hypotheses has also been studied in comparison with Support Vector Machines in [19] and with Decision Trees in [20].

Our work is inspired by these comparisons between models. Similarly, we show a separation in terms of the sizes of the hypotheses that are necessary to fit functions. More precisely, we study the advantage that Turing machines have over circuits and neural networks. Pushing this observation to PAC-learning claims, we study potential gains on the number of samples that are necessary to learn Boolean functions by Turing machines.

Objectives and formal choices. Given the motivations for induction in computationally universal programming languages, we investigate the sample efficiency gains that we can hope from this approach relative to classical ANNs.

We use Turing machines to represent models based on Turing-complete systems, such as the new expressive DL architectures for example. This choice of the computational model is not determinant since Turing machines can serve as a proxy to study other Turing-complete systems.

We compare Turing machines with Boolean circuits and classical ANNs. By ANNs we mean the simplest form of DL with non-linear activation functions composed on top of linear transformations and without any repetition of the weights such as in CNN or RNN. To represent ANNs we choose a computationally feasible model, the model is not based on real numbers but is discrete and finite.

The *minimum description length* (MDL) principle consists in choosing the hypothesis with the shortest description length while being consistent with the data. It is a classical formalization of Occam's razor principle. The MDL principle will allow us to translate our choices of models —which are ways to express the hypotheses— into learning algorithms. It will give us a general and effective way to compare inductive biases posed by different representations such as ANNs and Turing machines for example.

To assess the performance of the models/learning algorithms, we follow the classical PAC-learning framework. However, we do not explore the computational efficiency question of finding the shortest hypothesis; rather we focus on the sample efficiency. In other words, throughout the paper, we neglect the computational resources needed to find the minimum description length hypothesis consistent with the data; but set our attention on the size of the dataset that is necessary to find a hypothesis that is Probably Approximately Correct.

Outline. In Section 2, we provide some background on PAC-learning, interpreters, and the MDL principle.

Then, from it, we introduce in Section 3 the critical metric at the heart of our objectives: the *sample efficiency gains* of a model over another.

In Section 4, we prove bounds on the sample efficiency gains that circuits have over (polynomial-time) Turing machines and conversely. In particular, we show that the sample efficiency gains of polynomial-time Turing machines over circuits are at least linear in the input-size of the function to learn. Whether they are superlinear or not is an open question. We connect this question to different open problems from Computational Complexity.

2. Background

2.1. Our learning problem

We want to learn an unknown function f belonging to a *hypothesis class*

$$H^n = \{f : \mathcal{B}^n \to \mathcal{B}\}, \tag{1}$$

where $\mathcal{B} = \{0, 1\}$, and thus H^n is finite.

For some fixed integer $n > 0$, a *learning problem* is determined by a boolean function $f \in H^n$ and a probability measure $\mathcal{P}$ on $\mathcal{B}^n$, $\mathcal{P} \in \Delta(\mathcal{B}^n)$.

Definition 1. *A* learning problem m-sample dataset *is a random variable defined as* $[x_j, f(x_j)]_{j=1}^m$ *where the* x_j *are sampled independently and according to the learning problem probability measure* $\mathcal{P}$*.*

To *solve* a learning problem is to find an approximation $\hat{f}$ of f from a realization of the learning problem m-sample dataset, for some natural m.

We formalize the notion of approximation with the classical accuracy.

Definition 2. *The* accuracy *of a function* $\hat{f} \in H^n$ *with respect to learning problem* $f \in H^n$, $\mathcal{P} \in \Delta(\mathcal{B}^n)$ *is*

$$acc_f^{\mathcal{P}}(\hat{f}) \stackrel{\text{def}}{=} \Pr_{x \sim \mathcal{P}} \left[\hat{f}(x) = f(x) \right]. \tag{2}$$

Definition 3. *A* learning algorithm *is a function that given the realization of a learning problem* m*-samples dataset, for some learning problem* $f \in H^n, \mathcal{P} \in \Delta(\mathcal{B}^n)$*, outputs a function* $\hat{f} \in H^n$*, for any* $n, m \in \mathbb{N}^+$*.*

We do not specify a representation for the output's function since, as said in the introduction, our work focuses on the sample efficiency of the learning algorithm and not its computational complexity.

The learning algorithm sample efficiency performance will be assessed by PAC-learning claims.

Definition 4. *For any* $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$*, an algorithm* $\mathcal{A}$ *has an* (ϵ, δ)*-PAC-learning performance with an* m*-sample dataset on learning problem* $(f, \mathcal{P})$ *if for all* $m' \geq m$

$$\Pr_{x \sim \mathcal{P}^{m'}} \left[acc_f^{\mathcal{P}} \left(\mathcal{A}([x_j, f(x_j)]_{j=1}^{m'}) \right) \geq 1 - \epsilon \right] \geq 1 - \delta. \tag{3}$$

2.2. Interpreter

We now introduce the concept of interpreters, which will allow us to define the notion of description-length of a hypothesis given a model.

Definition 5. *An* interpreter φ^T *is a Turing machine computing a two-arguments partial computable binary-valued function* $\varphi : \mathcal{B}^* \times \mathcal{B}^* \to \mathcal{B} \cup \{\perp\}$*. Where* $\mathcal{B}^*$ *is the set of finite length binary strings,* $\mathcal{B}^* = \cup_{i=0,1,\ldots} \mathcal{B}^i$*, and* $\perp$ *is the symbol representing non-halting executions.*

In the rest of the paper, we will identify the Turing machine implementation with its computed function by dropping the T in φ^T with exceptions where the distinction is useful.

In our developments, the first argument will correspond to the code/program/hypothesis and the second argument will correspond to the input of the function to learn.

We make Definition 5 concrete by presenting some interpreters, see Section Appendix C in the appendices for complete descriptions:

- **Universal Turing Machine $\mathcal{U}$:**

$$\mathcal{U}([\text{binary encoding of a two-inputs Turing machine } \mathcal{T}, \text{first input}], \text{second input}) = \mathcal{T}(\text{first input}, \text{second input}).$$

  Note that we will define Turing machines with, only, binary-valued outputs. The encoding of Turing machines and the Universal Turing machine are formally defined in the appendices, Definitions 30 and 31 respectively.

- **Polynomial-time Universal Turing Machine $\mathcal{U}^c$:** A Universal Turing machine with a limited computation time in $\mathcal{O}(n^c)$ steps, where n is the size of the input and $c \in \mathbb{N}^+$.

  This interpreter will allow us to make claims using hypotheses with reasonable running time.

- **Boolean circuit interpreter $\mathcal{C}$:** A Boolean circuit is a directed acyclic graph where each node is either an input node or a gate. Each input node is associated with an input value, and gates are associated with unary or binary logical operators (OR, AND, and NOT). There is one node with no child (sink), this node is the output node of the circuit.

  The topology of the graph is consistent with the nodes' logical association: input nodes' are not the child of any other node, gates with a binary operator have two parent nodes, and gates with a unary logical operator have a unique parent node.

  The output of a circuit on a binary input is the result of the output node after the application of the logical operations associated with the gates. In this computation, the input nodes naturally take their values from the input.

Boolean circuits are encoded as binary strings by first noting the input-size, then the number of nodes in the circuit (the circuit's size), and finally by describing the nodes one by one (associated input or logical operation, and the potential parents).

With this encoding, the description-length of a Boolean circuit of size S is in $O(S \log S)$.

Again the application of this interpreter is

$\mathcal{C}(\text{binary representation of a Boolean circuit}, \text{input}) = \text{output of the circuit}.$

- **ANNs interpreter:** We define an ANN as a directed acyclic graph whose nodes correspond to either an input or a floating-point operator. Similarly to Boolean circuits, each input node is associated with one variable of the binary input. The floating-point operators are taken from a predefined arbitrary set. This set can contain binary or unary operators such as $+, -, \times, /, \max(.,.), \exp(.)$; it also contains 0-ary/constant operators: the floating-point values themselves.

  Again similarly to Boolean circuits, there is a unique output node that has no children.

  The values of the nodes, given an input, are determined by the recursive application of the input nodes' association to input variables or of the corresponding floating-point operators until a value for the output node is obtained.

  There is a linear relationship between the description-length of a function with the Boolean circuits' or the ANNs' interpreter, see Proposition 42 in the appendices. We use this link to present all the results with the interpreter of Boolean circuits, $\mathcal{C}$, while exactly the same results will hold for ANN.

- **Support Vector Machines, Binary Decision Trees, CNN, RNN, GNN :** An interpreter can be defined for each of these classes. Note that in some cases the length of the binary representation of some functions can be drastically reduced or increased by using these specialized interpreters.

  The existence of these interpreters is noted to point out that the formal background presented applies to more than just Turing machines,

Boolean circuits, and ANNs. None of these interpreters will be discussed further in this work.

2.3. The Minimum description length (MDL) principle

We define how the application of the MDL principle with an interpreter gives a learning algorithm.

Notation. *Let $|h|$ be the length of a binary string h.*

We denote, for any interpeter φ, any n and any function $f \in H^n$:

$$|f|_\varphi \stackrel{\text{def}}{=} \min_{h \in \mathcal{B}^*} |h| \;\; s.t.\; \varphi(h, x) = f(x),\; \forall x \in \mathcal{B}^n.$$

By convention, if the problem is infeasible the value will be $+\infty$.

Definition 6. MDL principle with an interpreter φ: *MDL$^\varphi$. From an interpreter φ, one can create the following learning algorithm, MDL$^\varphi$.*

Input: the realization of a learning problem m-samples dataset.

Select the output function, $\hat{f}$, corresponding to a minimal-description-length program consistent with the dataset

$$\begin{aligned} h^* \in \arg\min_{h \in \mathcal{B}^*} \quad & |h| \\ \textit{subject to} \quad & \varphi(h, x_j) = f(x_j),\; j \in \{1, \dots m\}; \\ & \varphi(h, x) \neq \perp,\; \forall x \in \mathcal{B}^n. \end{aligned} \tag{4}$$

The output function is thus $\hat{f} = \varphi(h^*, .)$.

Thus, the choice of an interpreter completely determines the learning algorithm with the MDL principle. The inductive bias is fixed toward low complexity (short to express) hypotheses.

Let us remark that, any computable time-limit on the execution of an interpreter φ will ensure that *MDL$^\varphi$* is computable. In this paper, we will always assume that such an arbitrary sufficiently large time-limit is imposed, say $2^{2^{2^n}}$ for example.

The following PAC-learning bound will be the main proposition used through the paper to go from descriptions' length constraints to PAC-learning affirmations. It is based on a classical argument in the PAC-learning literature to show uniform convergence of the accuracy for finite hypothesis classes, see [21]. The argument is simply adapted to our specific MDL framework.

Proposition 7 (Description-length PAC-guarantee)**.** *There exists constants $a_1, a_2 > 0$ such that the following holds.*

For any interpreter φ and associated learning algorithm MDL$^{\varphi}$, any $(f, \mathcal{P})$ learning problem and any PAC-learning parameters $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, MDL$^{\varphi}$ has an (ϵ, δ)-PAC-learning performance with an m-sample dataset on the learning problem, where

$$m = \frac{a_1}{\epsilon}\left[\log\frac{1}{\delta} + |f|_{\varphi} + a_2\right]. \tag{5}$$

For $\varphi = \mathcal{U}$ or $\varphi = \mathcal{C}$ under mild conditions on the size of the circuits considered, the guarantee of Proposition 7 is tight up to a fixed factor for any configurations (ϵ, δ), any input-size n and any sufficiently large maximal description-length $|f|_{\varphi}$ on some learning problems ($f \in H^n, \mathcal{P} \in \Delta(\mathcal{B}^n)$). This is shown using a VC-dimension based argument in the appendices, we refer to Proposition 48 and its associated section.

3. Sample efficiency gains

We propose here a PAC-learning criterion to compare the sample efficiency of two arbitrary MDL-based learning algorithms. This criterion will then be applied to MDL with circuits and MDL with Turing machines.

We want to analyze the largest gap, in terms of necessary samples to get some learning performance, that can exist between two learning algorithms. We quantify the number of samples needed by a learning algorithm in order to solve a learning problem as the following.

Definition 8. *The minimal number of samples needed to get an (ϵ, δ)-PAC-learning performance for a learning algorithm MDL$^{\varphi}$ on a learning problem $(f, \mathcal{P})$ is*

$$m_{\varphi}^{\epsilon,\delta}(f, \mathcal{P}) \stackrel{\text{def}}{=} \min\big\{\{m \in \mathbb{N}^+ |\ \text{MDL}^{\varphi} \text{ has an } (\epsilon, \delta)\text{-PAC-learning performance with an } m\text{-sample dataset on learning problem } (f, \mathcal{P})\} \cup \{+\infty\}\big\}. \tag{6}$$

The PAC-learning guarantee given in Proposition 7 provides an upper-bound on the quantity defined in Equation 6.

Our main goal in this work is to study variations of the sample efficiency according to n the input-size of the underlying function to learn. We will thus parametrize our criterion with this quantity.

Moreover, we impose a practical restriction on the description-length of the function to learn.

Definition 9. *Given two interpreters φ and ψ, any $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, and any $n, d \in \mathbb{N}^+$, the* sample efficiency gain *of MDL^{φ} over MDL^{ψ} is*

$$G^{d}_{\varphi\to\psi}(\epsilon,\delta,n) \stackrel{\text{def}}{=} \sup_{f\in H^n, \mathcal{P}\in\Delta(\mathcal{B}^n)} \frac{m^{\epsilon,\delta}_{\psi}(f,\mathcal{P})}{m^{\epsilon,\delta}_{\varphi}(f,\mathcal{P})} \quad \textit{subject to} \quad |f|_{\varphi} \leq n^d. \tag{7}$$

The restriction $|f|_{\varphi} \leq n^d$ naturally applies on the interpreter for which we try to study a potential sample efficiency advantage, φ; the restriction ensures from the PAC-learning guarantee presented in Proposition 7 an (ϵ, δ)-PAC-learning performance with a number of samples polynomial in n.

We now define an auxiliary metric for two reasons. First, it is a lower bound on the sample efficiency gains and will be used as such in some theorems' proofs. Second, on some questions we only obtained partial results that are expressed through this metric, instead of sample efficiency gains.

Definition 10. *Given two interpreters φ and ψ, any $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, any $n, d \in \mathbb{N}^+$, and $a_1, a_2 > 0$ fixed in Proposition 7,*

$$\tilde{G}^{d}_{\varphi\to\psi}(\epsilon,\delta,n) \stackrel{\text{def}}{=} \sup_{f\in H^n, \mathcal{P}\in\Delta(\mathcal{B}^n)} \frac{m^{\epsilon,\delta}_{\psi}(f,\mathcal{P})}{\frac{a_1}{\epsilon}(\log\frac{1}{\delta} + |f|_{\varphi} + a_2)} \quad \textit{subject to} \quad |f|_{\varphi} \leq n^d. \tag{8}$$

This value can be interpreted as the best sample efficiency gains that can be obtained from MDL^{φ} over MDL^{ψ} while being provable from the guarantee given in Proposition 7.

Remark. *We defined G —the sample efficiency gain— as our metric of interest. However, in the literature —for example, the literature on depth-separation in ANNs— a usually assumed criterion of comparison is the necessary sizes of the hypotheses to fit functions, since these can usually be translated into PAC-learning guarantees.*

We refer the reader to Section Appendix B in the appendices for a complete development based on the gains in description-length, $\sup_{f\in H^n} \frac{|f|_{\psi}}{|f|_{\varphi}}$ instead of G; and, to Section Appendix D for links between constraints on the description-length of an hypothesis class and its VC-dimension for Turing machines and circuits.

4. Main results: comparison of Turing machines and circuits

Now that the formal background and the metric of interest are defined, we study the sample efficiency gains by learning with Turing machines or Boolean circuits interpreters under the minimum description length principle. All the results are given for Boolean circuits but also hold for ANNs. In other words, the Boolean circuits' interpreter $\mathcal{C}$ can be substituted for the ANNs' interpreter, given in Definition 41, in all the theorems that we give.

4.1. Sample efficiency gains of circuits over Turing machines

Before studying the sample efficiency gains of Turing machines over Boolean circuits, we present a partial result in the direction of showing that the sample efficiency gains of circuits over Turing machines are below a constant.

The following theorem shows that on any particular learning problem a PAC-guarantee obtained through Proposition 7 for learning with circuits will imply a similar PAC-guarantee for learning with Turing machines.

Theorem 11. *There exists a constant $q \in \mathbb{R}^+$ such that for all $\epsilon \in (0, 1/2)$, $\delta \in (0,1)$ and $n, d \in \mathbb{N}^+$,*

$$\tilde{G}^d_{\mathcal{C}\to\mathcal{U}}(\epsilon, \delta, n) \leq q. \tag{9}$$

The rest of the paper analyzes the converse question: can we prove an advantage of Turing-complete systems over circuits for learning in terms of sample efficiency gains?

4.2. Sample efficiency gains of Turing machines over circuits

In the next theorem, we show that we can construct a sequence of learning problems such that learning with Turing-complete languages becomes more and more advantageous in terms of sample efficiency over learning with Boolean circuits. Moreover, the advantage grows exponentially in the input-size of the function to learn.

Theorem 12. *For all $\epsilon \in (0, 1/2), \delta \in (0,1), d \in \mathbb{N}^+$ we have*

$$G^d_{\mathcal{U}\to\mathcal{C}}(\epsilon, \delta, n) \in \Omega(2^n/n). \tag{10}$$

The theorem shows that the potential advantage of learning with Turing machines over circuits can quickly become significant.

In the proof of Theorem 12, an explicit sequence of learning problems with an advantage for Turing machines is given, the functions to learn are defined by a Turing machine that enumerates on the functions in H^n and the Boolean circuits. The functions to be learned are thus hard to compute.

This can be seen as a practical limitation on the scope of this theorem. We will now use polynomial computational limits on our interpreter. Our formalism will use the interpreter of polynomial-time Turing machines —$\mathcal{U}^c$— to this effect.

4.3. Limits on the sample efficiency gains of polynomial-time Turing machine over circuits

In this new computationaly constrained setting, we prove a bound on the sample efficiency gains that can be shown from the length-based PAC-guarantee of Proposition 7.

Theorem 13. *For all* $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $c, d \in \mathbb{N}^+$ *we have*

$$\tilde{G}^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in O(n^c \log^2 n). \tag{11}$$

The proof of Theorem 13 is based on a result of [22], their result states that circuits can compute functions as fast as multi-tape Turing machines. More precisely, we use a refinement by [23] that takes into account the size of the involved Turing machine.

4.4. Sample efficiency gains of polynomial-time Turing machines over circuits are at least linear in the input-size

We show a positive result for learning with polynomial-time Turing machines, the sample efficiency gains grow at least (nearly) linearly in the input-size of the function to learn.

Theorem 14. *For all* $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $1 < c, d \in \mathbb{N}^+$ *and all* $\gamma > 0$ *we have*

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in \Omega(n^{1-\gamma}). \tag{12}$$

4.5. Are sample efficiency gains of polynomial-time Turing machines over circuits superlinear in the input-size?

The Theorems 13 and 14 open the question of whether the growth of the sample efficiency gains is actually a superlinear polynomial in the input-size of the function to learn.

As we will show, this question connects with open problems in Computational Complexity.

4.5.1. If gains are superlinear

The first open problem with which we make a connection is the existence of a problem in **P** for which superlinear sized circuits are necessary.

This problem is of importance in Computational Complexity. There are links between the collapse at the first and second level of the polynomial hierarchy and the computability of languages in **NP** by polynomial-sized families of Boolean circuits, see [24].

However, despite years of efforts, the maximal size-lower-bound known on Boolean circuits for a language in **NP** is linear, see [25, 26, 27].

The Theorem 15 shows that proving that the sample efficiency gains are actually superlinear through the PAC-guarantee offered by Proposition 7 solves this frontier. It solves the frontier by proving the existence of a problem in **P**, and thus **NP**, with superlinear circuit complexity.

Theorem 15. *If there exists* $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $c, d \in \mathbb{N}^+$ *and some* $\gamma > 0$ *such that*

$$\tilde{G}^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \notin O(n^{1+\gamma}) \tag{13}$$

then there exists a language in $\boldsymbol{P}$ *not computable by any sequence of Boolean circuits whose sizes are in* $O(n^{1+\tau})$ *for some* $\tau > 0$.

4.5.2. If gains are not superlinear

On the other hand, if the sample efficiency gains are not superlinear in the input-size of the function to learn for polynomial-time Turing machines over circuits then $\mathbf{P} \neq \mathbf{NP}$.

Theorem 16. *If for all* $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $c, d \in \mathbb{N}^+$ *and all* $\gamma > 0$

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in O(n^{1+\gamma}) \tag{14}$$

then $\boldsymbol{P \neq NP}$.

4.5.3. Gains are superlinear under circuit lower bounds

The contraposite of the last result yields superlinear gains in the case $\mathbf{P} = \mathbf{NP}$. However, $\mathbf{P} = \mathbf{NP}$ is not a common assumption in computer science. We propose an alternative in the following theorem. Let **E** be the set a language decidable with time-complexity $O(2^{O(n)})$ by Turing machines.

Theorem 17. *If there exists* $f \in \boldsymbol{E}$ *and* $\iota > 0$ *such that the Boolean circuits computing* f_n *are at least of size* $2^{\iota n}$*; then for any* $\gamma > 0$ *there exists* $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $c, d \in \mathbb{N}^+$ *such that we have*

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in \Omega(n^{1+\gamma}). \tag{15}$$

We note that the circuit lower-bounds of the assumption are the same as the ones arising in derandomization research, and the theorem is based on a worst-case to average-case hardness result [26].

5. Conclusion

In this work, we analyzed the sample efficiency gains of Turing machines over Boolean circuits and classical neural networks under the minimum description length principle in the PAC-learning framework. The Turing machines served as a proxy in the analysis for other Turing-complete systems, such as the recently proposed expressive Deep Learning architectures cited in the introduction, or programs (written in computationally universal languages) that can be learned from Inductive Programming techniques.

We showed that learning with expressive models such as Turing machines can yield sample efficiency gains that are exponential in the input-size of the function to learn. Learning with polynomial-time Turing machines can also yield PAC-learning profits in comparison to learning with circuits. The sample efficiency gains grow linearly in the input-size of the function to learn.

Whether they are superlinear or not is an open problem. One of our main results showed that if these sample efficiency gains are not superlinear in the input-size then $\mathbf{P} \neq \mathbf{NP}$. Another of our main contributions demonstrated that if it is possible to prove superlinear gains using a classical PAC-learning uniform convergence argument, then there exists a problem in $\mathbf{P}$ with superlinear circuit complexity. Additionally, under a circuit lower bound the gains are superlinear in the input-size.

A parallel investigation of the gains in terms of description-length to express Boolean functions is also an output of this research, given in Appendix B.

This paper also leaves some questions open. Some of our results offer bounds on $\tilde{G}$, thus providing information on the sample efficiency gains that are provable using the PAC-guarantee of Proposition 7. Improving these results by using G instead would render them independent of any particular PAC-guarantee.

An extension of the analysis to classically used DL architectures such as Convolutional Neural Networks and Recurrent Neural Networks would also broaden our insight on the potential advantage of using Turing-complete systems as models for learning. We leave this for further work.

References

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, arXiv preprint arXiv:1806.01261 (2018).

[2] K. Xu, J. Li, S. S. Du, K. ichi Kawarabayashi, S. Jegelka, What can neural networks reason about?, in: Proceedings of the International Conference on Learning Representations, 2020, pp. –.

[3] A. Graves, G. Wayne, I. Danihelka, Neural turing machines, arXiv preprint arXiv:1410.5401 (2014).

[4] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al., Hybrid computing using a neural network with dynamic external memory, Nature 538 (2016) 471.

[5] A. Joulin, T. Mikolov, Inferring algorithmic patterns with stack-augmented recurrent nets, in: Advances in neural information processing systems, 2015, pp. 190–198.

[6] Ł. Kaiser, I. Sutskever, Neural gpus learn algorithms, arXiv preprint arXiv:1511.08228 (2015).

[7] S. Sukhbaatar, J. Weston, R. Fergus, et al., End-to-end memory networks, in: Advances in neural information processing systems, 2015, pp. 2440–2448.

[8] K. Kurach, M. Andrychowicz, I. Sutskever, Neural random-access machines, ICLR (2016).

[9] I. Schlag, J. Schmidhuber, Learning to reason with third order tensor products, in: Advances in Neural Information Processing Systems, 2018, pp. 9981–9993.

[10] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, Ł. Kaiser, Universal transformers, arXiv preprint arXiv:1807.03819 (2018).

[11] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, T. Lillicrap, Relational recurrent neural networks, Advances in neural information processing systems 31 (2018) 7299–7310.

[12] J. R. Koza, R. Poli, Genetic programming, in: Search Methodologies, Springer, 2005, pp. 127–164.

[13] E. Kitzelmann, Inductive programming: A survey of program synthesis techniques, in: International workshop on approaches and applications of inductive programming, Springer, 2009, pp. 50–73.

[14] S. Gulwani, A. Polozov, R. Singh, Program Synthesis, volume 4, NOW, 2017.

[15] M. Telgarsky, Representation benefits of deep feedforward networks, arXiv preprint arXiv:1509.08101 (2015).

[16] S. Liang, R. Srikant, Why deep neural networks for function approximation?, arXiv preprint arXiv:1610.04161 (2016).

[17] R. Eldan, O. Shamir, The power of depth for feedforward neural networks, in: Conference on learning theory, 2016, pp. 907–940.

[18] B. Rossman, R. A. Servedio, L.-Y. Tan, An average-case depth hierarchy theorem for boolean circuits, in: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, IEEE, 2015, pp. 1030–1048.

[19] Y. Bengio, Y. LeCun, et al., Scaling learning algorithms towards ai, Large-scale kernel machines 34 (2007) 1–41.

[20] Y. Bengio, O. Delalleau, C. Simard, Decision trees do not generalize to new variations, Computational Intelligence 26 (2010) 449–467.

[21] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth, Occam's razor, Information processing letters 24 (1987) 377–380.

[22] N. Pippenger, M. J. Fischer, Relations among complexity measures, Journal of the ACM (JACM) 26 (1979) 361–381.

[23] C.-P. Schnorr, The network complexity and the turing machine complexity of finite functions, Acta Informatica 7 (1976) 95–107.

[24] R. M. Karp, R. J. Lipton, Some connections between nonuniform and uniform complexity classes, in: Proceedings of the twelfth annual ACM symposium on Theory of computing, ACM, 1980, pp. 302–309.

[25] K. Iwama, H. Morizumi, An explicit lower bound of 5n- o (n) for boolean circuits, in: International Symposium on Mathematical Foundations of Computer Science, Springer, 2002, pp. 353–364.

[26] S. Arora, B. Barak, Computational complexity: a modern approach, Cambridge University Press, 2009.

[27] S. Jukna, Boolean function complexity: advances and frontiers, volume 27, Springer Science & Business Media, 2012.

[28] F. C. Hennie, R. E. Stearns, Two-tape simulation of multitape turing machines, Journal of the ACM (JACM) 13 (1966) 533–546.

[29] A. Pavan, R. Santhanam, N. Vinodchandran, Some results on average-case hardness within the polynomial hierarchy, in: International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer, 2006, pp. 188–199.

[30] R. Kannan, Circuit-size lower bounds and non-reducibility to sparse sets, Information and control 55 (1982) 40–56.

[31] M. Li, P. Vitányi, et al., An introduction to Kolmogorov complexity and its applications, volume 4, Springer, 2019.

[32] G. S. Frandsen, P. B. Miltersen, Reviewing bounds on the circuit size of the hardest functions, Information processing letters 95 (2005) 354–357.

[33] V. N. Vapnik, A. Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, in: Measures of complexity, Springer, 2015, pp. 11–30.

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

[35] R. J. Solomonoff, A preliminary report on a general theory of inductive inference., Technical Report, Zator Company, Cambridge, MA, 1960.

[36] R. J. Solomonoff, An inductive inference code employing definitions., Technical Report, Rockford Research, Cambridge, MA, 1962.

[37] R. J. Solomonoff, A formal theory of inductive inference. part i, Information and control 7 (1964) 1–22.

[38] R. J. Solomonoff, A formal theory of inductive inference. part ii, Information and control 7 (1964) 224–254.

[39] A. N. Kolmogorov, Three approaches to the quantitative definition of information, Problems of information transmission 1 (1965) 1–7.

[40] J. v. Lint, Introduction to coding theory, Springer, 1999.

Appendix A. Main proofs

Proposition 7 (Description-length PAC-guarantee)**.** *There exists constants $a_1, a_2 > 0$ such that the following holds.*

For any interpreter φ and associated learning algorithm MDL$^\varphi$, any $(f, \mathcal{P})$ learning problem and any PAC-learning parameters $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, MDL$^\varphi$ has an (ϵ, δ)-PAC-learning performance with an m-sample dataset on the learning problem, where

$$m = \frac{a_1}{\epsilon}\left[\log\frac{1}{\delta} + |f|_\varphi + a_2\right]. \tag{5}$$

Proof. The algorithm MDL$^\varphi$ never outputs a function with a description-length larger than $|f|_\varphi$. There are at most $I = \sum_{i=0}^{|f|_\varphi} 2^i = 2^{|f|_\varphi+1} - 1$ functions in this set.

Let's compute an upper-bound on the probability that there exists a function $\tilde{f}$ of description-length smaller than $|f|_\varphi$ such that $acc_f^{\mathcal{P}}(\tilde{f}) < 1 - \epsilon$ and which is consistent with a dataset composed of $m \geq \frac{1}{\epsilon}\left[\log\frac{I}{\delta}\right]$ samples.

For any function $\tilde{f}$ with $acc_f^{\mathcal{P}}(\tilde{f}) < 1 - \epsilon$ the probability to be consistent with the dataset is upper-bounded by $(1-\epsilon)^m$ since each sample is drawn independently according to $\mathcal{P}$.

By the union bound the probability of the existence of one low accuracy function consistent with the data is thus upper-bounded by $I(1-\epsilon)^m$.

Which develops in $I(1-\epsilon)^m \leq Ie^{-\epsilon m} \leq \delta$.

Moreover, there exists constants $a_1, a_2 > 0$ s.t

$$\frac{1}{\epsilon}\left[\log\frac{I}{\delta}\right] \leq \frac{a_1}{\epsilon}\left[\log\frac{1}{\delta} + |f|_\varphi + a_2\right] \tag{A.1}$$

independently of the interpreter, learning problem, and PAC-learning parameters. □

Theorem 11. *There exists a constant $q \in \mathbb{R}^+$ such that for all $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$ and $n, d \in \mathbb{N}^+$,*

$$\tilde{G}^d_{\mathcal{C}\to\mathcal{U}}(\epsilon, \delta, n) \leq q. \tag{9}$$

Proof. From Definition 10 of $\tilde{G}$

$$\tilde{G}^d_{\mathcal{C}\to\mathcal{U}}(\epsilon, \delta, n) = \sup_{f\in H^n, \mathcal{P}\in\Delta(\mathcal{B}^n)} \frac{m^{\epsilon,\delta}_{\mathcal{U}}(f, \mathcal{P})}{\frac{a_1}{\epsilon}(\log\frac{1}{\delta} + |f|_{\mathcal{C}} + a_2)} \quad \text{subject to} \quad |f|_{\mathcal{C}} \leq n^d. \tag{A.2}$$

For any n, consider any learning problem $(f \in H^n, \mathcal{P} \in \Delta(\mathcal{B}^n))$.

By the Definition 8 of $m_\varphi^{\epsilon,\delta}$ and the PAC-guarantee given in Proposition 7, for any $\epsilon \in (0, 1/2)$ and $\delta \in (0, 1)$, we have

$$m_{\mathcal{U}}^{\epsilon,\delta}(f, \mathcal{P}) \leq \frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f|_{\mathcal{U}} + a_2). \tag{A.3}$$

By the main theorem of Kolomogorov complexity, Proposition 49, we have $|f|_{\mathcal{U}} \leq |f|_{\mathcal{C}} + K$ for some constant K independent of f. The PAC-learning guarantee for $MDL^{\mathcal{C}}$ can thus be transformed in a guarantee for $MDL^{\mathcal{U}}$ since

$$\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f|_{\mathcal{U}} + a_2) \leq \frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f|_{\mathcal{C}} + K + a_2). \tag{A.4}$$

Following these inequalities, we get

$$\tilde{G}^d_{\mathcal{C}\to\mathcal{U}}(\epsilon, \delta, n) \leq \frac{\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f|_{\mathcal{C}} + K + a_2)}{\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f|_{\mathcal{C}} + a_2)} \leq \frac{K}{a_2} + 1. \tag{A.5}$$

□

Theorem 12. *For all $\epsilon \in (0, 1/2), \delta \in (0, 1), d \in \mathbb{N}^+$ we have*

$$G^d_{\mathcal{U}\to\mathcal{C}}(\epsilon, \delta, n) \in \Omega(2^n/n). \tag{10}$$

Proof. By Definition 9 of the sample efficiency gains, we have

$$G^d_{\mathcal{U}\to\mathcal{C}}(\epsilon, \delta, n) = \sup_{f\in H^n, \mathcal{P}\in\Delta(\mathcal{B}^n)} \frac{m_{\mathcal{C}}^{\epsilon,\delta}(f, \mathcal{P})}{m_{\mathcal{U}}^{\epsilon,\delta}(f, \mathcal{P})} \quad \text{subject to} \quad |f|_{\mathcal{U}} \leq n^d. \tag{A.6}$$

For any $\epsilon \in (0, 1/2), \delta \in (0, 1), d \in \mathbb{N}^+$, we define a sequence of learning problems which prove the statement. For any n, we define a learning problem to solve. For any n, the learning problem is to learn under the uniform distribution, U, the binary function computed by the interpretation of the following program.

For input x of size n:

1. *Compute the input size n.*
2. *Enumerate all the functions in H^n in some fixed lexicographic order. For each of these functions, f:*

(a) *Compute the hypothesis of minimal-description-length according to the interpreter* $\mathcal{C}$ *to represent a function* $\hat{f}$ *such that* $acc_f^U(\hat{f}) \geq 1-\epsilon$. *Rewritten*

$$\begin{aligned} &\min_{h\in\mathcal{B}^*} && |h| \\ &\textit{subject to} && acc_f^U(\mathcal{C}(h,.)) \geq 1-\epsilon. \end{aligned} \tag{A.7}$$

(b) *With* $H(.)$ *the binary entropy function, see Definition 54, if the optimal description-length is bigger than* $2^n(1-H(\epsilon))-2$ *then return* $f(x)$.

We first show that this is a well-defined computable function in the sense that the step *(b)* will always be satisfied for some function in the enumeration for all n sufficiently large.

The next development shows by a counting argument, that all the binary functions cannot be approximated within $1-\epsilon$ by Boolean circuits of binary description-length smaller than $2^n(1-H(\epsilon))-2$.

The number of functions that can approximated with accuracy higher than $1-\epsilon$ by Boolean circuits of description-length smaller than $2^n(1-H(\epsilon))-2$ is

$$\left| \bigcup_{l\in 0,\ldots,\lfloor 2^n(1-H(\epsilon))-2\rfloor} \bigcup_{h\in\mathcal{B}^l} \left\{ f\in H^n \middle|\, acc_f^U(\mathcal{C}(h,.)) \geq 1-\epsilon \right\} \right|. \tag{A.8}$$

Using Proposition 55 and the fact that there are 2^{2^n} functions in H^n,

$$\begin{aligned} &\left| \bigcup_{l\in 0,\ldots,\lfloor 2^n(1-H(\epsilon))-2\rfloor} \bigcup_{h\in\mathcal{B}^l} \left\{ f\in H^n \middle|\, acc_f^U(\mathcal{C}(h,.)) \geq 1-\epsilon \right\} \right| \\ &\quad \leq \sum_{l\in 0,\ldots,\lfloor 2^n(1-H(\epsilon))-2\rfloor} \sum_{h\in\mathcal{B}^l} \left| \left\{ f\in H^n \middle|\, acc_f^U(\mathcal{C}(h,.)) \geq 1-\epsilon \right\} \right| \\ &\quad = \sum_{l\in 0,\ldots,\lfloor 2^n(1-H(\epsilon))-2\rfloor} \sum_{h\in\mathcal{B}^l} \sum_{i\in 0,\ldots,\lfloor \epsilon 2^n\rfloor} \binom{2^n}{i} \\ &\quad \leq \sum_{l\in 0,\ldots,\lfloor 2^n(1-H(\epsilon))-2\rfloor} \sum_{h\in\mathcal{B}^l} 2^{2^n H(\epsilon)} \\ &\quad \leq 2^{2^n(1-H(\epsilon))-1} 2^{2^n H(\epsilon)} \\ &\quad = 2^{2^n-1} < 2^{2^n} = |H^n|. \end{aligned}$$

Thus the condition in step *(b)* will always be satisfied for some function.

Also, the condition $|f|_{\mathcal{U}} \leq n^d$ will always be satisfied for all n large enough since the learning problem's function corresponds to a program/Turing-machine of fixed description-length.

Moreover, using the guarantee of Proposition 7, $m_{\mathcal{U}}^{\epsilon,\delta}(f,\mathcal{P}) \leq \frac{a_1}{\epsilon}(\log\frac{1}{\delta} + |f|_{\mathcal{U}} + a_2)$, we deduce that $m_{\mathcal{U}}^{\epsilon,\delta}(f,\mathcal{P})$ is upper-bounded by a constant independent of n.

Also by construction, $MDL^{\mathcal{C}}$ has to select circuits of description-length growing at least as fast as $2^n(1-H(\epsilon))-2$ to be able to approximate the function to learn with an average error at most ϵ. We show that this condition has implications on the size of the minimal circuit that has to be selected and then on the minimal number of samples needed.

By Definition 37, for all n sufficiently large, all circuits of size lower than $\alpha 2^n/n$, for any $\alpha > 0$, can be described with the Boolean circuit interpreter $\mathcal{C}$ such that their description-length is lower than

$$9\alpha\frac{2^n}{n}\log(\alpha 2^n/n). \tag{A.9}$$

Thus, for all n sufficiently large, the description-lengths of these circuits are lower than

$$9\alpha\left[\log(2)2^n + \log(\alpha)\frac{2^n}{n} - \frac{2^n}{n}\log n\right] \leq 9\alpha\log(2\alpha)2^n. \tag{A.10}$$

Thus there exists a α sufficiently small such that, for all n sufficiently large, the description-lengths of these circuits are smaller than $2^n(1-H(\epsilon))-2$.

Consequently, all these circuits with size at most $\alpha 2^n/n$ must be eliminated by $MDL^{\mathcal{C}}$. This requires at least $\frac{C}{b}2^n/n$ samples, for some fixed $b > 0$, by Proposition 53. □

Theorem 13. *For all $\epsilon \in (0, 1/2)$, $\delta \in (0,1)$, $c, d \in \mathbb{N}^+$ we have*

$$\tilde{G}_{\mathcal{U}^c\to\mathcal{C}}^{d}(\epsilon,\delta,n) \in O(n^c\log^2 n). \tag{11}$$

Proposition 18. *[22, 23]. If a multi-tape Turing machine M computes a function on inputs of size n within t steps then there exists a Boolean circuit of size at most $\alpha(\text{number of rules of } M)t\log t$ that computes the same function, where α depends only on the number of tapes and the alphabet size of the Turing machine.*

Proof of Theorem 13. By Definition 10 of $\tilde{G}$

$$\tilde{G}^{d}_{\mathcal{U}^c\to\mathcal{C}}(\epsilon,\delta,n) = \sup_{f\in H^n,\mathcal{P}\in\Delta(\mathcal{B}^n)} \frac{m^{\epsilon,\delta}_{\mathcal{C}}(f,\mathcal{P})}{\frac{a_1}{\epsilon}(\log\frac{1}{\delta}+|f|_{\mathcal{U}^c}+a_2)} \quad \text{subject to} \quad |f|_{\mathcal{U}^c}\leq n^d. \tag{A.11}$$

We will upper-bound the ratio for any learning problem ($f\in H^n, \mathcal{P}\in\Delta(\mathcal{B}^n)$). Note that only functions f corresponding to finite $|f|_{\mathcal{U}^c}$ have to be considered.

We use Proposition 34 on the Turing machine of the interpreter $\mathcal{U}^c$ and the input h of length $|f|_{\mathcal{U}^c}$ such that $\mathcal{U}^c(h,.)=f$. From the application of the proposition, we deduce that there exists a Turing machine with at most $\rho\,|f|_{\mathcal{U}^c}$ rules that compute f, where $\rho>0$ is a parameter independent of f and n. Moreover, the proposition tells us that the resulting Turing machine has the same computational time-limit as $\mathcal{U}^c$, and is thus also bounded by βn^c, for some $\beta>0$.

Applying now Proposition 18 on these facts, we deduce that there exists a Boolean circuit computing f of size at most, with $t=\beta n^c$,

$$\alpha\rho\,|f|_{\mathcal{U}^c}\,t\log t, \tag{A.12}$$

where $\alpha>0$ is independent of f and n.

By Definition 37, the description-length of this circuit with the Boolean circuit interpreter $\mathcal{C}$, and thus $|f|_{\mathcal{C}}$, will be upper bounded by

$$2\lceil\log_2(n)\rceil+2+\underbrace{\alpha\rho\,|f|_{\mathcal{U}^c}\,t\log t(3+\max\{2\lceil\log_2\alpha\rho\,|f|_{\mathcal{U}^c}\,t\log t\rceil),\lceil\log_2 n\rceil\}}_{\stackrel{\text{def}}{=}B}. \tag{A.13}$$

We use the PAC-guarantee of Proposition 7 to upper-bound $m^{\epsilon,\delta}_{\mathcal{C}}(f,\mathcal{P})$ with the upper-bound on $|f|_{\mathcal{C}}$ of Equation A.13.

Then the quantity of Equation A.11 is also upper-bounded

$$\begin{aligned}\tilde{G}_{\mathcal{U}^c\to\mathcal{C}}(\epsilon,\delta,n) &\leq \sup_{f\in H^n}\frac{\frac{a_1}{\epsilon}(\log\frac{1}{\delta}+2\lceil\log_2(n)\rceil+2+B+a_2)}{\frac{a_1}{\epsilon}(\log\frac{1}{\delta}+|f|_{\mathcal{U}^c}+a_2)}\ \text{s.t.}\ |f|_{\mathcal{U}^c}\leq n^d,\\ &\leq \sup_{f\in H^n}\frac{B}{|f|_{\mathcal{U}^c}}+\frac{2}{a_2}\lceil\log_2(n)\rceil+\frac{2}{a_2}+1\ \text{s.t.}\ |f|_{\mathcal{U}^c}\leq n^d.\end{aligned} \tag{A.14}$$

Where $\frac{B}{|f|_{\mathcal{U}^c}}$ equals

$$\alpha\rho t \log t(3 + \max\{2\lceil \log_2 \alpha\rho\, |f|_{\mathcal{U}^c}\, t \log t\rceil), \lceil \log_2 n\rceil\}. \tag{A.15}$$

Using $|f|_{\mathcal{U}^c} \leq n^d$ and $t = \beta n^c$, the value in Equation A.15 is in

$$O(n^c \log^2 n). \tag{A.16}$$

Returning this result to the inequalities of Equation A.14, we obtain

$$\tilde{G}_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in O(n^c \log^2 n). \tag{A.17}$$

□

We note that the Theorem 13 can be improved upon by using another definition for the polynomial-time universal Turing machine, $\mathcal{U}^c$. The Definition 32 use the construction of [28]. Another possibility is to use the construction of [22] related to Proposition 18. This construction would make $\mathcal{U}^c$ oblivious and directly transformable into a Boolean circuit without the additional $\log n$ factor. The final bound would thus be in $O(n^c \log n)$ instead of $O(n^c \log^2 n)$.

Theorem 14. *For all $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $1 < c, d \in \mathbb{N}^+$ and all $\gamma > 0$ we have*

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in \Omega(n^{1-\gamma}). \tag{12}$$

Proof. By Definition 9, the sample efficiency gain is

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) = \sup_{\substack{f \in H^n, \mathcal{P} \in \Delta(\mathcal{B}^n) \\ \text{subject to} \quad |f|_{\mathcal{U}^c} \leq n^d.}} \frac{m^{\epsilon,\delta}_{\mathcal{C}}(f, \mathcal{P})}{m^{\epsilon,\delta}_{\mathcal{U}^c}(f, \mathcal{P})} \tag{A.18}$$

Fix any combination of $\epsilon \in (0, 1/2), \delta \in (0, 1), 1 < c, d \in \mathbb{N}^+$.

For any input size n, we contruct the following learning problem, ($f \in H^n, \mathcal{P} \in \Delta(\mathcal{B}^n)$). The function to learn f is the parity function on $\mathcal{B}^n$, noted $\oplus$ (it is the number of 1 in the input modulo 2). The probability measure on the domain $\mathcal{B}^n$, $\mathcal{P}$, is the uniform distribution U^n. Let $\oplus|_n$ denote the function $\oplus$ restricted to size n inputs.

There exists a fixed Turing machine computing the parity function in linear time for all n. By Definition 32, for $c > 1$ there exists an $h \in \mathcal{B}^*$ such

that $\mathcal{U}^c(h,.)$ computes $\oplus$ for all n large enough. Thus $|\oplus|_n|_{\mathcal{U}^c}$ is at most some constant.

A first consequence is that, for n large enough, $|\oplus|_n|_{\mathcal{U}^c} \leq n^d$ will be satisfied for the form of learning problem we defined.

A second consequence, using the PAC-guarantee given in Proposition 7, is that the denominator $m^{\epsilon,\delta}_{\mathcal{U}^c}(f,\mathcal{P}) \leq \frac{a_1}{\epsilon}(\log\frac{1}{\delta} + |\oplus|_n|_{\mathcal{U}^c} + a_2)$ is less than a constant.

Now for the numerator, for any $\gamma > 0$ take any sampling of size less than $n^{1-\gamma}$. By Proposition 53 and Definition 37, we know that a Boolean circuit of size less than $bn^{1-\gamma}$ will be selected by $MDL^{\mathcal{C}}$ for some fixed $b > 0$.

For n sufficiently large $bn^{1-\gamma} < n$, and thus the circuit selected by $MDL^{\mathcal{C}}$ will not depend on all the inputs' variables. Suppose without loss of generality that x_n is one of these variables to which the circuit is not sensible. The accuracy of a function C computed by such a selected circuit on our learning problem is

$$\begin{aligned} acc^{U^n}_{\oplus}(C) &= \frac{1}{2^n}\sum_{x\in\mathcal{B}^n} \oplus(x) = C(x_1,\ldots,x_{n-1},x_n) \\ &= \frac{1}{2^n}\sum_{x^-\in\mathcal{B}^{n-1}}\sum_{x_n\in\mathcal{B}} \oplus(x^-,x_n) = C(x_1^-,\ldots,x_{n-1}^-) \\ &= \frac{1}{2^n}\sum_{x^-\in\mathcal{B}^{n-1}} 1 \\ &= \frac{1}{2}. \end{aligned} \tag{A.19}$$

Since $\epsilon < 1/2$, it is impossible for $MDL^{\mathcal{C}}$ to get a sufficient accuracy with only $n^{1-\gamma}$ samples for n sufficiently large, and so $m^{\epsilon,\delta}_{\mathcal{C}}(f,\mathcal{P}) \in \Omega(n^{1-\gamma})$. □

Theorem 15. *If there exists $\epsilon \in (0, 1/2)$, $\delta \in (0,1)$, $c,d \in \mathbb{N}^+$ and some $\gamma > 0$ such that*

$$\tilde{G}^d_{\mathcal{U}^c\to\mathcal{C}}(\epsilon,\delta,n) \notin O(n^{1+\gamma}) \tag{13}$$

then there exists a language in $\boldsymbol{P}$ not computable by any sequence of Boolean circuits whose sizes are in $O(n^{1+\tau})$ for some $\tau > 0$.

Proof. Let's suppose superlinear gains $\tilde{G}^d_{\mathcal{U}^c\to\mathcal{C}}(\epsilon,\delta,n) \notin O(n^{1+\gamma})$ for some $\epsilon \in (0,1/2)$, $\delta \in (0,1)$, $c,d \in \mathbb{N}^+$ and $\gamma > 0$.

Recall that by Definition 10, for some constants a_1, $a_2 > 0$ defined in Proposition 7,

$$\tilde{G}^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) = \sup_{f \in H^n, \mathcal{P} \in \Delta(\mathcal{B}^n)} \frac{m^{\epsilon,\delta}_{\mathcal{C}}(f, \mathcal{P})}{\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f|_{\mathcal{U}^c} + a_2)} \quad \text{subject to} \quad |f|_{\mathcal{U}^c} \leq n^d. \tag{A.20}$$

We give the proof outline:

1. First, we prove that the premise of the theorem's statement implies the existence of an infinite sub-sequence of functions with description-length gains superlinear in the input-size. This result appears in Equation A.25.
2. Second, we present a fixed program of polynomial computational complexity.
3. Third, we show lower-bounds on some input-sizes for this program with the interpreter for circuits. These lower-bounds come from the developments in the first point.
4. Fourth, we show from these lower-bounds that the description-length of the program with the circuit interpreter is at least superlinear in the input-size.
5. Fifth, we conclude by showing superlinear circuit complexity for the presented program.

For any n, the sets H_n are finite and thus, for any n, the supremum can be attained for some $f_n \in H_n$. Then there exists a sequence of functions $(f_n) = f_1 \in H_1, \ldots, f_n \in H_n, \ldots$ such that the sequence in n

$$\sup_{\mathcal{P} \in \Delta(\mathcal{B}^n)} \frac{m^{\epsilon,\delta}_{\mathcal{C}}(f_n, \mathcal{P})}{\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f_n|_{\mathcal{U}^c} + a_2)} \tag{A.21}$$

is not in $O(n^{1+\gamma})$, and, morevover, with $f_n \leq n^d$ for all n.

By contraposition of the PAC-guarantee offered by Proposition 7, for all n and $\mathcal{P}$, $\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f_n|_{\mathcal{C}} + a_2) \geq m^{\epsilon,\delta}_{\mathcal{C}}(f_n, \mathcal{P})$, and thus

$$\frac{\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f_n|_{\mathcal{C}} + a_2)}{\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |f_n|_{\mathcal{U}^c} + a_2)} \notin O(n^{1+\gamma}). \tag{A.22}$$

Since ϵ and δ are fixed, the sequence of function (f_n) satisfies

$$\frac{|f_n|_{\mathcal{C}}}{|f_n|_{\mathcal{U}^c}} \notin O(n^{1+\gamma}). \tag{A.23}$$

We now restrict n to the indices that form a sub-sequence of (f_n) such that

$$\frac{|f_n|_{\mathcal{C}}}{|f_n|_{\mathcal{U}^c}} \in \Omega(n^{1+\gamma/2}). \tag{A.24}$$

Let $N \subseteq \mathbb{N}$ denote the set of such indices. Notice that such a restriction remove any function f such that $|f|_{\mathcal{U}^c} = +\infty$ from the sequence.

By definition of the big-Ω notation, there exists some $b > 0$ such that for all n sufficiently large

$$|f_n|_{\mathcal{C}} \geq b\,|f_n|_{\mathcal{U}^c}\, n^{1+\gamma/2}. \tag{A.25}$$

We will now define a Boolean function computable in polynomial-time and, using Equation A.25, prove a superlinear circuit complexity for it.

This Boolean function will be noted I and is defined by

$$I(\langle x_1, x_2\rangle) = \mathcal{U}^c(x_1, x_2), \tag{A.26}$$

where $\langle .,. \rangle : \mathcal{B}^* \times \mathcal{B}^* \to \mathcal{B}|\, \langle x_1, x_2\rangle = \underbrace{0\ldots0}_{|x_2|}1x_1x_2$. The encoding $\langle .,. \rangle$ is bijective and invertible in polynomial-time. Thus I is computable in polynomial-time since $\mathcal{U}^c$ is also computable in polynomial-time.

We prove a lower-bound on the description-length of I with the Boolean circuit interpreter on some inputs' sizes as we show next by contradiction. We denote $I|_{2n+1+|f_n|_{\mathcal{U}^c}}$ the Boolean function I restricted to $2n+1+|f_n|_{\mathcal{U}^c}$ sized-inputs. We will show $\left|I|_{2n+1+|f_n|_{\mathcal{U}^c}}\right|_{\mathcal{C}} \geq |f_n|_{\mathcal{C}}$.

Suppose there exists some Boolean circuit that computes $I|_{2n+1+|f_n|_{\mathcal{U}^c}}$, and that its description-length is strictly lower than $|f_n|_{\mathcal{C}}$, i.e. $\left|I|_{2n+1+|f_n|_{\mathcal{U}^c}}\right|_{\mathcal{C}} < |f_n|_{\mathcal{C}}$.

For all n, let $p_n \in \mathcal{B}^{|f|_{\mathcal{U}^c}}$ be the Boolean string such that $\mathcal{U}^c(p_n, .) = f_n$. By definition of I (Equation A.26) the function $I(\langle p_n, .\rangle)$ computes f_n, i.e. for all $x \in \mathcal{B}^n$ we have $I(\langle p_n, x\rangle) = f_n(x)$.

For the supposed circuit, we hardwire the $n+2$ to the $n+2+|f_n|_{\mathcal{U}^c}$ inputs' variables to p_n. This force the circuit to compute f_n as shown.

When we hardwire some of the inputs' variables, the Boolean circuit size can only diminish. By Definition 37 of the Boolean circuit interpreter, the description-length of a circuit is an increasing monotone function of its size. So, when we hardwire p_n in the input, the circuit can only diminish in description-length. This implies that $|f_n|_{\mathcal{C}} \leq \left|I|_{2n+1+|f_n|_{\mathcal{U}^c}}\right|_{\mathcal{C}} < |f_n|_{\mathcal{C}}$, this inequality is a contradiction. Consequently, we must have

$$\left|I|_{2n+1+|f_n|_{\mathcal{U}^c}}\right|_{\mathcal{C}} \geq |f_n|_{\mathcal{C}}\,. \tag{A.27}$$

With our lower-bounds on circuits' description-lengths of Equation A.25, it gives, for all n sufficiently large,

$$\left|I|_{2n+1+|f_n|_{\mathcal{U}^c}}\right|_{\mathcal{C}} \geq bn^{1+\gamma/2}\,|f_n|_{\mathcal{U}^c}\,. \tag{A.28}$$

We cannot conclude the Theorem directly from this, the fact that $|f_n|_{\mathcal{U}^c}$ can vary with n complexifies the analysis. The rest of the proof address this issue by identifying an infinite subsequence of (f_n) for which the evolution of $|f_n|_{\mathcal{U}^c}$ has a tight characterization.

We distribute the sequence of learning problems' functions, f_n, in different sets. For some precision parameter $\nu > 0$, we define $i^{\max} = \lceil \frac{1}{\nu} \rceil d$, the sequence of indices $i = 1, \ldots, i^{\max}$, the following sets

$$S_i = \left\{ n \in N | \, \frac{i-1}{i^{\max}} n^{\frac{i-1}{i^{\max}}d} < |f_n|_{\mathcal{U}^c} \leq \frac{i}{i^{\max}} n^{\frac{i}{i^{\max}}d} \right\}, \tag{A.29}$$

and S_0 containing the unique possible description-length of 0 function.

We have the upper-bound on the description-length of Equation A.20, for all n, $|f_n|_{\mathcal{U}^c} \leq n^d$. Thus, the functions in the infinite sequence (f_n) are well partitioned in the defined sets. By the pigeon-hole principle there exists an index i^* such that S_{i^*} is of infinite size.

We now restrict all n to be in S_{i^*}. We prove that the description-length of the program I is superlinear in the input-size with the interpreter for circuits for this sub-sequence of indices.

We distingish three cases that cover all the possibilities for i^*:

1. Zero-length $i^* = 0$.
   By Definition 29 of Turing machines, a Turing machine has at least two states. By Definition 30 of the encoder for Turing machines, a Turing machine with at least two states has at least a description-length of two bits. From these two facts and by Definition 32 of $\mathcal{U}^c$, a zero-length description interpreted by $\mathcal{U}^c$ outputs $\perp$. Thus, for all n and any function $f \in H^n$, $|f|_{\mathcal{U}^c} > 0$. Consequently, the set S_0 is empty and this case is not possible.
2. Sub-linear $i^* \leq \lceil 1/\nu \rceil = i^{\max}/d \to \frac{i^*}{i^{\max}} d \leq 1$.
   We provide a lower-bound on the power linking the input-size to the circuits' description-lengths lower-bounds given in Equation A.25.
   We note that we have, $0 < |f_n|_{\mathcal{U}^c} \leq \alpha n$ for $\alpha = i^*/i^{\max}$ by Equation A.29.

Using properties of the logarithm, for all $\kappa_1, \kappa_2 > 0$, and all n sufficiently large,

$$\begin{aligned}\log_{2n+1+|f_n|_{\mathcal{U}^c}} bn^{1+\gamma/2}\,|f_n|_{\mathcal{U}^c} &\geq \log_{(2+\alpha)n+1} n^{1+\gamma/2} + \log_{2n+1+|f_n|_{\mathcal{U}^c}} b + \log_{2n+1+|f_n|_{\mathcal{U}^c}} |f_n|_{\mathcal{U}^c} \\ &\geq \frac{\log_n n^{1+\gamma/2}}{\log_n((2+\alpha)n+1)} - \kappa_1 + 0 \\ &\geq \frac{1+\gamma/2}{1+\kappa_2} - \kappa_1.\end{aligned} \tag{A.30}$$

Since $\gamma > 0$, there exists κ_1, κ_2 small enough such that this lower bound is strictly greater than one.

3. Superlinear $i^* > \lceil 1/\nu \rceil \to i^* - 1 \geq \lceil 1/\nu \rceil = i^{\max}/d \to \frac{i^*-1}{i^{\max}} d \geq 1$.
We pose $q = \frac{i^*}{i^{\max}} d$, $\nu' = \frac{1}{\lceil 1/\nu \rceil} \leq \nu$, and $\alpha_1 = \frac{i^*-1}{i^{\max}}, \alpha_2 = \frac{i^*}{i^{\max}}$.
The following holds, by definition of S_{i^*} in Equation A.29, $\alpha_1 n^{q-\nu'} \leq |f_n|_{\mathcal{U}^c} \leq \alpha_2 n^q$. Also, we have $q \geq 1$ and $q - \nu' = \frac{i^*-1}{i^{\max}} d \geq 1$.
In this case the power linking the input-size to the circuits' description-lengths lower-bounds is, for all $\kappa_1, \kappa_2, \kappa_3 > 0$ and all sufficiently large n,

$$\begin{aligned}\log_{2n+1+|f_n|_{\mathcal{U}^c}} bn^{1+\gamma/2}\,|f_n|_{\mathcal{U}^c} &\geq \log_{2n+1+|f_n|_{\mathcal{U}^c}} \alpha_1 bn^{1+\gamma/2+q-\nu'} \\ &\geq \log_{(2+\alpha_2)n^q+1} n^{1+\gamma/2+q-\nu'} + \log_{2n+1+|f_n|_{\mathcal{U}^c}} \alpha_1 b \\ &\geq \frac{\log_{n^q} n^{1+\gamma/2+q-\nu'}}{\log_{n^q}((2+\alpha_2)n^q+1)} - \kappa_1 \\ &\geq \frac{1+\frac{1+\gamma/2-\nu'}{q}}{1+\log_{n^q}(2+\alpha_2)+\kappa_2} - \kappa_1 \\ &\geq \frac{1+\frac{1+\gamma/2}{d}}{1+\kappa_2+\kappa_3} - \frac{\nu}{1+\kappa_2+\kappa_3} - \kappa_1.\end{aligned} \tag{A.31}$$

Our reasoning can be taken with arbitrarily small ν and $\kappa_1, \kappa_2, \kappa_3$, such that the lower bound can be made strictly greater than one.

All the possible cases have been treated.

The proved bound on the Boolean circuits' description-length extends to their sizes. More precisely, any superlinear lower-bound of the type $n^{1+\iota}$, for some $\iota > 0$, on the description-length of the Boolean circuits with interpreter

$\mathcal{C}$ implies, for all n sufficiently large, a similar lower-bound, $n^{1+\tau}$, for some $0 < \tau < \iota$, for the Boolean circuits' sizes by Definition 37.

This finishes the proof. □

Theorem 16. *If for all* $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $c, d \in \mathbb{N}^+$ *and all* $\gamma > 0$

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in O(n^{1+\gamma}) \tag{14}$$

then $\boldsymbol{P \neq NP}$.

Proposition 19. *[29]. For any* $k_1, k_2 \in \mathbb{N}^+$, *there exists a language* $L \in \boldsymbol{P}^{\Sigma_2^p}$ *such that for every circuit sequence* $(C_1, \ldots, C_n, \ldots)$ *whose circuits' sizes are at most* n^{k_1}, *the following holds*

$$\Pr_{x \in U(\mathcal{B}^n)} [L(x) = C_n(x)] \leq 1/2 + 1/n^{k_2}, \tag{A.32}$$

where $U(\mathcal{B}^n)$ *denotes the uniform distribution on* $\mathcal{B}^n$.

Proof of Theorem 16. By contraposition, we suppose $\mathbf{P} = \mathbf{NP}$, then the polynomial hierarchy collapses, $\mathbf{P} = \mathbf{PH}$, and thus in particular $\mathbf{P} = \mathbf{P}^{\Sigma_2^p}$.

Implying with Proposition 19 that for all k_1 and k_2 there exists a language in $\mathbf{P}$ such that for all n there does not exist a Boolean circuit of size smaller than n^{k_1} which approximate the language with accuracy at least $\frac{1}{2} + \frac{1}{n^{k_2}}$ under the uniform distribution.

Fix $\epsilon = 1/4$, any $\delta \in (0, 1)$, any $d \in \mathbb{N}^+$, and $\gamma = 1$.

Take $k_1 = 2$ and $k_2 = 1$. For any $n \geq 4$, to select a function of error rate at most $1/4$ all circuits of size lower than n^2 must be eliminated. By Definition 37 of the Boolean circuits' interpreter and Proposition 53, the link between circuits' size and description-length is an increasing monotonic function, and thus, at least n^2/b samples will be necessary to eliminate all these circuits for the learning algorithm $MDL^{\mathcal{C}}$, for some fixed constant $b > 0$.

Moreover, by Definition 32 of $\mathcal{U}^c$, since the language is in $\mathbf{P}$ there exists some $c \in \mathbb{N}^+$ and some $s \in \mathcal{B}^*$ such that the language is computed by $\mathcal{U}^c(s, .)$. For any $d \in \mathbb{N}^+$ and for all sufficiently large n, we have $|s| \leq n^d$.

Thus we have some combination of parameters for which, for some constant $|s|$,

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \geq \frac{n^2/b}{\frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |s| + a_2)} \in \Omega(n^2). \tag{A.33}$$

□

Theorem 17. *If there exists $f \in \boldsymbol{E}$ and $\iota > 0$ such that the Boolean circuits computing f_n are at least of size $2^{\iota n}$; then for any $\gamma > 0$ there exists $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$, $c, d \in \mathbb{N}^+$ such that we have*

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) \in \Omega(n^{1+\gamma}). \tag{15}$$

Proposition 20. *[26]. Let $S : \mathbb{N} \to \mathbb{N}$ and $f \in \boldsymbol{E}$ such that Boolean circuits that decide $f|_n$ are at least of sizes $S(n)$ for every n. Then there exists a function $g \in \boldsymbol{E}$ and a constant $b > 0$ such that approximating $g|_n$ under the uniform distribution with accuracy at least 0.99 requires Boolean circuits of sizes at least $S(n/b)/n^b$ for every sufficiently large n.*

Proof of Theorem 17. By Definition 9, the sample efficiency gain is

$$G^d_{\mathcal{U}^c \to \mathcal{C}}(\epsilon, \delta, n) = \sup_{f \in H^n, \mathcal{P} \in \Delta(\mathcal{B}^n)} \frac{m^{\epsilon,\delta}_{\mathcal{C}}(f, \mathcal{P})}{m^{\epsilon,\delta}_{\mathcal{U}^c}(f, \mathcal{P})} \quad \text{subject to} \quad |f|_{\mathcal{U}^c} \leq n^d. \tag{A.34}$$

Let's define a sequence of learning problems entailing our theorem.

By Proposition 20 and the assumption there exists a language $g \in \mathbf{E}$ such that $g|_n$ can only be approximated with accuracy 0.99 under the uniform distribution by Boolean circuits of size at least in $\Omega(2^{\Omega(n)}/n^b)$ for some constant $b > 0$.

We define g' to be the application of g on the $a \log n$ first variables of the input, for some constant $a > 0$. For every n, we define D_n a probability distribution on $\mathcal{B}^n$, where x_1 denotes the first $a \log n$ variables of the input and x_2 the others, $U(\mathcal{B}^N)$ is the uniform distribution on $\mathcal{B}^N$,

$$D_n[x_1, x_2] = \begin{cases} U(\mathcal{B}^{a \log n})[x_1] & \text{if } x_2 = \mathbf{0}^{n - a \log n} \\ 0 & \text{else.} \end{cases} \tag{A.35}$$

Finally, for every n, we define the learning problem $(g'|_n, D_n)$.

There exists a polynomial-time Turing machine that decides g'. Also the condition $|g'|_n|_{\mathcal{U}^c} \leq n^d$ will always be satisfied for all n large enough since the learning problem's function corresponds to a program/Turing-machine of fixed description-length.

Using the guarantee of Proposition 7, $m^{\epsilon,\delta}_{\mathcal{U}^c}(g'|_n, \mathcal{P}) \leq \frac{a_1}{\epsilon}(\log \frac{1}{\delta} + |g'|_n|_{\mathcal{U}^c} + a_2)$, we deduce that $m^{\epsilon,\delta}_{\mathcal{U}^c}(f, \mathcal{P})$ is upper-bounded by a constant independent of n.

For all n sufficiently large, all the Boolean circuits that approximates $g'|_n$ with accuracy at least 0.99 under D_n have sizes at least in $\tilde{\Omega}(n^{a\Omega(1)})$.

Thus, by Proposition 53, at least $\Omega(n^{a\Omega(1)})$ samples are necessary to solve the learning problem $(g'|_n, D_n)$ by $MDL^{\mathcal{C}}$ for all n sufficiently large.

Consequently, for any $\gamma > 0$ there exists a fixed and large enough, such that there exists $\epsilon \in (0, 1/2)$, $\delta \in (0, 1)$ and $c > 0$ satisfying $G^d_{\mathcal{U}^c}(\epsilon, \delta, n) \in \Omega(n^{1+\gamma})$. □

Appendix B. Description-length gains of Turing machines over circuits and neural networks

This section highlights the results focusing on description-length instead of PAC-learning gains. The proofs are similar to the proofs of the last section, and sometimes a reference to the proofs of the last section will be made.

The structure of the results' presentation is the same as for the study of PAC-learning gains.

Theorem 21. *There exists a constant $q \in \mathbb{R}^+$ such that for all $n \in \mathbb{N}^+$,*

$$\sup_{f \in H^n} \frac{|f|_{\mathcal{U}}}{|f|_{\mathcal{C}}} \leq q. \tag{B.1}$$

Proof. A technical detail is that, by Definition 37, for any n and function $f \in H^n$, $|f|_{\mathcal{C}} > 0$. So, the denominator is always non-zero.

The Proposition 49, affirms that there exists a constant K such that for any n and any function $f \in H^n$, the following holds $|f|_{\mathcal{U}} \leq |f|_{\mathcal{C}} + K$.

This fact, with the fact that the denominator is never null, implies

$$\sup_{f \in H^n} \frac{|f|_{\mathcal{U}}}{|f|_{\mathcal{C}}} \leq \frac{|f|_{\mathcal{C}} + K}{|f|_{\mathcal{C}}} \leq K + 1. \tag{B.2}$$

□

Theorem 22. *We have*

$$\sup_{f \in H^n} \frac{|f|_{\mathcal{C}}}{|f|_{\mathcal{U}}} \in \Omega(2^n). \tag{B.3}$$

Proof. Let be the Turing machine p^T, computing function p, and let $p|_n$ denote function p restricted to inputs of size n.

For input x of size n:

1. *Compute the input size n.*
2. *Enumerate all binary function in H^n in some pre-defined fixed lexicographic order. For each function, f:*
   (a) *Compute the minimal description-length necessary to compute the function f for a circuit according to interpreter $\mathcal{C}$:*

$$\begin{aligned} &\min_{h\in\mathcal{B}^*} && |h| \\ &\textit{subject to} && \mathcal{C}(h,y) = f(y) \quad \forall y \in \mathcal{B}^n. \end{aligned} \tag{B.4}$$

   (b) *If $|h| \geq 2^n$ then return $f(x)$.*

For any input-size n, there always exists a function that will satisfy the description-length condition that appears in step *(b)*. We prove it by a counting argument, there are $2^{(2^n)}$ functions in H^n and at most $2^{(2^n-1)}$ functions can be represented by a binary representation of length at most $2^n - 1$.

By construction and Definition 31 of the universal Turing machine, the computed function is computable by the Turing machine p^T and, thus, for any n, $|p|_n|_{\mathcal{U}} \leq \alpha$ for some constant α.

Also by construction, for any n, the computed function is only computed by circuits of description-length at least 2^n, $|p|_n|_{\mathcal{C}} \geq 2^n$. □

Theorem 23. *For all $c \in \mathbb{N}^+$, we have*

$$\sup_{f\in H^n} \frac{|f|_{\mathcal{C}}}{|f|_{\mathcal{U}^c}} \in O(n^c \log^2 n). \tag{B.5}$$

Proof. The proof is the same as the proof of Theorem 13 pruned of the PAC-learning related terms. □

Theorem 24. *For all $1 < c \in \mathbb{N}^+$, we have*

$$\sup_{f\in H^n} \frac{|f|_{\mathcal{C}}}{|f|_{\mathcal{U}^c}} \in \Omega(n \log n). \tag{B.6}$$

Proof. Consider the parity function $\oplus(x) = \sum_i x_i \mod 2$. For any n, let $\oplus|_n$ denote the function $\oplus$ restricted to inputs of size n.

The parity function can be computed by a Turing machine in linear time, and thus, for some constant α and all n sufficiently large, $|\oplus|_n|_{\mathcal{U}^c} \leq \alpha$ according to Definition 32.

Moreover, for any n, if a circuit compute $\oplus|_n$ then it must have at least n nodes to depend on all the input's variables. Then, by Definition 37 linking the circuit's size to its description-length, $|\oplus|_n|_{\mathcal{C}} \in \Omega(n \log n)$. □

Theorem 25. *If there exists $c, d \in \mathbb{N}^+$, and $\gamma > 0$ such that*

$$\left[\sup_{f \in H^n} \frac{|f|_{\mathcal{C}}}{|f|_{\mathcal{U}^c}} \quad \textit{such that} \quad |f|_{\mathcal{U}^c} \leq n^d\right] \notin O(n^{1+\gamma}) \tag{B.7}$$

then there exists a language in $\boldsymbol{P}$ not computable by a sequence of Boolean circuits whose sizes are in $O(n^{1+\tau})$ for some $\tau > 0$.

Proof. Take the proof of Theorem 15 beginning in Equation A.23. □

We note that the upper-bound n^d on the description-length $|f|_{\mathcal{U}^c}$ is not necessary for Theorem 25 to hold. Using Proposition 35, for any function relevant in the proof of the theorem, an upper-bound on the function description-length with $\mathcal{U}^c$ polynomial in the input-size holds. This fact can replace the bound in n^d in the proof of Theorem 15.

Theorem 26. *If for all $c \in \mathbb{N}^+$, and all $\gamma > 0$,*

$$\sup_{f \in H^n} \frac{|f|_{\mathcal{C}}}{|f|_{\mathcal{U}^c}} \in O(n^{1+\gamma}) \tag{B.8}$$

then $\boldsymbol{P} \neq \boldsymbol{NP}$.

Proposition 27. *Kannan [30]. For any nonnegative integer k, there exists a language $L \in \sum_2^p$ such that L is not computable by a sequence of circuits whose sizes are in $O(n^k)$, where n is the input-size.*

Proof of Theorem 26. By contraposition, suppose $\mathbf{P} = \mathbf{NP}$; then the polynomial hierarchy collapses and $\mathbf{P} = \mathbf{PH} = \sum_2^p$.

By Proposition 27, with $k = 2$, $\mathbf{P}$ has a language not computable by any circuit sequence whose sizes are in $O(n^2)$. Denote the function representing this language by f, and $f|_n$ its restriction to size n inputs.

Then, by Definition 37 of $\mathcal{C}$, a sequence of circuits that computes the function does not have their description-length in $O(n^2)$.

Moreover, by Definition 32 of $\mathcal{U}^c$, there exists constants c and α such that, for all n sufficiently large, $|f|_n|_{\mathcal{U}^c} \leq \alpha$ since there is a fixed Turing machine able to compute the function for all n. □

Theorem 28. *If there exist a language $g \in \boldsymbol{E}$ such that $g|_n$ can only be computed by circuits of sizes at least $2^{\epsilon n}$ for some $\epsilon > 0$; then for all $\gamma > 0$ there exists $c \in \mathbb{N}^+$ such that*

$$\sup_{f \in H^n} \frac{|f|_{\mathcal{C}}}{|f|_{\mathcal{U}^c}} \in \Omega(n^{1+\gamma}). \tag{B.9}$$

Proof. For any n pose f_n to be g applied to the first $a \log n$ variables of the input, for some $a > 0$. By construction $|f_n|_{\mathcal{C}} \in \Omega(n^{a\Omega(1)})$, and since there exists a polynomial-time Turing machine deciding (f_n), $|f_n|_{\mathcal{U}^c} \in O(1)$, for c large enough.

Fix a large enough to complete the proof. □

Appendix C. Interpreters

Appendix C.1. Universal Turing Machine

We restrict our Turing machines to binary-valued outputs in the whole work.

The definitions of this sub-section are given with the number of working-tape let as a variable in some cases. The results of this research are correct for any value of this parameter.

Definition 29. Turing machines. *We first define one-tape Turing machines before generalizing the definition to multi-tape Turing machines.*

A binary-valued one-tape Turing machine is defined by

- *Q a finite set of states;*
- *$A = \{0, 1, b\}$ the Turing machine's alphabet;*
- *$q_0 \in Q$ the initial state in which the Turing machine begins;*
- *$\{accept, reject\} = F \subset Q$ the set of the two final states which determine the output of the Turing machine: if the Turing machine stops in the accept state then the output is 1, else if it stops in the final state reject then the output is 0, else it is $\perp$;*
- *a mapping from $((Q \backslash F) \times A)$ to $(Q \times S)$ defining the transition of the Turing machine, where $S = A \cup \{L, R\}$ which correspond to either writing the element of A to the current head place or move the head to the L:left or R:right.*

The partial computable function implemented by the Turing machine is defined by setting the binary input on the unique tape in contiguous cells (the b symbol being assigned to the other cells); positioning the head on the first cell; set the Turing machine in the q_0 state; then to apply recursively the mapping of the Turing that determines the change of states, writing on the tape, and head movements; finally the output is obtained either when a final state in F is obtained, or else by $\perp$.

For $k_1, k_2 \in \mathbb{N}^+$, we define k_1-input-tape, k_2-working-tape Turing machines. These machines have k_1 binary-inputs that are placed on the k_1 inputs' tapes. There is one head by tape and these tapes are read-only. There are also k_2-working-tapes with one head by tape, they are blank at the start, and are read and write.

These machines are determined in a similar way to one-tape Turing machine: the elements Q, A, q_0, F, and S are defined in the same way. The mapping is adapted, the mapping goes from $((Q \backslash F) \times A^{k_1+k_2})$ to $(Q \times \{L, R\}^{k_1} \times S^{k_2})$, with the natural interpretation.

The partial computable function implemented by such Turing machines follows from a natural generalization of one-tape Turing machines.

Definition 30. Turing machines encoding $E(.)$, *[31]. Following Definition 29, any Turing machine, T, can be fully described by a set of states Q, the initial state q_0, and a mapping from $((Q \backslash F) \times A)$ to $(Q \times S)$.*

The mapping and T can be described by a list of quadruples $[(p_i, t_i, q_i, s_i)]_{i=1}^{r}$ where r is the number of rules and for all i, $p_i, q_i \in Q$, $t_i \in A$, $s_i \in S$. Each element can be identified with $s = \lceil \log(|Q| + 5) \rceil$ bits. Be $e(.) : Q \cup S \to \mathcal{B}^s$ this encoding. By convention this encoding will satisfy the following constraint, the states q_0, accept, and reject will be encoded to predefined arbitrary values (the three first elements in the Boolean lexicographic order of the output for example).

We define the encoding of T to be

$$E(T) = \underbrace{0 \dots 0}_{s} 1 \underbrace{0 \dots 0}_{r} 1 [e(p_i)e(t_i)e(s_i)e(q_i)]_{i=1}^{r}. \tag{C.1}$$

This encoding completely defines the Turing machine T.

The encoding is prefix-free: no encoding is the prefix of another.

To define an encoding for multiple-tape Turing machines, generalize the encoding to their mappings defined in Definition 29.

Definition 31. Universal Turing machine $\mathcal{U}$. *For any $k \in \mathbb{N}^+$, the interpreter $\mathcal{U} : \mathcal{B}^* \times \mathcal{B}^* \to \mathcal{B} \cup \{\perp\}$ computes $T(u, x)$ on input $([E(T), u], x)$, where E follows Definition 30 for two-input-tape and $k-$working-tape Turing machines. If the input has not a form that encodes a Turing machine then $\perp$ is the output.*

Note that the decomposition of the first argument in $E(T)$ and u is well defined since the encoding E is prefix-free.

Definition 32. Polynomial-time universal Turing machines $\mathcal{U}^c$.

For any $k, c \in \mathbb{N}^+$, we define an interpreter $\mathcal{U}^c : \mathcal{B}^* \times \mathcal{B}^* \to \mathcal{B} \cup \{\perp\}$. It is a 2-input-tape, 3-working-tape Turing machine.

On input $([E(T), u], x \in \mathcal{B}^*)$, for T a 2-input-tape k-working tape Turing machine, E the encoding in Definition 30, and $u \in \mathcal{B}^*$; the following operations are performed:

1. *On the third working tape, the interpreter computes the input-size, n, of the second input, x.*
2. *Still on the third work-tape, it computes n^c.*
3. *The interpreter computes in at most n^c steps that the form of the first input corresponds to the encoding of a Turing machine. If it does not correspond to a Turing machine or if the number of steps limit is reached, it outputs $\perp$.*
4. *Then, it computes a simulation of the behavior of the Turing machine T on input (u, x) with the two first work-tapes using the construction of [28], whose result is given in Proposition 33. Simultaneously, the interpreter computes the number of steps dedicated to the simulation on the third work-tape. (Note that it is well the number of steps dedicated to the simulation and not the number of simulated steps that are counted.)*
5. *In the computed simulation if a final state in F is reached then enters this state for the universal Turing machine. If the limit of computation for the simulation, n^c, is attained without entering a final state of F in the computed simulation then enter the state reject.*

For all these operations, the total number of steps for the first input, $[E(T), u]$, fixed can be made in βn^c, for some fixed $\beta > 0$.

Using Proposition 33, for any $\delta > 0$, any Turing machine with computational complexity in $O(n^{c-\gamma})$ can be simulated, for some $h \in \mathcal{B}^*$ and all n sufficiently large, by $\mathcal{U}^c(h, .)$.

Proposition 33. *Efficient universal Turing machines, [28], [26]. There exists a universal Turing machine which, for any Turing machine T, on inputs $E(T)$ and x computes $T(x)$.*

Furthermore, for some $\alpha > 0$, if the Turing machine T on input x stops in t steps then the universal Turing machine stops in $\alpha t \log t$.

Proposition 34. Hardwiring. *For any 2-input-tape Turing machine, T, and input $h \in \mathcal{B}^*$ there exists a Turing machine T^h such that T^h compute the function $T(h, .)$.*

Moreover, T^h has $\rho\,|h|$ rules, for some ρ independent of h; and T^h computes the function $T(h, .)$ in the same number of steps.

Proof. Let Q be the states and R be the set of rules of T that defines its mapping, construct the new states $Q \times \{1, \ldots, |h|\}$ and the new rules $R \times \{1, \ldots, |h|\}$. The new rules are made such that any operation of the Turing machine T on the first input-tape is translated into an equivalent change in the state of the Turing machine T^h. Allowing the simulation of the tape's head corresponding to input h in the states of T^h.

In our construction there are thus $R \cdot |h|$ rules in T^h, fix $\rho = R$ in the theorem statement. □

Proposition 35. *There exists a constant $\beta > 0$ such that for any function f in H_n, if $|f|_{\mathcal{U}^c} < +\infty$ then $|f|_{\mathcal{U}^c} \leq \beta n^c$.*

Proof. Any solution h of length larger than βn^c can be made smaller by cropping all binary symbols after index βn^c on the tape, since, by Definition 32, $\mathcal{U}^c$ cannot read them in βn^c steps. □

Appendix C.2. Boolean circuit

Definition 36. Boolean circuit*, [26]. A Boolean circuit C is a directed acyclic graph with $n \in \mathbb{N}$ potential sources and one sink. The source vertices have an associated input variable whose index is between 1 and n. The non-source vertices are called gates and have an associated logical operation OR, AND, or NOT ($\wedge$, $\vee$ or $\neg$) called label.*

The OR and AND vertices have two input edges, the NOT vertices have one input edge.

The number of vertices will be denoted $|C|$ and called the size of the circuit.

The output of the circuit on an input $x \in \mathcal{B}^n$ is the value associated to the sink vertice applying recursively the following assignment for each vertex v:

if v is a source corresponding to input variable i then its value is x_i; else v is a gate, apply the logical operator corresponding to its label on the input values (values from its parent vertices).

Definition 37. Boolean circuit interpreter, $\mathcal{C}$. *We define $\mathcal{C} : \mathcal{B}^* \times \mathcal{B}^* \to \mathcal{B} \cup \{\perp\}$ to compute circuit $C(x)$ on input (h, x) with h of the form*

$$\underbrace{0\ldots0}_{\lceil \log_2 n \rceil} 1[\textit{binary description of } n] \underbrace{0\ldots0}_{|C|} 1[\textit{label and inputs' vertices of vertice } i]_{i=1}^{|C|}, \tag{C.2}$$

where label and inputs' vertices of any vertice correspond to $2+\max\{2\lceil \log_2 |C| \rceil, \lceil \log_2 n \rceil\}$ bits; 2 bits to denote its logical label, and $2\lceil \log_2 |C| \rceil$ bits for the input vertices.

It outputs $\perp$ if h has not an acceptable form.

The description-length of h is thus of $2\lceil \log_2(n) \rceil + 2 + |C| \, (3+\max\{2\lceil \log_2 |C| \rceil), \lceil \log_2 n \rceil\}$ bits.

Which can be bounded by $9\,|C| \log |C|$ bits for circuits of sizes $|C| \geq n \geq 3$.

Proposition 38. *[32].*

Boolean circuits of size at most

$$\frac{2^n}{n}(1 + 3\frac{\log(n)}{n} + O(\frac{1}{n})) \tag{C.3}$$

compute all functions in H^n.

Appendix C.3. Artificial Neural Network

Similarly to Boolean circuits, we define an interpreter for Artificial Neural Networks (ANNs) and provide propositions linking the two definitions in terms of description-length.

Definition 39. Floating-point operators. *For any $d \in \mathbb{N}^+$, a floating-point operator is:*

- *a 0-ary/constant operator, which is a float-number —an element in $\mathcal{B}^d$;*
- *an unary operator from $\mathcal{B}^d$ to $\mathcal{B}^d$, such as negation $-(.)$, inverse $1/(.)$, or the exponential $\exp(.)$;*
- *a binary operator from $\mathcal{B}^d \times \mathcal{B}^d$ to $\mathcal{B}^d$, such as addition $(.+.)$, or product $(.\cdot.)$.*

Definition 40. Artificial Neural Network. *Given $d \in \mathbb{N}^+$ and a fixed predefined finite set of floating-point operators $\mathcal{O}$ on $\mathcal{B}^d$ with at least two 0-ary operators identified as 0^F and 1^F (thus both in $\mathcal{B}^d$).*

For any $n \in \mathbb{N}^+$, an ANN is a directed acyclic graph with one sink and at most n input-variable sources. The sources vertices have each an associated input variable whose index is between 1 and n. The non-source vertices have an associated operator taken from the set $\mathcal{O}$. If the operator is 0-ary then they have no parent vertice, if it is unary then they have one parent vertice, else if the operator is binary then they have two parent vertices.

We denote $|A|$ the number of vertices of ANN A.

To compute the output of the ANN on input $x \in \mathcal{B}^n$, the following operations are performed:

1. *for all the input-variable sources: the floating-point operator $0^F \in \mathcal{B}^d$ or $1^F \in \mathcal{B}^d$ is assigned accordingly to the associated input variable value in $\{0,1\} = \mathcal{B}$.*
2. *for all the other vertices assign the value in $\mathcal{B}^d$ corresponding to the associated floating-point operator in $\mathcal{O}$ and the values assigned to the potential parents.*
3. *when the output sink vertex has been assigned a value: if it is 0^F then output $0 \in \mathcal{B}$, if it is 1^F then output $1 \in \mathcal{B}$, else output $\perp$.*

Definition 41. Artificial Neural Network interpreter, $\mathcal{ANN}^{\mathcal{O}}$. *We define $\mathcal{ANN}^{\mathcal{O}} : \mathcal{B}^* \times \mathcal{B}^* \to \mathcal{B} \cup \{\perp\}$ the interpreter for ANN. On input (g, x) it computes the result of applying ANN A on x, $A(x)$, where g is of the form*

$$\underbrace{0\ldots0}_{\lceil \log_2 n \rceil} 1[\textit{binary description of } n] \underbrace{0\ldots0}_{|A|} 1[\textit{input variable/operator and inputs' vertices of vertice } i]_{i=1}^{|A|} \tag{C.4}$$

the operator in $\mathcal{O}$ and the parent(s) or the input variable will be described in $\lceil \log_2(|\mathcal{O}|+1) \rceil + \max\{2\lceil \log_2 |\mathcal{O}| \rceil), \lceil \log_2 n \rceil\}$ bits for each vertex.

If g does not have a correct form then $\perp$ is returned.

The length of g encoding an ANN A is $2\lceil \log_2 n \rceil + 2 + |A|\,(1 + \lceil \log_2(|\mathcal{O}| + 1) \rceil + \max\{2\lceil \log_2 |A| \rceil, \lceil \log_2 n \rceil\})$

Proposition 42. *For any fixed set of floats' operators $\mathcal{O}$ for which the AND, OR, and NOT Boolean functions can each be computed by an ANN using the operators in $\mathcal{O}$, there exists constant $\alpha, \beta > 0$ such that the following holds.*

For any $n \in \mathbb{N}^+$ and any $h \in \mathcal{B}^*$ there exists $g \in \mathcal{B}^*$ such that for all $x \in \mathcal{B}^n$

$$\mathcal{C}(h, x) = \mathcal{ANN}^{\mathcal{O}}(g, x) \tag{C.5}$$

and

$$|g| \leq \alpha \, |h| \, . \tag{C.6}$$

Conversely, for any $n \in \mathbb{N}^+$ and any $g \in \mathcal{B}^*$ there exists $h \in \mathcal{B}^*$ such that for all $x \in \mathcal{B}^n$

$$\mathcal{ANN}^{\mathcal{O}}(g, x) = \mathcal{C}(h, x) \tag{C.7}$$

and

$$|h| \leq \beta \, |g| \, . \tag{C.8}$$

Proof. For the first part, if h has not a correct form to represent a circuit then $\mathcal{C}(h, .)$ always output $\perp$ which can also be done by a g that do not represent an ANN.

Otherwise, there is a circuit, C, that corresponds to $\mathcal{C}(h, .)$, the logical operation of each node of this circuit can be simulated by a part of an ANN of fixed maximal size. A combination of these parts gives an ANN, A, which computes the same function as the Boolean circuit and whose size is at most a fixed multiple of the size of the circuit.

Let's pose $\gamma > 0$ the factor of the sizes, $|A| \leq \gamma \, |C|$. The description-lengths have the following relationship

$$\begin{aligned} & 2\lceil \log_2 n \rceil + 2 + |A| \, (1 + \lceil \log_2(|\mathcal{O}| + 1) \rceil + \max\{2\lceil \log_2 |A| \rceil, \lceil \log_2 n \rceil\}) \\ \leq \, & 2\lceil \log_2 n \rceil + 2 + \gamma \, |C| \, (1 + \lceil \log_2(|\mathcal{O}| + 1) \rceil + \max\{2\lceil \log_2 |C| \rceil + 2\lceil \log_2 \gamma \rceil, \lceil \log_2 n \rceil\}) \\ \leq \, & \alpha \, (2\lceil \log_2(n) \rceil + 2 + |C| \, (3 + \max\{2\lceil \log_2 |C| \rceil), \lceil \log_2 n \rceil\}) \, , \end{aligned} \tag{C.9}$$

with $\alpha = \max\{1, \gamma, (1 + \lceil \log_2(|\mathcal{O}| + 1) \rceil + 2\lceil \log_2 \gamma \rceil)/3\}$.

A similar argument holds for the second part by noting that any floating-point operator can be computed by a finite size Boolean circuit by Proposition 38. □

Appendix D. VC-dimension analysis and tightness of Proposition 7

Definition 43. Shatter. *Let be a set C.*

A set $A \subset 2^C$ shatters a set $B \subset C$ iff

$$\{a \cap B | \, a \in A\} = 2^{|B|}. \tag{D.1}$$

Definition 44. VC-Dimension*, [33]. The VC-dimension of a set $F \subset H^n$, $VC(F)$, is the size of the largest set $B \in \mathcal{B}^n$ such that F shatters B, where each binary-valued function in F is translated as a set in $\mathcal{B}^n$.*

Proposition 45. *VC-Dimension of Turing machines. There exists a constant $a > 0$ such that the following holds.*

For any n and D in $\mathbb{N}^+$, with D larger than some constant, we define the sets of functions $F^D = \{\mathcal{U}(h, .) \in H^n | \, h \in \mathcal{B}^* \ |h| \leq D\}$.

These sets satisfy $VC(F^D) \geq aD$.

Proof. Consider the two inputs Turing machine, $T(u, x)$, that output u_x if $x \leq |u|$ and 0 else, where the binary input string x is understood as the binary representation of a number.

From this Turing machine create for each D the set of functions $\{\mathcal{U}([E(T), u], .) | \, u \in \mathcal{B}^{\lfloor D/2 \rfloor}\}$, this set shatters the set of the aD strings corresponding to the aD first natural numbers in binary representation, for some $a > 0$. Moreover, if D is bigger than $2 \cdot |E(T)|$, the construction satisfies the upper-bound of D on the description-lengths. □

Proposition 46. *VC-Dimension of Boolean circuits. There exists constants a, b, N such that the following holds for all $n \geq N$.*

Be the sets of functions $F^D = \{\mathcal{C}(h, .) \in H^n | \, h \in \mathcal{B}^* \ |h| \leq D\}$ for some constant $D \in \mathbb{N}$, with $D \geq bn^{1.01}$.

Then $VC(F^D) \geq aD$.

Proof. We pose $b = b_1 b_2$, for two positive constants b_1 and b_2.

We have $1.01 \log n \leq \log D - \log b$. Select $q = \lfloor \log D - \log b_1 \rfloor$ an integer and $b_2 \geq e$ for $1.01 \log n \leq q$ to hold.

Consider the inputs $\{[z \underbrace{0 \dots 0}_{n-q}] \in \mathcal{B}^n | \, z \in \mathcal{B}^q\}$. This set is shattered by a set of circuits of size lower than $2\frac{2^q}{q}$ for n (and thus q) sufficiently large by Proposition 38.

For all n sufficiently large, we have $2\frac{2^q}{q} \geq 2\frac{n^{1.01}}{1.01 \log n} \geq n$.

Thus, by Definition 37 these circuits can be expressed with

$$18\frac{2^q}{q} \log \frac{2^q}{q} = \frac{18}{\log_2 e} 2^q (1 - \log_2^{-1}(e)\frac{\log q}{q} + \frac{\log 2}{\log_2(e)}\frac{1}{q}) \tag{D.2}$$

bits.

There exists a constant b_1 sufficiently large such that, for all n sufficiently large (forcing q to be sufficiently large), $q \leq \log D - \log b_1 \equiv 2^{\log b_1} 2^q \leq D$ implies that the description-length of the circuits, as bounded by Equation D.2, is smaller than D.

Finally, take $a = 2^{-\log b_1 - 1}$, we have $VC(F^D) \geq 2^q \geq aD$ since $q \geq \log D - \log b_1 - 1$. □

Proposition 47. *PAC-learning lower-bound, [34]. There exists a constant α such that the following holds for any $\epsilon \in (0, 1/2), \delta \in (0, 1)$, and any $n \in \mathbb{N}^+$.*

Consider the set of learning problems $(f, \mathcal{P})$ where $f \in F \subset H^n$ and $\mathcal{P} \in \Delta(\mathcal{B}^n)$ a probability measure.

For any learning algorithm, there exists a learning problem in the set such that a m-sample dataset of the considered learning problem with

$$m \geq \frac{\alpha}{\epsilon}\left[VC(F) + \log(\frac{1}{\delta})\right] \tag{D.3}$$

is necessary to get an (ϵ, δ)-PAC-learning performance.

Proposition 48. *Tightness of Proposition 7. There exists a constant α such that for any $\epsilon \in (0, 1/2), \delta \in (0, 1), n \in \mathbb{N}^+$ and any interpreter φ, associated learning algorithm MDL^φ, and bound $D \in \mathbb{N}^+$ on the description-length of an underlying function to learn; there exists a learning problem such that a m-sample learning dataset with*

$$m \geq \frac{\alpha}{\epsilon}\left[VC(\{f \in H^n |\ |f|_\varphi \leq D\}) + \log(\frac{1}{\delta})\right] \tag{D.4}$$

is necessary for MDL^φ to have an (ϵ, δ)-PAC-learning performance on the learning problem.

Proof. The statement is a direct consequence of Proposition 47. □

Appendix E. Kolmogorov complexity

The following proposition comes from Kolmogorov complexity theory, it is adapted to the context and notation of this paper. See [31] for a reference on the subject. The origins of the theorem can be found in [35, 36, 37, 38] and [39].

Proposition 49. *Invariance Theorem. For all interpreters φ there exists a constant K such that for all $n \in \mathbb{N}^+$ and all functions $f \in H^n$, the following holds $|f|_{\mathcal{U}} \leq |f|_{\varphi} + K$.*

Proof. Be f^{φ} the string of length $|f|_{\varphi}$ such that $\varphi(f^{\varphi}, .) = f(.)$.

Take $E(\varphi^T)$ the encoding of the Turing machine corresponding to the interpreter φ, the encoding follows Definition 30.

The function $\mathcal{U}([E(\varphi^T), f^{\varphi}], .)$ is equal to f by Definition 31, and the string $[E(\varphi^T), f^{\varphi}]$ has length $|f|_{\varphi} + K$ where $K = \left|E(\varphi^T)\right|$ is independent of f. □

Appendix F. Technical Propositions

Appendix F.1. Number of necessary samples for Boolean Circuits

Definition 50. *For any n, a* binary decision tree *is determined by*

- *a tree where all non-leaf nodes have exactly 3 neighbors: a first child, a second child, and one parent with the exception of one node which has no parent and is called the root;*
- *to each non-leaf node is associated an input-Boolean-variable;*
- *to each leaf is associated a Boolean value.*

The binary-valued function computed by the binary decision tree on an input $x \in \mathcal{B}^n$ is the result of the following computation:

1. *The current node is set to be the root.*
2. *If the current node is not a leaf, then if the associated input-Boolean-variable in x is 1 then set the current node as the first child, else set the second child as the current node. Else continue to the next step.*
3. *The current node is thus a leaf, return as output the Boolean value associated to the current node.*

Proposition 51. *Given m samples of a binary function there exists a binary decison tree with at most $2m - 1$ nodes consistent with the samples.*

Proof. In an optimal-size binary decision tree, there are at most m leaves, one for each sample. By induction, we can show that in any binary decision tree there is at most the number of leaves minus one internal node. □

Proposition 52. *There exists $a > 0$ such that if there exists a binary decision tree of size S computing a boolean function then there exists a Boolean circuit of size at most aS computing that function.*

Proof. Create $S - 1$ nodes, one for each node of the tree except the root. Each node has its value defined by the AND of the parent node and of either the input-Boolean-variable associated with the parent node if it is the first child or of its negation if the second child. To achieve this each time an input-variable is needed, create a node associated with the input-variable in the circuit. In the case of the second child, one supplementary node associated with the unary logical operator NOT is used to compute the negation of the corresponding input-variable.

For an input x, the values obtained at the nodes that correspond to the leaves —let's denote them b^t for leaf t– are all 0 except for the leaf at which the procedure described in Definition 50 terminates.

A network of logical operators can aggregate the output associated with the leaves —let's denote them o^t for leaf t— and output the answer associated with the only leaf to which 1 has been associated.

To do so consider the following two Boolean variables of $x_1, x_2 \in \mathcal{B}^n$, with $x_1^0, x_2^0 = 0$, and with the following update for leaf number $t \in \mathbb{N}$ with associated output Boolean value o_t and Boolean node value at runtime b_t,

$$\begin{bmatrix} x_1^{t+1} \\ x_2^{t+1} \end{bmatrix} = \begin{bmatrix} \text{if } x_2^t \text{ then } x_1^t; \text{else } o^t \\ \text{if } x_2^t \text{ then } x_2^t; \text{else } b^t \end{bmatrix}. \tag{F.1}$$

This system iterated for all the leaves computes the output of the binary decision tree in x_1, and this iteration can be computed with a number of nodes that scale linearly with the number of leaves.

The final circuit size is in $O(S)$. □

Proposition 53. *There exists a constant $b > 0$ such that given m samples of a binary function there exists a Boolean circuit of size at most bm consistent with the samples.*

Proof. Merge the last two results, Propositions 51 and 52, to first produce a binary decision tree, then to convert it into a Boolean circuit of suitable size. □

Appendix F.2. A combinatoral Proposition

An useful Definition and Proposition, it comes from [40].

Definition 54. *The* binary entropy function H *is defined by*

$$H(x) = \begin{cases} 0 & \text{if } x = 0; \\ -x\log_2 x - (1-x)\log_2(1-x),\ 0 < x \leq \frac{1}{2} & \text{else if } 0 < x \leq 1/2. \end{cases} \tag{F.2}$$

Proposition 55. *Let* $0 \leq \epsilon \leq \frac{1}{2}$ *and* $M \in \mathbb{N}^+$*, we have*

$$\sum_{0 \leq i \leq \lfloor \epsilon M \rfloor} \binom{M}{i} \leq 2^{MH(\epsilon)}. \tag{F.3}$$