UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# MODEL INTERPRETABILITY THROUGH THE LENS OF COMPUTATIONAL COMPLEXITY

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

BERNARDO ANÍBAL SUBERCASEAUX ROA

Advisor:
PABLO BARCELÓ BAEZA
Advisor 2:
JORGE PÉREZ ROJAS

Committee:

This work has been partially funded by Instituto Milenio Fundamento de los Datos, and it is based on joint work with both advisors and Mikaël Monet.

SANTIAGO - CHILE
JULY 2020

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN
POR: BERNARDO ANÍBAL SUBERCASEAUX ROA
FECHA: JULY 2020
PROF. GUÍA: PABLO BARCELÓ BAEZA JORGE PÉREZ ROJAS

# MODEL INTERPRETABILITY THROUGH THE LENS OF COMPUTATIONAL COMPLEXITY

In spite of several claims stating that some models are more interpretable than others – e.g., "linear models are more interpretable than deep neural networks" – we still lack a principled notion of interpretability to formally compare among different classes of models. We make a step towards such a notion by studying whether folklore interpretability claims have a correlate in terms of computational complexity theory. We focus on *local post-hoc explainability queries* that, intuitively, attempt to answer why individual inputs are classified in a certain way by a given model. In a nutshell, we say that a class $\mathcal{C}_1$ of models is *more interpretable* than another class $\mathcal{C}_2$, if the computational complexity of answering post-hoc queries for models in $\mathcal{C}_2$ is higher than for those in $\mathcal{C}_1$. We prove that this notion provides a good theoretical counterpart to current beliefs on the interpretability of models; in particular, we show that under our definition and assuming standard complexity-theoretical assumptions (such as P $\neq$ NP), both linear and tree-based models are strictly more interpretable than neural networks. Our complexity analysis, however, does not provide a clear-cut difference between linear and tree-based models, as we obtain different results depending on the particular post-hoc explanations considered. Finally, by applying a finer complexity analysis based on parameterized complexity, we are able to prove a theoretical result suggesting that shallow neural networks are more interpretable than deeper ones.

# Contents

# Introduction

Machine learning, and statistical models in general, empower many of the technologies we daily use. Assistants in our phones, films recommendations we get, and search engines that make science easily accesible over the internet; they all rely to a certain degree on these methods. Nonetheless, different forms of artificial intelligence are used as well in high-stake domains, like the judicial system, healthcare, bank loans, self-driving cars, etc. While annoying, a mistake in the voice recognition component of our favorite phone assistant is usually not a big deal. However, a single wrongful conviction made with the assistance of a machine learning model is more than enough to deeply worry about the role of artificial intelligence in our society. Unfortunately, such cases have already been reported [6, 23, 48]. It is thus essential to produce models that we humans can manage effectively, and whose decisions we can understand and trust. This is precisely the purpose of *Explainable Artificial Intelligence (XAI)* or *interpretability* [29].

It is not easy to define the precise meaning of concepts like "understand" or "interpret", which are crucial for the field, and despite substancial work [14, 23, 36] the interpretability community does not seem to have achieved consensual definitions [6]. But even without a precise definition, several claims [6, 29, 41] state that some models are more interpretable than others – e.g., "linear models are more interpretable than deep neural networks" – making up for a folklore wisdom of the field. However, we still lack a principled notion of interpretability that allows us to formally compare among different classes of models.

A principled understanding of model interpretability is particularly important when discussing about systems that are suitable for high-stake decisions, like justice or healthcare. Not only a part of the research community [48] claims that models that we cannot easily interpret should not be used for such delicate tasks; this idea has even become part of legislative discussions and regulations. A widely debated example is that of the European Union's General Data Protection Regulation, and in particular, Article 22, which is argued to guarantee citizens a "right to an explanation" [26].

In order to study and compare the interpretability of different classes of models, we focus on *local post-hoc explanations* [6, 28, 36, 39, 40]. The term "local" refers to explaining the verdict of the system for a particular input [28, 40]. The term "post-hoc" refers to interpreting the system after it has been trained and without elucidating its internal mechanisms [36, 39]. In particular, we study the computational complexity of a set of *explainability queries*, that is, well-defined computational problems whose answer serves as a post-hoc explanation for a given decision taken by a system. As a result, we find that answering these explainability queries is inherently harder for certain classes of models than it is for others. We largely develop and extend this idea, providing a frame-

work that allows to formally prove statements of the form "linear models are more interpretable than deep neural networks", under a precise mathematical definition of *more interpretable than*.

**The main contribution of this thesis is thus the proposal of a formal and objective framework to measure and compare the theoretical interpretability of different classes of Machine Learning models. Its main goal is to take a step towards a rigorous science of interpretability**.

As for the organization, Chapter 2 presents our main contribution: a framework based on computational complexity for measuring and comparing the interpretability of different classes of models. In a nutshell, we say that a class $\mathcal{C}_1$ of models is *more interpretable* than another class $\mathcal{C}_2$, if the computational complexity of answering post-hoc explainability queries for models in $\mathcal{C}_2$ is higher than for those in $\mathcal{C}_1$. We prove, throughout Chapter 3, that this notion provides a good theoretical counterpart for the folklore wisdom on the interpretability of models [29, 36, 41]. For this, we focus on linear models, FBDDs (a superclass of decision trees) and neural networks (referred to as Multilayer Perceptrons or MLPs).

Then, in Chapter 4, we use the more refined notion of *parameterized complexity* [15, 21] to compare the interpretability of MLPs according to their number of layers. Using this theory, we are able to prove that there are explainability queries that are more difficult to solve for deeper MLPs compared to shallow ones, thus giving theoretical evidence that shallow MLPs are more interpretable.

In Chapter 5 we present a correlate between the presented framework and the complexity of detecting bias. Under a precise definition, we prove that detecting bias in MLPs is harder than it is for linear models and FBBDs.

We finally dedicate Chapter 6 to discuss achievements and limitations of our work, together with a brief mention to similar efforts in the literature, and future directions of work.

But before diving into deep waters, we take 3 preliminary sections in Chapter 1 to get the reader started and well motivated. Section 1.1 walks the reader through an example that will help us unpack the different components of the core issue at hand, and provide intuition on the framework that will be presented throughout this document. Section 1.2 aims to put our work in context by presenting a brief a discussion on the problem of defining *interpretability*, and the role of *explanations* in the field. Finally, Section 1.3 presents the required background in complexity theory.

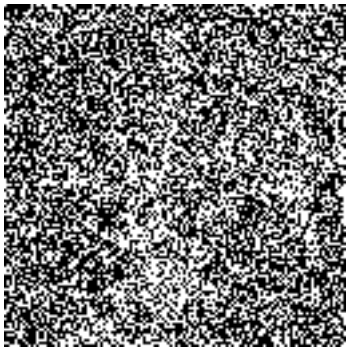
# Chapter 1

# Preliminaries on Interpretablity and Complexity

## 1.1 Cat Recognition, a bottom-up example to get started

Most of the problems that traditional algorithms were designed to solve can be easily formalized in mathematical language, like finding the employee with the largest salary in a company, or detecting if every router in a network is reachable from a given source. However, a wide variety of the problems we encounter in a daily basis, like distinguishing familiar faces, seem to elude such formalisms. Consider the problem, stated in Figure 1.1 of looking at a black-and-white picture of 128x128 pixels and deciding whether it portrays a cat or not. This problem can be interpreted as computing a function $f\colon \{0,1\}^{128x128} \to \{0,1\}$, that even children seem to be effective at computing, and yet for which we lack a simple mathematical description. What we do tend to have is data; millions of pictures of cats, and millions of pictures of things that are not cats. Deducing such a function $f$ from this data, in such a way that it can take good decisions on new pictures (that is, not present in the data it was generated with) is the whole goal of *Supervised Learning*. However, this thesis is not about the process of how one gets to learn such functions, but rather about *understanding* them once they have already been learned.

(a) A 128x128 B&W picture of a cat



(b) A 128x128 B&W picture that does not portray a cat

Figure 1.1: We can easily recognize a cat on the left image. It is harder, however, to explain the exact characteristics of the image that trigger such classification.

But before we get to discuss about understanding functions, it is convenient to take a look at how to represent them. Listing every single 128x128 picture that a function classifies as a cat is prohibitively expensive, as some functions would require the inconceivable amount of $2^{128x128}$ bits to be represented. By a similar argument, if one wished to learn an arbitrary functions this way, and by looking at a labeled picture one was able to discard half of the possible functions, there would be some functions requiring $2^{128x128}$ pictures to be learned. The natural consequence of this limitation is that instead of trying to learn or represent arbitrary functions, one first fixes a particular class $\mathcal{F}$ of syntactically restricted functions, that can then be learned and represented efficiently. For example, consider the class of functions $f_t$ that upon a picture $X$ decide if $X$ portrays a cat in the following way:

$$f_t(X) = \begin{cases} 1 & \text{if } X \text{ has at least } t \text{ black pixels} \\ 0 & \text{otherwise} \end{cases}$$

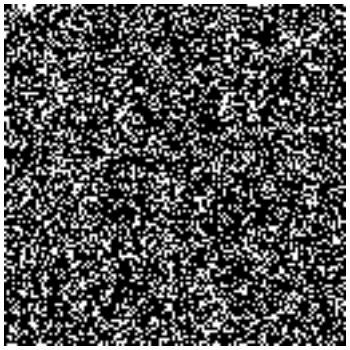
This class of functions can be easily represented, and the value of $t$ that minimizes the error over a set of pictures can be efficiently learned. These syntactically restricted classes of functions are referred to as classes of *models*. For example, we can say that the functions $f_t$ described above are *threshold models*. A model is thus an instance of a class of models.

Consider the model $f_{10000}$. One can check (hopefully with the help of a computer) that the picture in Figure 1.1a is correctly classified as a cat by this model, while that of Figure 1.1b is correctly classified as not-a-cat. Both humans and $f_{10000}$ agree on the fact Figure 1.1a portrays a cat, however, while it is hard for a human to describe the precise characteristics that let them know the picture portrays a cat, it is trivial to answer why $f_{10000}$ claims it. Indeed, $f_{10000}$ claims there is a cat on the picture because it contains $12427$ black pixels, while its counterpart, Figure 1.1b contains only $8969$ and is thus classified as not-a-cat. By listing the coordinates of the first $10000$ black pixels (in lexicographic order, for instance) of Figure 1.1a, we have described a precise set of characteristics that *explain* its classification. We can explain as well that Figure 1.1b is classified as not-a-cat by listing $128 \cdot 128 - 10000 + 1 = 6385$ white pixels in it.

Now that we *understand* how $f_{10000}$ works, we have clear reasons to *not trust it.* As shown in Figure 1.2, the number of black pixels alone is not a good criterion for telling cats. Once we understand how $f_{10000}$ works, it becomes evident that it is not a good classifier, but it could be far from obvious if we had only presented a maliciously crafted dataset over which $f_{10000}$ could perform very well, and a highly obfuscated code for it.

Imagine that this is the case, we are given a model as a file with thousands of lines of code, that is completely unreadable, and a dataset for which it performs extremely well. If our life depended on the performance of this model over new examples, we would be in trouble; we do not understand how it works. So how can we increase our trust in the model? Well, we do know that the model has taken right decisions, but what we lack is a principled understanding of the *reasons behind* such decisions. In the case of $f_{10000}$, as mentioned before, listing any set of $10000$ black pixels is enough to explain its decision over the picture of a cat in Figure 1.1a. We present a possible *explanation* for this decision in Figure 1.3a, where exactly $10000$ black pixels of the original picture are displayed. This explanation is a clear argument for not trusting $f_{10000}$, and if we obtained a similar one for the obfuscated classifier we received, we would know that it is not trustworthy either. On the other hand, if we knew that a classifier has decided based on the set

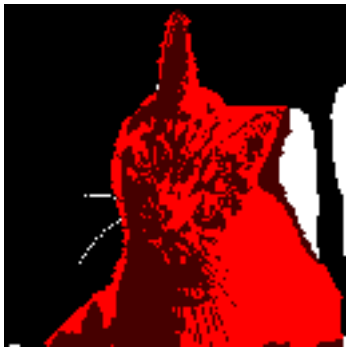
(a) An adversarial 128x128 B&W picture of a cat misclassified by $f_{10000}$



(b) A 128x128 B&W picture that does not portray a cat, and yet is accepted by $f_{10000}$

Figure 1.2: Examples of mistakes that $f_{10000}$ makes as a cat-classifier.

(a) An explanation for why $f_{10000}$ classified Figure 1.1a as a cat.



(b) A more convincing explanation for classifying a cat

Figure 1.3: Minimum sufficient reasons for cat recognition. Pixels that belong to the minimum sufficient reasons have been colored with red (dark red if they were previously black, light red if they were previously white)

of pixels that we associate with the cat, our trust would increase; we now know that the model is at least looking at the right thing.

This kind of explanation, that we will define with precision in Chapter 2, is called a *Minimum Sufficient Reason* (MSR), and intuitively represents the minimum portion of an input that allows a model to take a decision, regardless of the rest of the input.

It is algorithmically easy to compute an MSR for $f_{10000}$. Indeed, if for a picture $X$ we have $f_{10000}(X) = 1$, then iterating over the pixels of $X$ and listing the first $10000$ black pixels we find is enough. On the other hand, if $f_{10000}(X) = 0$, it is enough to list the first $128 \cdot 128 - 10000 + 1$ white pixels we find, as even if all the rest were black, one could not achieve the $10000$ required to be classified as a cat. As it will proved in Chapter 3, finding an MSR is algorithmically easy for a class of models that includes $f_{10000}$ as a very particular case; linear models, as for example used in a logistic regression. We will prove as well that finding an MSR is computationally hard for more sophisticated models like *Multilayer Perceptrons* (MLPs).

We have showed, with a simple example, that explanations like an MSR can give insight about

the way a model works, and with this, increase our trust and understanding. This implies, as it will be developed in Chapter 2, that in a precise sense it is easier to interpret and understand decisions taken by models like $f_{10000}$ than it is to do so over MLPs, as it is computationally easier to obtain explanations for the former than it is for the latter.

We can now have a better understanding of the main contribution of this work: we develop a framework that allows to formally compare the interpretability of different classes of models by studying the computational complexity of obtaining explanations for a decision a model has taken.

In the next section we visit the main challenges of interpretability and the related work, as this will help us to set the context required for a deeper understanding of our particular inquiry.

## 1.2 But what is interpretability after all? A brief literature discussion

We have used the word *interpretability* a few times thus far, but without giving it a precise meaning. This section discusses potential definitions, challenges and relevant research in the area.

Let us start with a simple proposal that captures well the intuition behind the concept. We will then unpack its different components, and use it as well as an excuse to go through some of the discussions that are present in the literature.

**Proposed Definition 1** (Miller [38], Biran & Cotton [9], Doshi-Velez et al.[14]) Interpretability is the degree to which a human can understand the cause of a decision.

First, it is relevant to notice that this definition directly involves humans in the loop, and thus, accounts for an inherently social nature to the concept. The relationship between explainable AI and social sciences is thoroughly developed in the work of Miller [38], and a practical interpretability setup with humans in the loop is explored for example in the work of Lage et al. [35]. This idea is directly contrasted by some authors [31], stating that human bias towards simple explanations can be dangerous, and one should be wary as well of incentivizing *persuasive* systems rather than transparent systems.

It is the last part of this definition however that is most controversial. Even if one could *understand the cause* of every decision of a decision-taking-agent, that does not necessary elucidates the means and inner mechanisms through which the agent has arrived to it. As a consequence, it is necessary to distinguish between two different forms in which decisions can be easily understood. On the one hand, it could be that the mechanisms by which an agent takes decisions are self-explanatory, evident, or more generally *transparent*. But it could be as well that, while the mechanisms are opaque, an agent could present convincing explanations for its decisions. Taking on an example by Miller [38], imagine person $A$ closes the window of their room, and person $B$ asks "Why did you do that?" It seems a priori that "Because I was cold." could be a convincing explanation for the decision taken. However, we completely lack a principled understanding of person $A$'s thought process. This could be important, say, if person $B$ was not convinced about

why did $A$ not choose to put on a coat instead. While one could argue that a more detailed explanation from $A$ could clarify this point, there is still a fundamental lack. This distinction, between an agent's decision process being transparent, and the ability of its decisions being justified, is consistently made throughout the literature [6, 36, 38]. The latter is referred to as *post-hoc explainability* or *post-hoc interpretability* [6, 36, 38].

Summarizing, Proposed Definition 1 identifies degrees of interpretability either with an inherent degree of transparency, or with the degree to which it is easy to provide convincing explanations. The exact definition presented by Doshi-Velez et al. aligns more with the latter, as it says that "interpretability is the ability to explain or to present in understandable terms to a human" [14].

### 1.2.1 Transparency

While the concept of transparency is not easy to define either, certain key aspects are identified in the the work of Lipton [36] describing properties of transparent models. We present here a summary inspired by the survey of Arrieta el al. [6].

- *Simulatability*: how easy it is for humans to manually simulate the model. This encompasses the size of models, and their associated computational cost.
- *Decomposability*: how easy it is to explain the different parts (parameters, hyper-parameters, input, etc.) of a model independently.
- *Algorithmic Transparency*: how easy it is for humans to understand the process followed by the model to produce any given output from its input data. This includes the mathematical guarantees (e.g. convergence) that we have around models.

A relevant distinction to our work is that simulatability can be understood as a predicate one evaluates over each individual case, which would imply for example that no model with an output size big enough could ever be transparent, even if the models processing was simply to add 1 to every coordinate in the input. Or, on the other hand, as a property of the model stating that, for a fixed input/output size, the simulation is easier to perform than it is for other models. This is exactly the approach we take in our work.

### 1.2.2 Explanations or Post-hoc interpretability

As opposed to an idea of intrinsic transparency, one could gain understanding of a system through explanations that enhance its interpretability. This can even be done in a way that treats systems as black-boxes, without any access to their internal mechanisms, as in the famous case of LIME [46]. A relevant distinction is that of *local* vs. *global* explanations [40, 28]. While local explanations attempt to clarify the particular reasons why a certain decision has been taken, global explanations attempt to give insight into the decision-taking process as a whole, in a way that would allow the explainee to understand further decisions. LIME (Local Interpretable Model-Agnostic Explanations) [46] is an example of local explanations as it approximates a given model on a given input by a linear model in the neighborhood of the input. This means that the explanation is faithful to the model only in a local region of the input space. On the other hand, many have approached interpretability for Deep Learning by creating models of supposedly more in-

herent transparency, like Decision Trees or Rules [6, 23, 30, 57]. The idea is to build a proxy model that approximates a given Deep Neural Network as faithfully as possible. This is an example of a global explanation method, as the model that is used as a proxy attempts to approximate the original model in the whole input space.

There is substancial literature from the social sciences on what explanations are, their relation with causality and their relationship with humans. The subject has been throughly studied since Hume, and great reviews on different definitions and components of explanations, and what constitutes a *good explanation* can be found in the work of Miller [38], or by following the leads in that of Gilpin et al. [23].

### 1.2.3 Interpretability vs. Explainability

A core problem associated with this definition is that of the difference between *interpretability* and *explainability*. Both terms are used substantially throughout the literature, and their respective popularity has changed over time. While that of interpretability was significantly more used in the past, the concept of explainability has gained more traction since 2019 [6]. But, do they mean the same or are they truly different? According to Gilpin et al. [23] interpretability and explainability have been used interchangeably in the literature, and yet they present subtle but important differences. Gilpin et al. [23] claim that explainability goes one step further than interpretability – "Explainable models are interpretable by default, but the reverse is not always true". On the other hand, Rudin [48] suggests that, especially in high-stakes decisions, one should aim for interpretability instead of explainability. The argument Rudin makes include potential issues with explanations, as a lack of faithfulness with respect to the model. For example, one could train a complex model both to make predictions and write convincing explanations in natural language, but how could we guarantee that the explanations actually account for the computation is being done? It is also argued in the work of Rudin [48] that explanations often do not make sense or are precise enough to fully explain phenomenons. Up to the present date, there does not seem to be a universally accepted clear cut between interpretability and explainability.

### 1.2.4 Desiderata

Perhaps disappointingly, there is still no consensus on the definition of interpretability [14, 36]. However, given that significant word [14, 36] has been done to this respect, it is to be seen in the next years whether the community will agree upon a definition. In the meantime, there seems to be an implicit agreement on what the *goals* of interpretability are [36, 6]. While models tend to be evaluated on predictive performance during testing, once deployed stakeholders desire things that usually cannot be easily captured by mathematical functions. Taking Lipton's example, a model used for hiring decisions should take into account not only predictive performance, but also legality and ethics.

More precisely, several authors have described a *desiderata* of interpretability [6], meaning a certain set of characteristics that the models we deem interpretable should have. We refer in particular to that of Lipton [36].

- *Trust*: Often portrayed as one of the main goals of interpretability [6], it has to do with the confidence a model inspires on humans about its performance, fairness and behavior on new and more diverse examples.

- *Causality*: Refers to the desire of obtaining and understanding causality relations through the interpretation of models. It is covered extensively by Miller [38], and the foundations of the underlying mathematical theory date back to Judea Pearl [43].

- *Transferability*: Refers to the ability of a model to transfer skills to unfamiliar situations, or generalize. In particular, it has to do with models retaining predictive performance once deployed.

- *Informativeness*: For models whose purpose is to help human decision-makers, one would desire to be able to not get only answers but the most information possible from the models.

- *Fairness*: It is often argued that interpretability is required to asses whether automatic decisions confirm to ethical standards. The desire is that of systems whose decisions are *contestable*, and that can provide clear reasoning based on falsifiable propositions.

Finally, for a detailed survey on Explainable AI and practical methods, we direct the reader to the survey of Barredo Arrieta et al. [6], and for discussions on the principles of explanations and interpretability, we refer to the work of Lipton [36], Doshi-Velez et al. [14], Rudin [48] and Miller [38].

## 1.3 Background in Complexity Theory

A general background in discrete mathematics (as presented for example in the appendix of Arora and Barak [5]) is assumed, including the definition of both deterministic and non-deterministic Turing Machines.

Let $\Sigma$ be a finite alphabet, and $\Sigma^*$ the set of all finite strings whose symbols all belong to $\Sigma$. Any subset $L$ of $\Sigma^*$ is said to be a *language* or, equivalently, a *problem*. Given a Deterministic Turing Machine (DTM) $M$, defined over the alphabet $\Sigma$, we can identify the set $L = L(M)$ of strings accepted by $M$, and analogously, we can do so for a Non-deterministic Turing Machine (NTM) by considering the set of strings for which there is a computation path leading to an accepting state. For any Turing Machine, deterministic or not, we can define a function $\texttt{time}_M : \Sigma^* \to \mathbb{N}$ such that $\texttt{time}_M(w)$ describes the number of steps in the computation of string $w$ by the machine $M$. This naturally defines a function $t_M : \mathbb{N} \to \mathbb{N}$ for any Turing Machine $M$, such that $t_M(n) = \max_{w,|w|=n} \texttt{time}_M(w)$. This definitions allow us to define the following classes:

**Definition 1.1** The class PTIME is the class of all problems that can be solved in polynomial time by a DTM. More precisely:

$$\text{PTIME} = \{L \mid \text{there exists a DTM } M \text{ such that } L = L(M) \text{ and } t_M(n) \in n^{O(1)}\}$$

Note that we have kept the alphabet $\Sigma$ implicit, and will do so repeatedly.[1]

[1] It is easy to show that the classes we define in this section are essentially equal for any fixed alphabet of size at least 2. In general, very small alphabets are enough to encode most natural problems. Consider, for example, that the alphabet $\Sigma = \{\exists, \forall, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, \cdot, =, \hat{}, /\}$ is powerful enough to encode most arithmetic problems.

**Definition 1.2** The class NP is the class of all problems that can be solved in polynomial time by a NTM. More precisely:

$$\text{NP} = \{L \mid \text{there exists a NTM } M \text{ such that } L = L(M) \text{ and } t_M(n) \in n^{O(1)}\}$$

As DTMs are a particular case of NTMs, we trivially have that PTIME $\subseteq NP$. On the other hand, perhaps the most important open problem in theoretical computer science is whether PTIME = NP. It is widely believed that this is not the case, and many of the results presented in this work are conditioned by the hypothesis PTIME $\neq$ NP.

We now present an equivalent definition for the class NP, that will be amply used.

**Definition 1.3** (Verifier based)[2] The class NP is the class of all problems that can be verified in polynomial time by a DTM. More precisely, a language $L$ belongs to NP if and only if there is a DTM $M$ such that for every string $w$ in $L$ there exists a string $c$, that we call a *certificate* for $w$, such that the string $\langle w, c\rangle$ is accepted by $M$, and $\texttt{time}_M(\langle w, c\rangle) \in |w|^{O(1)}$.

For any class $\mathcal{C}$, we define co$-\mathcal{C}$ as the set of languages whose complement belongs to $\mathcal{C}$. A particular case we will refer to is that of co-NP, the class of problems whose complement is in NP. It is not known whether NP $\subseteq$ co-NP or co-NP $\subseteq$ NP but it is widely believed that neither inclusion holds. We now define a new class that closely relates to both NP and co-NP.

**Definition 1.4** We say a language $L$ belongs to DP if $L = L_1 \setminus L_2$ for $L_1, L_2 \in$ NP, or equivalently $L = L_1 \cap L_2'$ for $L_1 \in$ NP, $L_2' \in$ co-NP.

One can define as well classes of machine that depend on *oracles*. Intuitively, an oracle can be thought of as a magical entity that equips Turing Machines and allows them to outsource computational work that will be done for them in constant time. For example, a DTM equipped with an oracle `DotProduct` for the problem of deciding whether the dot product between two vectors is a vector of zeros, can verify that the product of two $n \times n$ matrices is the null matrix in time $O(n^2)$, by performing $n^2$ calls to `DotProduct`. For a formal definitions of oracles we refer the reader to Arora and Barak [5]. We note $M^A$ the machine equivalent to equipping machine $M$ with an oracle for problem $A$. Based on this notion of oracle, we can define new complexity classes. Let $\mathcal{C}$ be a complexity class defined through a time restriction on Turing Machines, and $A$ a problem. We say $\mathcal{C}^A$ is the class of problems that can be accepted by a machine complying with the time restriction of $\mathcal{C}$ but having an oracle to problem $A$. For a complexity class $\mathcal{D}$, we define $\mathcal{C}^{\mathcal{D}} = \cup_{A \in \mathcal{D}} \mathcal{C}^A$. We can now define an important class based on this notion of oracle.

**Definition 1.5** The class $\Sigma_2^p$ is the class of problems that can be solved by a NTM in polynomial time with access to an oracle to any problem in the class NP. More precisely, $\Sigma_2^p = \text{NP}^{\text{NP}} = \text{NP}^{\text{co-NP}}$.

Note that we trivially have that NP $\subseteq \Sigma_2^p$, and this inclusion is widely believed to be proper.

[2] The equivalence between this and the previous definition is a trivial theorem.

In other words, it is widely believe that $\Sigma_2^p$ contains harder problems than those in NP.

We say a function $f\colon \Sigma^* \to \Sigma^*$ is a polynomial time reduction[3] from problem $A$ to problem $B$ if for any string $w \in \Sigma^*$ it holds that $w \in A$ if and only if $f(w) \in B$, and $f$ can be computed in polynomial time by a DTM[4]. We say $A \leq_p B$ if there is a polynomial time reduction from $A$ to $B$, as the fact that we can solve an instance $w$ of $A$ by first computing $f(w)$ and then solve an instance of $B$ means intuitively that problem $B$ is harder to solve than problem $A$.

**Proposition 1.6** We say a class $\mathcal{C}$ is closed under polynomial time reductions if for any problem $B \in \mathcal{C}$ and any problem $A$ it holds that $A \leq_p B$ implies $A \in \mathcal{C}$. The classes PTIME, NP, co-NP, DP and $\Sigma_2^p$ are all closed under polynomial time reductions.

For each class $\mathcal{C}$, we say a problem $B$ is $\mathcal{C}$-hard if for every problem $A \in \mathcal{C}$ it holds that $A \leq_p B$. When a problem is $\mathcal{C}$-hard, and also belongs to $\mathcal{C}$, we say it is $\mathcal{C}$-complete.

We now focus on function problems, that is the computational problem of computing a function $f\colon \Sigma^* \to \mathbb{N}$. In order to define the most relevant class, #P, we define first a for an NTM $M$ the function $\texttt{accept}_M\colon \Sigma^* \to \mathbb{N}$ such that $\texttt{accept}_M(w)$ describes the number of accepting paths when machine $M$ is run with the string $w$. We say a function $f\colon \Sigma^* \to \mathbb{N}$ belongs to #P if there is an NTM $M$ such that for every string $w \in \Sigma^*$ it holds that $f(w) = \texttt{accept}_M(w)$. Contrary to the other classes defined so far, #P is closed under a different kind of polynomial time reductions, usually called *Turing reductions*. We say a problem $A \leq_p^T B$ if and only if $A \in P^B$. Finally, we distinguish a particular kind of reductions for #P, which is that of *parsimonious reductions*. Given function problems $A$ and $B$, we say there is a parsimonious reduction from $A$ to $B$, denoted as $A \leq_p^{par} B$ if there is a function $f\colon \Sigma^* \to \Sigma^*$ computable in polynomial time such that $A(w) = B(f(w))$ for every string $w \in \Sigma^*$. Note immediately that every parsimonious reduction is a Turing reduction as well. Parsimonious reductions are of particular interest because it is widely believe that if $\#SAT$, the problem of counting the satisfying assignments of a boolean formula, can be parsimoniously reduced to a problem $A$, then there are no efficient approximation algorithms for $A$ [5]. The notion of #P-hardness and #P-completeness can be defined analogously as for the previous classes, but by using Turing reductions instead.

We remark that although #P and $\Sigma_2^p$ refer to different kinds of problems (function problems and decision problems respectively), there is a precise sense in which the class #P is harder than $\Sigma_2^p$ under complexity assumptions; it is easy to see that if one could solve a #P-complete problem in polynomial time, then one could solve every $\Sigma_2^p$ problem in polynomial time as well. However, the opposite direction is believed to fail [5].

Finally, we note that all big-Oh results are given assuming a standard RAM model [5]. While this impacts precise runtimes, it makes no difference for the complexity classes defined above, as the overhead of simulating the RAM model in Turing Machines is always polynomial.

[3] As all reductions we will treat are many-to-one (see [5]) we keep this implicit.

[4] We cannot say exactly that $f \in P$, as $f$ is a function and not a problem or language. The class of this kind of functions is called FP.

# Chapter 2

# A framework to measure and compare model interpretability

Although interpretability has a clear social component [38], and it is usually defined in terms of *human understanding* (see Section 1.2), many computational techniques aim to enhance interpretability of a given model by computing local post-hoc *explanations* for its decisions [6]. Recall that a local post-hoc explanation is one that, given a model and a particular input, aims to elucidate the reason *why* the model took a certain decision on that particular input, without necessarily giving any insight on the model processing or its general behavior (see Section 1.2.2). Experimental evaluation studies with humans have shown that post-hoc explanations do have impact on their understanding and trust for models [46].

We propose the notion of *interpretability in terms of complexity* (*c-interpretability* for short), to describe the computational complexity of computing post-hoc explanations for a given model. In a nutshell, a model is c-interpretable if it is easy, computationally speaking, to provide certain kinds of post-hoc explanations to its user with the assistance of a computer. The precise relationship between the notion of c-interpretability, and the *traditional* notion of interpretability (as in Proposed Definition 1) is discussed in Chapter 6.

We now proceed to describe the different ingredients of the proposed framework.

## 2.1 Models and instances

We define a model $\mathcal{M}$ as a Boolean function $\mathcal{M}\colon \{0,1\}^n \to \{0,1\}$, where $n$ is a positive integer that we call the *input size* of $\mathcal{M}$. That is, we focus on binary classifiers with Boolean input features. Restricting inputs and outputs to be Booleans makes our setting cleaner while still covering several relevant practical scenarios. For example, deciding whether a black and white picture contains a cat or not, as discussed in Section 1.1, falls naturally into our definition of a model. Moreover, if one where to consider colored pictures instead, it would be enough to use 24 Boolean input features per pixel, representing a 24-bit depth color[1].

[1]Representing a ingle pixel by 24 Boolean features might not be practical, but illustrates the generality of the Boolean setting. Such differences in representation would not make a difference for our theoretical results, as the

A class of models is just a way of grouping models together. Models tend to be grouped based on specific algorithms, as we will see in Chapter 3.

An *instance* is a vector in $\{0,1\}^n$ and represents a possible input for a model. We say a Boolean vector $\boldsymbol{x}$ is an instance of a model $\mathcal{M}$ if the dimension of $\boldsymbol{x}$ matches the input size of $\mathcal{M}$. An instance $\boldsymbol{x}$ of a model $\mathcal{M}$ is said to be *positive* if $\mathcal{M}(\boldsymbol{x}) = 1$, and *negative* otherwise. A *partial instance* is a vector in $\{0,1,\perp\}^n$, with $\perp$ intuitively representing "undefined" components. A partial instance $\boldsymbol{x} \in \{0,1,\perp\}^n$ represents, in a compact way, the set of all instances in $\{0,1\}^n$ that can be obtained by replacing undefined components in $\boldsymbol{x}$ with values in $\{0,1\}$. We call these the *completions* of $\boldsymbol{x}$.

## 2.2 Explainability Queries

An *explainability query* is a question that we ask about a model $\mathcal{M}$ and a (possibly partial) instance $\boldsymbol{x}$, and refers to what the model $\mathcal{M}$ does on instance $\boldsymbol{x}$. We assume all queries to be stated either as *decision problems* (that is, YES/NO queries) or as *counting problems* (queries that ask, for example, how many completions of a partial instance satisfy a given property). Thus, for now we can think of queries simply as functions having models and instances as inputs. We will first introduce some specific queries with an example, and then give formal definitions.

**Example 1** Consider an unethical bank that uses a model to decide whether to accept or reject loan applications. The bank uses binary features like: does the requester has a stable job, are they older than 40, etc.

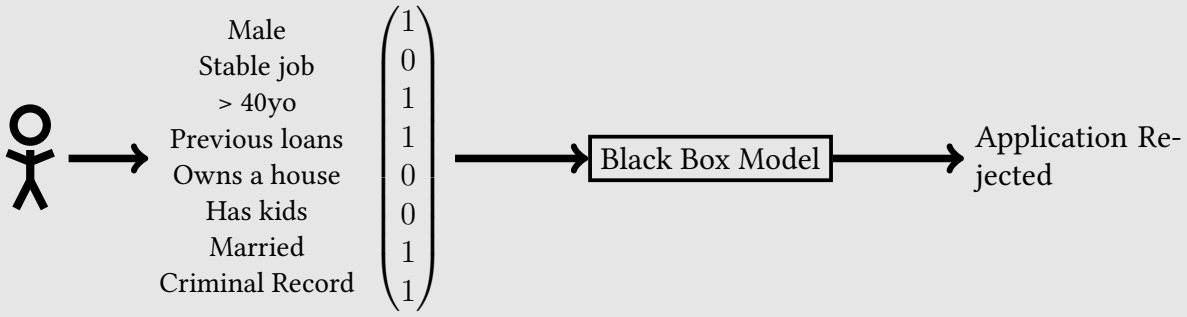



Figure 2.1: Diagram of a particular loan decision.

As represented in Figure 2.1, one day John Doe asks for a loan at this particular bank, and his application gets rejected. John asks the bank executive for a reason "Why did I got rejected?". One can envision 3 different kinds of answer, representative of the queries we will define next.

**Answer 1** If John did not have a criminal record, his application would have been accepted.

sizes of the corresponding models differ only by a multiplicative constant.

**Answer 2** The bank does not make loans for males that are older than 40 and do not own a house, regardless of other features.

**Answer 3** Only 1 out of 20 loan applications by someone who does not own a house get accepted.

Each of these answers can be thought as the solution to an explainability query. Answer 1 describes the *minimum change required* in the input to change the model's prediction. Answer 2 is known as a *minimum sufficient reason* [53] and it describes a minimum set of features that suffice to obtain the current verdict. Finally, Answer 3 describes, given a set of features, the proportion of the inputs that respect those features and are accepted by the model, this is the query we will call *counting completions*.

We proceed to define the main explainability queries we will be dealing with. Several of which are inspired by the work of Darwiche et al. [13, 51, 53], or can be tracked further down to the field of logic [55]. Given instances $\boldsymbol{x}$ and $\boldsymbol{y}$, we define $\mathrm{d}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sum_{\mathrm{i}=1}^{n} |\boldsymbol{x}_{\mathrm{i}} - \boldsymbol{y}_{\mathrm{i}}|$ as the number of components in which $\boldsymbol{x}$ and $\boldsymbol{y}$ differ. We now formalize the minimum-change-required problem, which checks if the output of the model can be changed by flipping the value of at most $k$ components in the input.

| | |
|---|---|
| Problem: | MinimumChangeRequired (MCR) |
| Input: | Model $\mathcal{M}$, instance $\boldsymbol{x}$, and $k \in \mathbb{N}$ |
| Output: | Yes, if there exists an instance $\boldsymbol{y}$ with $\mathrm{d}(\boldsymbol{x}, \boldsymbol{y}) \leq k$ and $\mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y})$, and No otherwise |

The intuitive idea is to explain a decision by making explicit why the result could not have been different. This kind of explanations is known in the literature as a *counterfactual* explanation [38], and interestingly enough, several studies on explanations claim that *counterfactuals* are the most effective kind of explanations [38].

Notice that, in the above definition, instead of "finding" the minimum change we state the problem as a Yes/No query (a decision problem) by adding an additional input $k \in \mathbb{N}$ and then asking for a change of size at most $k$. This is a standard way of stating a problem to analyze its complexity [5]. Moreover, in our results, when we are able to solve the problem in PTIME then we can also output a minimum change, and it is clear that if the decision problem is hard then the optimization problem is also hard. Hence, we can indeed state our problems as decision problems without loss of generality.

To introduce our next query, recall that a partial instance is a vector $\boldsymbol{y} = (y_1, \ldots, y_n) \in \{0, 1, \bot\}^n$, and a completion of it is an instance $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0, 1\}^n$ such that for every i where $y_{\mathrm{i}} \in \{0, 1\}$ it holds that $x_{\mathrm{i}} = y_{\mathrm{i}}$. That is, $\boldsymbol{x}$ coincides with $\boldsymbol{y}$ on all the components of $\boldsymbol{y}$ that are not $\bot$. Given an instance $\boldsymbol{x}$ and a model $\mathcal{M}$, a *sufficient reason for $\boldsymbol{x}$ with respect to $\mathcal{M}$* is a partial instance $\boldsymbol{y}$, such that $\boldsymbol{x}$ is a completion of $\boldsymbol{y}$ and every possible completion $\boldsymbol{x}'$ of $\boldsymbol{y}$ satisfies $\mathcal{M}(\boldsymbol{x}') = \mathcal{M}(\boldsymbol{x})$. That is, knowing the value of the components that are defined in $\boldsymbol{y}$ is enough to determine the output $\mathcal{M}(\boldsymbol{x})$. For a partial instance $\boldsymbol{y}$, let us write $\|\boldsymbol{y}\|$ for its number of components that are not $\bot$.

| | |
|---|---|
| Problem: | MinimumSufficientReason (MSR) |
| Input: | Model $\mathcal{M}$, instance $\boldsymbol{x}$, and $k \in \mathbb{N}$ |
| Output: | Yes, if there exists a sufficient reason $\boldsymbol{y}$ for $\boldsymbol{x}$ wrt. $\mathcal{M}$ with $\|\boldsymbol{y}\| \leq k$, and No otherwise |

We define two simplified[2] variants of this query as well.

| | |
|---|---|
| Problem: | CheckSufficientReason (CSR) |
| Input: | Model $\mathcal{M}$, instance $\boldsymbol{x}$, partial instance $\boldsymbol{y}$ |
| Output: | Yes, if $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$ wrt. $\mathcal{M}$, and No otherwise |

In order to write comfortably the second variant of the MSR query, we say a partial instance $\boldsymbol{y}$ is *contained* in another partial instance $\boldsymbol{z}$ if $\boldsymbol{y}$ coincides with $\boldsymbol{z}$ on every component where $\boldsymbol{y}$ does not have $\perp$.

| | |
|---|---|
| Problem: | MinimalSufficientReason (mSR) |
| Input: | Model $\mathcal{M}$, instance $\boldsymbol{x}$, partial instance $\boldsymbol{y}$ |
| Output: | Yes, if $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$ wrt. $\mathcal{M}$ that does not contain any other sufficient reason for $\boldsymbol{x}$, and No otherwise |

The last query refers to counting the number of positive completions for a given partial instance.

| | |
|---|---|
| Problem: | CountCompletions (CC) |
| Input: | Model $\mathcal{M}$, partial instance $\boldsymbol{y}$ |
| Output: | The number of completions $\boldsymbol{x}$ of $\boldsymbol{y}$ such that $\mathcal{M}(\boldsymbol{x}) = 1$ |

Intuitively, this query informs us on the proportion of inputs that are accepted by the model, given that some particular features have been fixed; or, equivalently, on the *probability* that such an instance is accepted, assuming the other features to be uniformly and independently distributed.

## 2.3 Interpretability in terms of complexity

Given an explainability query $Q$ and a class $\mathcal{C}$ of models, we denote by $Q(\mathcal{C})$ the computational problem defined by $Q$ restricted to models in $\mathcal{C}$. We define next the most important notion for our framework: that of being *more interpretable in terms of complexity* (*c-interpretable* for short). We use this notion to compare among classes of models.

**Definition 2.1** Let $Q$ be an explainability query, and $\mathcal{C}_1$ and $\mathcal{C}_2$ be two classes of models. We say that $\mathcal{C}_1$ is *strictly more c-interpretable than $\mathcal{C}_2$ with respect to $Q$*, if the query $Q(\mathcal{C}_1)$ is in the complexity class $\mathcal{K}_1$, the query $Q(\mathcal{C}_2)$ is hard for complexity class $\mathcal{K}_2$, and $\mathcal{K}_1 \subsetneq \mathcal{K}_2$. .

[2]It might not be obvious that the next variants are computationally simpler, but it is seen to be the case in Chapter 3

We note that, amongst the complexity classes presented in Section 1.3, no strict inclusions are known, so $\mathcal{K}_1 \subsetneq \mathcal{K}_2$ will be interpreted from now on as having always the implicit subsequent "*under theoretical assumptions*".

# Chapter 3

# The computational complexity of interpreting different models

We start this chapter by presenting the details of the models concerned by our study, to then proceed with the presentation and proof of our main results.

## 3.1 Specific models

**Free Binary Decision Diagram (FBDD).** A *binary decision diagram* (BDD [56]) is a rooted directed acyclic graph $\mathcal{M}$ with labels on edges and nodes, verifying: (i) each leaf is labeled with true or with false; (ii) each internal node (a node that is not a leaf) is labeled with an element of $\{1, \ldots, n\}$; and (iii) each internal node has an outgoing edge labeled 1 and another one labeled 0. Every instance $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ defines a unique path $\pi_{\boldsymbol{x}}$ from the root to a leaf in $\mathcal{M}$, which satisfies the following condition: for every non-leaf node $u$ in $\pi_{\boldsymbol{x}}$, if i is the label of $u$, then the path $\pi_{\boldsymbol{x}}$ goes through the edge that is labeled with $x_{\mathrm{i}}$. The instance $\boldsymbol{x}$ is positive, i.e., $\mathcal{M}(\boldsymbol{x}) \coloneqq 1$, if the label of the leaf in the path $\pi_{\boldsymbol{x}}$ is true, and negative otherwise. The *size* $|\mathcal{M}|$ of $\mathcal{M}$ is its number of edges. A binary decision diagram $\mathcal{M}$ is *free* (FBDD) if for every path from the root to a leaf, no two nodes on that path have the same label. A *decision tree* is an FBDD whose underlying graph is a tree. An example is presented in Figure 3.1.

**Multilayer perceptron (MLP).** A multilayer perceptron $\mathcal{M}$ with $k$ layers is defined by a sequence of *weight* matrices $\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(k)}$, *bias* vectors $\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(k)}$, and *activation* functions $f^{(1)}, \ldots, f^{(k)}$. Given an instance $\boldsymbol{x}$, we inductively define

$$\boldsymbol{h}^{(\mathrm{i})} \coloneqq f^{(\mathrm{i})}(\boldsymbol{h}^{(\mathrm{i}-1)}\boldsymbol{W}^{(\mathrm{i})} + \boldsymbol{b}^{(\mathrm{i})}) \qquad (\mathrm{i} \in \{1, \ldots, k\}), \tag{3.1}$$

assuming that $\boldsymbol{h}^{(0)} \coloneqq \boldsymbol{x}$. The output of $\mathcal{M}$ on $\boldsymbol{x}$ is defined as $\mathcal{M}(\boldsymbol{x}) \coloneqq \boldsymbol{h}^{(k)}$. In this thesis we assume all weights and biases to be rational numbers. That is, we assume that there exists a sequence of positive integers $\mathrm{d}_0, \mathrm{d}_1, \ldots, \mathrm{d}_k$ such that $\boldsymbol{W}^{(\mathrm{i})} \in \mathbb{Q}^{\mathrm{d}_{\mathrm{i}-1} \times \mathrm{d}_{\mathrm{i}}}$ and $\boldsymbol{b}^{(\mathrm{i})} \in \mathbb{Q}^{\mathrm{d}_{\mathrm{i}}}$. The integer $\mathrm{d}_0$ is called the *input size* of $\mathcal{M}$, and $\mathrm{d}_k$ the *output size*. Given that we are interested in binary classifiers, we assume that $\mathrm{d}_k = 1$. We say that an MLP as defined above has $(k-1)$ *hidden layers*. The *size* of an MLP $\mathcal{M}$, denoted by $|\mathcal{M}|$, is the total size of its weights and biases, in which the size of a rational number $p/q$ is $\log_2(p) + \log_2(q)$ (with the convention that $\log_2(0) = 1$).

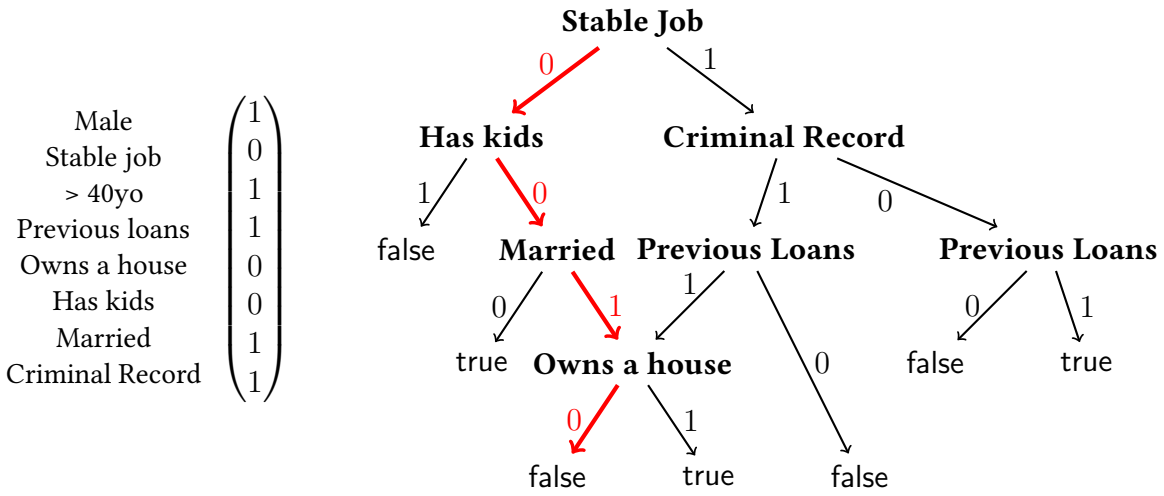



Figure 3.1: Example of an FBDD for the loan problem described in Example 1. The path followed by the example input is highlighted, and results in a rejection.

We focus on MLPs in which all internal functions $f^{(1)}, \ldots, f^{(k-1)}$ are the ReLU function $\mathrm{relu}(x) \coloneqq \max(0, x)$. Usually, MLP binary classifiers are trained using the *sigmoid* as the output function $f^{(k)}$. Nevertheless, when an MLP classifies an input (after training), it takes decisions by simply using the *pre activations*, also called *logits*. Based on this and on the fact that we only consider already trained MLPs, we can assume without loss of generality that the output function $f^{(k)}$ is the *binary step* function, defined as $\mathrm{step}(x) \coloneqq 0$ if $x < 0$, and $\mathrm{step}(x) \coloneqq 1$ if $x \geq 0$. An example is presented in Figure 3.2.

**Perceptron.** A perceptron is an MLP with no hidden layers (i.e., $k = 1$). That is, a perceptron $\mathcal{M}$ is defined by a pair $(\boldsymbol{W}, \boldsymbol{b})$ such that $\boldsymbol{W} \in \mathbb{Q}^{\mathrm{d}\times 1}$ and $\boldsymbol{b} \in \mathbb{Q}$, and the output is $\mathcal{M}(\boldsymbol{x}) = \mathrm{step}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$. An example is presented in Figure 3.3.

Because of its particular structure, a perceptron is usually defined as a pair $(\boldsymbol{w}, b)$ with $\boldsymbol{w}$ a rational vector and $b$ a rational number. The output of $\mathcal{M}(\boldsymbol{x})$ is then 1 if and only if $\langle \boldsymbol{x}, \boldsymbol{w} \rangle + b \geq 0$, where $\langle \boldsymbol{x}, \boldsymbol{w} \rangle$ denotes the dot product between $\boldsymbol{x}$ and $\boldsymbol{w}$.

## 3.2 Main interpretability theorems

We can now state our main results. In all these theorems we use $\mathcal{C}_{\mathrm{MLP}}$ to denote the class of all models (functions from $\{0,1\}^n$ to $\{0,1\}$) that are defined by MLPs, and similarly for $\mathcal{C}_{\mathrm{FBDD}}$ and $\mathcal{C}_{\mathrm{Perceptron}}$. The proofs for all these results will follow as corollaries from the detailed complexity analysis that we present in Section 3.3. We start by stating a strong separation between FBDDs and MLPs, which holds for all the queries presented above.

**Theorem 3.1** *$\mathcal{C}_{FBDD}$ is strictly more c-interpretable than $\mathcal{C}_{MLP}$ with respect to MCR, CSR, MSR, MSR, and CC*

For the comparison between perceptrons and MLPs, we can establish a strict separation for

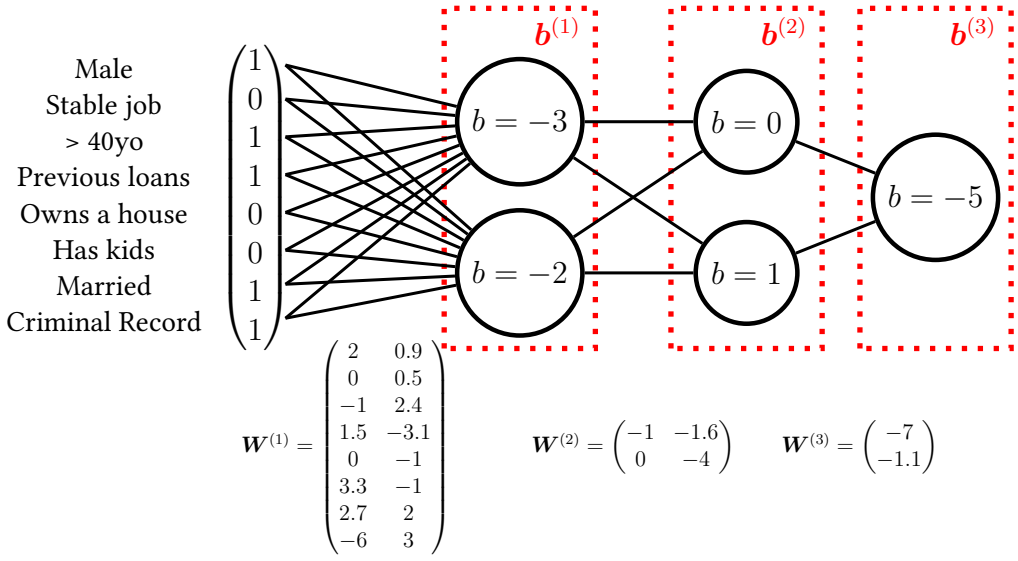



Figure 3.2: Example of an MLP for the loan problem described in Example 1. The result is a rejection.

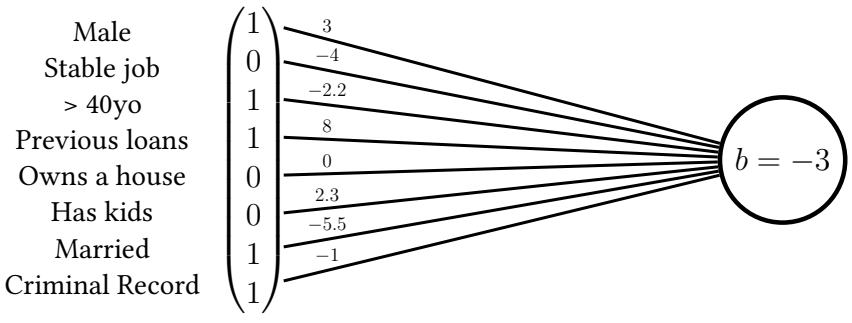



Figure 3.3: Example of a perceptron for the loan problem described in Example 1. The result is a rejection.

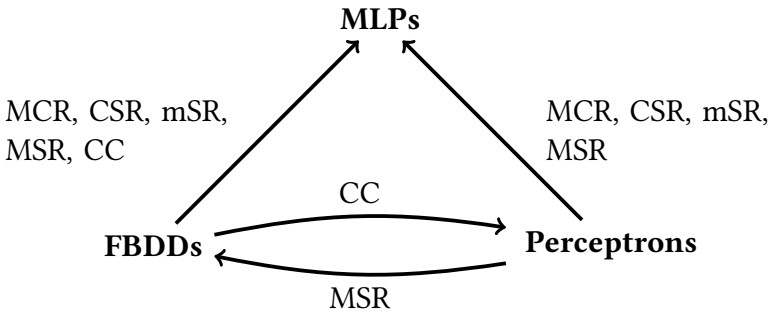



Figure 3.4: Illustration of the main interpretability results. Arrows depict that the pointed class of models is harder with respect to the query that labels the edge.

| | **FBDDs** | **Perceptrons** | **MLPs** |
|---|---|---|---|
| MinimumChangeRequired | PTIME (3.5) | PTIME (3.6) | NP-complete (3.8) |
| CheckSufficientReason | PTIME (3.10) | PTIME (3.12) | coNP-complete (3.13) |
| MinimalSufficientReason | PTIME (3.17) | PTIME (3.17) | DP-complete (3.18) |
| MinimumSufficientReason | NP-complete (3.20) | PTIME (3.21) | $\Sigma_2^p$-complete (3.23) |
| CountCompletions | PTIME (3.25) | #P-complete (3.28) | #P-complete (3.28) |

Table 3.1: Summary of our complexity results.

MCR, MSR, CSR and mSR but not for CC. In fact, CC has the same complexity for both classes of models, which means that none of these classes strictly "dominates" the other in terms of c-interpretability for CC.

**Theorem 3.2** $\mathcal{C}_{Perceptron}$ is strictly more c-interpretable than $\mathcal{C}_{MLP}$ with respect to MCR, CSR, mSR and MSR. In turn, the problems CC($\mathcal{C}_{Perceptron}$) and CC($\mathcal{C}_{MLP}$) are both complete for the same complexity class.

The next result shows that, in terms of c-interpretability, the relationship between FBDDs and perceptrons is not clear, as each one of them is strictly more c-interpretable than the other for some explainability query.

**Theorem 3.3** $\mathcal{C}_{Perceptron}$ is strictly more c-interpretable than $\mathcal{C}_{FBDD}$ with respect to MSR. In turn, $\mathcal{C}_{FBDD}$ is strictly more c-interpretable than $\mathcal{C}_{Perceptron}$ with respect to CC.

These results can be visualized in Figure 3.4. Proofs are presented in the next section, where for each query $Q$ and class of models $\mathcal{C}$ we pinpoint the exact complexity of the problem $Q(\mathcal{C})$.

## 3.3 The complexity of interpretability queries

We divide our results in terms of the queries that we consider. We also present a few other complexity results that we find interesting on their own. A summary of the results is shown in Table 3.1.

### 3.3.1 The complexity of MinimumChangeRequired

**Theorem 3.4** The MinimumChangeRequired query is (1) in PTIME for FBDDs, (2) in PTIME for perceptrons, and (3) NP-complete for MLPs.

We prove each item in a separate lemma.

**Lemma 3.5** The MinimumChangeRequired query can be solved in linear time for FBDDs.

Proof. Let $(\mathcal{M}, \boldsymbol{x}, k)$ be an instance of MinimumChangeRequired, where $\mathcal{M}$ is an FBDD. For every node $u$ in $\mathcal{M}$ we define $\mathcal{M}_u$ to be the FBDD obtained by restricting $\mathcal{M}$ to the nodes that are (forward-)reachable from $u$; in other words, $\mathcal{M}_u$ is the sub-FBDD rooted at $u$. Then, we define $\mathrm{mcr}_u(\boldsymbol{x})$ to be the minimum change required on $\boldsymbol{x}$ to obtain a classification under $\mathcal{M}_u$ that differs from $\mathcal{M}(\boldsymbol{x})$. More formally,

$$\mathrm{mcr}_u(\boldsymbol{x}) = \min\{k' \mid \text{there exists an instance } \boldsymbol{y} \text{ such that } \mathrm{d}(\boldsymbol{x}, \boldsymbol{y}) = k' \text{ and } \mathcal{M}_u(\boldsymbol{y}) \neq \mathcal{M}(\boldsymbol{x})\},$$

with the convention that $\min \varnothing = \infty$. Let $r$ be the root of $\mathcal{M}$. Then, by definition we have that $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of MinimumChangeRequired if and only $\mathrm{mcr}_r(\boldsymbol{x}) \leq k$. We now explain how we can compute all the values $\mathrm{mcr}_u(\boldsymbol{x})$ for every node $u$ of $\mathcal{M}$ in linear time.

Observe that, (†) for an instance $\boldsymbol{y}$ minimizing $k'$ in this equality, since the FBDD $\mathcal{M}_u$ does not depend on the features associated to any node $u'$ from the root of $\mathcal{M}$ to $u$ excluded, we have that for any such node $y_{u'} = x_{u'}$ holds (otherwise $k'$ would not be minimized).[1]

By definition, if $u$ is a leaf labeled with true we have that $\mathcal{M}_u(\boldsymbol{y}) = 1$ for every $\boldsymbol{y}$, and thus if $\mathcal{M}(\boldsymbol{x}) = 0$ we get $\mathrm{mcr}_u(\boldsymbol{x}) = 0$, while if $\mathcal{M}(\boldsymbol{x}) = 1$ we get that $\mathrm{mcr}_u(\boldsymbol{x}) = \infty$. Analogously, if $u$ is a leaf labeled with false, then $\mathrm{mcr}_u(\boldsymbol{x})$ is equal to $0$ if $\mathcal{M}(\boldsymbol{x}) = 1$ and to $\infty$ otherwise.

For the recursive case, we consider a non-leaf node $u$. Let $u_1$ be the node going along the edge labeled with $1$ from $u$, and $u_0$ analogously. Using the notation $[x_u = a]$ to mean $1$ if the feature of $\boldsymbol{x}$ indexed by the label of node $u$ has value $a \in \{0, 1\}$, and $0$ otherwise, and the convention that $\infty + 1 = \infty$, we claim that:

$$\mathrm{mcr}_u(\boldsymbol{x}) = \min\Big([x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x}), [x_u = 0] + \mathrm{mcr}_{u_1}(\boldsymbol{x})\Big)$$

Indeed, consider by inductive hypothesis that $\mathrm{mcr}_{u_0}(\boldsymbol{x})$ and $\mathrm{mcr}_{u_1}(\boldsymbol{x})$ have been properly calculated, and let us show that this equality holds. We prove both inequalities in turn:

- We show that $\mathrm{mcr}_u(\boldsymbol{x}) \leq \min\Big([x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x}), [x_u = 0] + \mathrm{mcr}_{u_1}(\boldsymbol{x})\Big)$. It is enough to show that both $\mathrm{mcr}_u(\boldsymbol{x}) \leq [x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x})$ and $\mathrm{mcr}_u(\boldsymbol{x}) \leq [x_u = 0] + \mathrm{mcr}_{u_1}(\boldsymbol{x})$ hold. We only show the first inequality, as the other one is similar. If $\mathrm{mcr}_{u_0}(\boldsymbol{x}) = \infty$ then clearly

[1] We slightly abuse notation and write $x_u$ for the value of the feature of $\boldsymbol{x}$ that is indexed by the label of $u$.

the inequality holds, hence let us assume that $\mathrm{mcr}_{u_0}(\boldsymbol{x}) = k' \in \mathbb{N}$. This means that there is an instance $\boldsymbol{y}'$ such that $\mathrm{d}(\boldsymbol{x}, \boldsymbol{y}') = k'$ and such that $\mathcal{M}_{u_0}(\boldsymbol{y}') \neq \mathcal{M}(\boldsymbol{x})$. Furthermore, by the observation (†) we have that for any node $u'$ from the root of $\mathcal{M}$ to $u$ (included), we have $y_{u'} = x_{u'}$. Therefore, the instance $\boldsymbol{y}$ that is equal to $\boldsymbol{y}'$ but has value $y_u = 0$ differs from $\boldsymbol{x}$ in exactly $k'' = [x_u = 1] + k'$, which implies that $\mathrm{mcr}_u(\boldsymbol{x}) \leq [x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x})$. Hence, the first inequality is proven.

- We show that $\mathrm{mcr}_u(\boldsymbol{x}) \geq \min\Big([x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x}), [x_u = 0] + \mathrm{mcr}_{u_1}(\boldsymbol{x})\Big)$. First, assume that both $\mathrm{mcr}_{u_0}(\boldsymbol{x})$ and $\mathrm{mcr}_{u_1}(\boldsymbol{x})$ are equal to $\infty$. This means that every path in both $\mathcal{M}_{u_0}$ and $\mathcal{M}_{u_1}$ leads to a leaf with the same classification as $\mathcal{M}(\boldsymbol{x})$. Then, as every path from $u$ goes either through $u_0$ or through $u_1$, it must be that every path from $u$ leads to a leaf with the same classification as $\mathcal{M}(\boldsymbol{x})$, and thus $\mathrm{mcr}_u(\boldsymbol{x}) = \infty$, and so the inequality holds. Therefore, we can assume that one of $\mathrm{mcr}_{u_0}(\boldsymbol{x})$ or $\mathrm{mcr}_{u_1}(\boldsymbol{x})$ is finite. Let us assume without loss of generality that (⋆) $\min\Big([x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x}), [x_u = 0] + \mathrm{mcr}_{u_1}(\boldsymbol{x})\Big) = [x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x}) \in \mathbb{N}$ (the other case being similar). Let us now assume, by way of contradiction, that the inequality does not hold, that is, we have that (††) $\mathrm{mcr}_u(\boldsymbol{x}) < [x_u = 1] + \mathrm{mcr}_{u_0}(\boldsymbol{x})$, and let $\boldsymbol{y}$ be an instance such that $\mathcal{M}_u(\boldsymbol{y}) \neq \mathcal{M}_u(\boldsymbol{x})$ and $\mathrm{d}(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{mcr}_u(\boldsymbol{x})$. Thanks to (⋆), we can assume wlog that $y_u = 0$. But then we would have that $\mathrm{mcr}_{u_0}(\boldsymbol{x}) \leq \mathrm{mcr}_u(\boldsymbol{x}) - [x_u = 1]$, which contradicts (††). Hence, the second inequality is proven.

It is clear that the recursive function $\mathrm{mcr}$ can be computed bottom-up in linear time, thus concluding the proof. □

**Lemma 3.6** The MinimumChangeRequired query can be solved in linear time for perceptrons.

Proof. Let $(\mathcal{M} = (\boldsymbol{w}, b), \boldsymbol{x}, k)$ be an instance of the problem, and let us assume without loss of generality that $\mathcal{M}(\boldsymbol{x}) = 1$, as the other case is analogous. For each feature $\mathrm{i}$ of $\boldsymbol{x}$ we define its importance $s(\mathrm{i})$ as $w_\mathrm{i}$ if $x_\mathrm{i} = 1$ and $-w_\mathrm{i}$ otherwise. Intuitively, $s$ represents how good it is to keep a certain feature in order to maintain the verdict of the model. We now assume (being ready to pay for it) that $\boldsymbol{x}$ and $\boldsymbol{w}$ have been sorted in decreasing order of score $s$. For example, if originally $\boldsymbol{w} = (3, -5, -2)$ and $\boldsymbol{x} = (1, 0, 1)$, then after the sorting procedure we have $\boldsymbol{w} = (-5, 3, -2)$ and $\boldsymbol{x} = (0, 1, 1)$. This sorting procedure has cost $O(|\mathcal{M}|)$ as it is a classical problem of sorting strings whose total length add up to $\mathcal{M}$ and can be carried with a variant of Bucketsort [12]. As a result, for every pair $1 \leq \mathrm{i} < j \leq n$ we have that $s(\mathrm{i}) \geq s(j)$.

Let $k'$ be the largest integer no greater than $k$ such that $s(k') > 0$ and then define $\boldsymbol{x}'$ as the instance that differs from $\boldsymbol{x}$ exactly on the first $k'$ features. We claim that $\mathcal{M}(\boldsymbol{x}') \neq \mathcal{M}(\boldsymbol{x})$ if and only if $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of MinimumChangeRequired. The forward direction follows from the fact that $k' \leq k$. For the backward direction, assume that $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of MinimumChangeRequired. This implies that there is an instance $\boldsymbol{y}$ that differs from $\boldsymbol{x}$ in at most $k$ features, and for which $\mathcal{M}(\boldsymbol{y}) = 0$. If $\boldsymbol{y} = \boldsymbol{x}'$, then we are immediately done, so we can safely assume this is not the cae.

We then define, for any instance $\boldsymbol{y}$ of $\mathcal{M}$ the function $v(\boldsymbol{y}) = \langle \boldsymbol{w}, \boldsymbol{y} \rangle$. Note that an instance $\boldsymbol{y}$

of $\mathcal{M}$ is positive if and only if $v(\boldsymbol{y}) \geq -b$. Then, since we have that $\mathcal{M}(\boldsymbol{y}) = 0$, it holds that $v(\boldsymbol{y}) < -b$. We now claim that $v(\boldsymbol{x}') \leq v(\boldsymbol{y})$:

**Claim 3.7** For every instance $\boldsymbol{y}$ such that $\mathrm{d}(\boldsymbol{y}, \boldsymbol{x}) \leq k$ and $\mathcal{M}(\boldsymbol{y}) \neq \mathcal{M}(\boldsymbol{x})$, it must hold that $v(\boldsymbol{x}') \leq v(\boldsymbol{y})$.

Proof. For an instance $\boldsymbol{z}$, let us write $C_{\boldsymbol{z}}$ for the set of features for which $\boldsymbol{z}$ differs from $\boldsymbol{x}$. We then have on the one hand

$$v(\boldsymbol{x}') = \sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}} (1 - x_\mathrm{i}) w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \cap C_{\boldsymbol{x}'}} (1 - x_\mathrm{i}) w_\mathrm{i} + \sum_{\mathrm{i} \notin C_{\boldsymbol{x}'} \cup C_{\boldsymbol{y}}} x_\mathrm{i} w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}} x_\mathrm{i} w_\mathrm{i}$$

and on the other hand

$$v(\boldsymbol{y}) = \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}} (1 - x_\mathrm{i}) w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \cap C_{\boldsymbol{x}'}} (1 - x_\mathrm{i}) w_\mathrm{i} + \sum_{\mathrm{i} \notin C_{\boldsymbol{x}'} \cup C_{\boldsymbol{y}}} x_\mathrm{i} w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}} x_\mathrm{i} w_\mathrm{i}$$

As the two middle terms are shared, we only need to prove that

$$\sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}} (1 - x_\mathrm{i}) w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}} x_\mathrm{i} w_\mathrm{i} \leq \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}} (1 - x_\mathrm{i}) w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}} x_\mathrm{i} w_\mathrm{i}$$

which is equivalent to proving that

$$\sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}, x_\mathrm{i} = 0} w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}, x_\mathrm{i} = 1} w_\mathrm{i} \leq \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}, x_\mathrm{i} = 0} w_\mathrm{i} + \sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}, x_\mathrm{i} = 1} w_\mathrm{i}$$

and by using the definition of importance, equivalent to

$$\sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}, x_\mathrm{i} = 0} -s(\mathrm{i}) + \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}, x_\mathrm{i} = 1} s(\mathrm{i}) \leq \sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}, x_\mathrm{i} = 0} -s(\mathrm{i}) + \sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}, x_\mathrm{i} = 1} s(\mathrm{i})$$

which can be rearranged into

$$\sum_{\mathrm{i} \in C_{\boldsymbol{y}} \setminus C_{\boldsymbol{x}'}} s(\mathrm{i}) \leq \sum_{\mathrm{i} \in C_{\boldsymbol{x}'} \setminus C_{\boldsymbol{y}}} s(\mathrm{i})$$

But this inequality must hold as $C'_{\boldsymbol{x}}$ is by definition the set $C$ of features of size at most $k$ that maximizes $\sum_{\mathrm{i} \in C} s(\mathrm{i})$. □

Because of the claim, and the fact that $v(\boldsymbol{y}) < -b$ we conclude that $v(\boldsymbol{x}') < -b$, and thus $\mathcal{M}(\boldsymbol{x}') \neq \mathcal{M}(\boldsymbol{x})$. This concludes the backward direction, and thus, the fact that checking whether $\mathcal{M}(\boldsymbol{x}') \neq \mathcal{M}(\boldsymbol{x})$ is enough to solve the entire problem. Since checking this can be done in linear time, and so can do the construction of $\boldsymbol{x}'$ (with the initial sorting procedure), we achieve the desired runtime. This concludes the proof of the lemma. □

**Lemma 3.8** The MinimumChangeRequired query is NP-complete for MLPs.

In order to prove this lemma, we will prove first a general result about the ability of MLPs to simulate arbitrary Boolean formulas. As it is well known that many problems are NP for Boolean formulas (e.g. satisfiability, tautology, etc.) this simulation lemma will allow us to prove most hardness result concerning MLPs.

In fact, and this will make the proof cleaner, we will show a slightly more general result: that MLPs can simulate arbitrary *Boolean circuits*. Formally, we show:

**Lemma 3.9** Given as input a Boolean circuit $C$, we can build in polynomial time an MLP $\mathcal{M}_C$ that is equivalent to $C$ as a Boolean function.

Proof. We will proceed in three steps. The first step is to build from $C$ another equivalent circuit $C'$ that uses only what we call *relu gates*. A relu gate is a gate that, on input $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$, outputs $\mathrm{relu}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)$, for some rationals $w_1, \ldots, w_m, b$. Observe that these gates do not necessarily output 0 or 1, and so the circuit $C'$ might not be Boolean. However, we will ensure in the construction that the output of every relu gate in $C'$, when given Boolean inputs (i.e., $\boldsymbol{x} \in \{0,1\}^m$), is Boolean. This will imply that the circuit $C'$ is Boolean as well. To this end, it is enough to show how to simulate each original type of internal gate (not, or, and) by relu gates:

- not-gate: simulated with a relu gate with only one weight of value $-1$ and a bias of 1. Indeed, it is clear that for $x \in \{0,1\}$, we have that $\mathrm{relu}(-x+1) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases}$.
- and-gate of in-degree $n$: simulated with a relu gate with $n$ weights, each of value 1, and a bias of value $-(n-1)$. Indeed, it is clear that for $\boldsymbol{x} \in \{0,1\}^n$, we have that $\mathrm{relu}(\sum_{\mathrm{i}=1}^{n} x_{\mathrm{i}} - (n-1)) = \begin{cases} 1 & \text{if } \bigwedge_{\mathrm{i}=1}^{n} x_{\mathrm{i}} = 1 \\ 0 & \text{otherwise} \end{cases}$.
- or-gate of in-degree $n$: we first rewrite the or-gate with not- and and-gates using De Morgan's laws, and then we use the last two items.

The second step is to build a circuit $C''$, again using only relu gates, that is equivalent to $C'$ and that is what we call *layerized*. This means that there exists a *leveling function* $l : C'' \to \mathbb{N}$ that assigns to every gate of $C'$ a *level* such that (i) every variable gate is assigned level 0, and (i) for any wire $g \to g'$ (meaning that $g$ is an input to $g'$) in $C''$ we have that $l(g') = l(g) + 1$. To this end, let us call a relu gate that has a single input and weight 1 and bias 0 an *identity gate*, and observe then that the value of an identity gate is the same as the value of its only input, when this input is in $\{0,1\}$. We will obtain $C''$ from $C'$ by inserting identity gates in between the gates of $C'$, which clearly does not change the Boolean function being computed. We can do so naïvely as follows. First, we initialize $l(g)$ to 0 for all the variable gates $g$ of $C'$. We then iterate the following process: select a gate $g$ such that $l(g)$ is undefined and such that $l(g')$ is defined for every input $g'$ of $g$. Let $g'_1, \ldots, g'_m$ be the inputs of $g$, and assume that $l(g'_1) \leq \ldots \leq l(g'_m)$. For every $1 \leq \mathrm{i} \leq m$, we insert a line of $l(g'_m) - l(g'_{\mathrm{i}})$ identity gates in between $g'_{\mathrm{i}}$ and $g$, and we set $l(g) := l(g'_m) + 1$, and we set the levels of the identity gates that we have inserted appropriately. It is clear that this construction can be done in polynomial time and that the resulting circuit $C''$ is layerized.

Finally, the last step is to transform $C''$ into an MLP $\mathcal{M}_C$ using only relu for the internal activation functions and the step function for the output layer (i.e., what we simply call "an MLP" in the paper), and that respects the structure given by our definition in Section 3.1 (i.e., where all neurons of a given layer are connected to all the neurons of the preceding layer). We first deal with having a step gate instead of a relu gate for the output. To achieve this, we create a fresh identity gate $g_0$, we set the output of $C''$ to be an input of $g_0$, and we set $g_0$ to be the new output gate of $C''$ (this does not change the Boolean function computed). We then replace $g_0$ by a *step gate* (which, on input $x \in \mathbb{R}$, outputs $0$ if $x < 0$ and $1$ otherwise) with a weight of $2$ and bias of $-1$, which again does not change the Boolean function computed; indeed, for $x \in \{0, 1\}$, we have that $\mathrm{step}(2x - 1) = \begin{cases} 1 & \text{if } x = 1 \\ 0 & \text{if } x = 0 \end{cases}$.

The level of $g_0$ is one plus the level of the previous output gate of $C''$. Therefore, to make $C''$ become a valid MLP, it is enough to do the following: for every gate $g$ of level $\mathrm{i}$ and gate $g'$ of level $\mathrm{i} + 1$, if $g$ and $g'$ are not connected in $C''$, we make $g$ be an input of $g'$ and we set the corresponding weight to $0$. This clearly does not change the function computed, and the obtained circuit can directly be regarded as an equivalent MLP $\mathcal{M}_C$. Since the whole construction can be performed in polynomial time, this concludes the proof. □

We can now go back to where we where, and prove Lemma 3.8. We state the Lemma once again for convenience.

**Lemma** The MinimumChangeRequired query is NP-complete for MLPs.

Proof. Membership is easy to see, it is enough to non-deterministically guess an instance $\boldsymbol{y}$ and check that $\mathrm{d}(\boldsymbol{x}, \boldsymbol{y}) \leq k$ and $\mathcal{M}(\boldsymbol{x}) = \mathcal{M}(\boldsymbol{y})$.

In order to prove hardness, we reduce from Vertex Cover, the problem of, given an undirected graph $G$ and an integer $k$, decide whether there is a subset $S$ of at most $k$ vertices such that every edge of $G$ touches a vertex in $S$. Let $(G = (V, E), k)$ be an instance of Vertex Cover, and let $n$ denote $|V|$. Based on $G$, we build a formula $\varphi_G$, where propositional variables correspond to vertices of $G$.

$$\varphi_G = \bigwedge_{(u,v) \in E} (x_u \vee x_v)$$

It is clear that the satisfying assignments of $\varphi_G$ correspond to the vertex covers of $G$, and furthermore, that a satisfying assignment of Hamming weight $k$ (number of variables assigned to $1$) corresponds to a vertex cover of size $k$.

Moreover, we can safely assume that there is at least $1$ edge in $G$, as otherwise the instance would be trivial, and a constant size positive instance of MCR would finish the reduction. This implies in turn, that we can assume that assigning every variable to $0$ does not satisfy $\varphi_G$.

We now build an MLP $\mathcal{M}_\varphi$ from $\varphi_G$, using Lemma 3.9. We claim that the instance $(\mathcal{M}_\varphi, 0^n, k)$

is a positive instance of MinimumChangeRequired if and only if $(G, k)$ is a positive instance of Vertex Cover.

Indeed, $0^n$ is a negative instance of $\mathcal{M}_\varphi$, as assigning every variable to 0 does not satisfy $\varphi_G$. Moreover a positive instance of weight $k$ for $\mathcal{M}_\varphi$ corresponds to a satisfying assignment of weight $k$ for $\varphi_G$, which in turn corresponds to a vertex cover of size $k$ for $G$. This is enough to conclude the proof, recalling that both the construction of $\varphi_G$ and $\mathcal{M}_\varphi$ take polynomial time. □

### 3.3.2 The complexity of CheckSufficientReason

**Theorem 3.10** The query CheckSufficientReason is (1) in PTIME for FBDDs, (2) in PTIME for perceptrons, and (3) co-NP-complete for MLPs.

We prove each item in a separate lemma.

**Lemma 3.11** The query CheckSufficientReason can be solved in linear time for FBDDs.

Proof. Let $(\mathcal{M}, \boldsymbol{x}, \boldsymbol{y})$ be an instance of the problem, with $\mathcal{M}$ being an FBDD. We first check that $\boldsymbol{x}$ is a completion of $\boldsymbol{y}$, which can clearly be done in linear time. We the define $\mathcal{M}'$ as the resulting FBDD from the following procedure: (i) For every internal node in $\mathcal{M}$ with label i, delete its outgoing edge with label 0 if $\boldsymbol{y}_\mathrm{i} = 1$ and its outgoing edge with label 1 if $y_\mathrm{i} = 0$. We note here that $\mathcal{M}'$ is not a well defined FBDDs, since some internal nodes may have only one outgoing edge: more precisely, the value $\mathcal{M}(\boldsymbol{x}') \in \{0, 1\}$ is well defined for every instance $\boldsymbol{x}'$ that is a completion of $\boldsymbol{y}$, and is not defined for an instance $\boldsymbol{x}'$ that is not a completion of $\boldsymbol{y}$. To check whether $\boldsymbol{y}$ is a sufficient reason, we can then simply check that every leaf that is reachable from the root in $\mathcal{M}'$ is labeled the same (either true or false). This can be done in linear time with standard graph algorithms. □

**Lemma 3.12** The query CheckSufficientReason can be solved in linear time for perceptrons.

Proof. Let $(\mathcal{M} = (\boldsymbol{w}, b), \boldsymbol{x}, \boldsymbol{y})$ be an instance of the problem. We first check in linear time that $\boldsymbol{x}$ is a completion of $\boldsymbol{y}$. We then get rid of the components that are defined by $\boldsymbol{y}$, as follows. We define:

- $A \coloneqq \sum_{y_\mathrm{i} \neq \perp} y_\mathrm{i} w_\mathrm{i}$;
- $\boldsymbol{w}' \coloneqq (w_\mathrm{i} \mid y_\mathrm{i} = \perp)$; and
- $b' \coloneqq b + A$;

and let $\mathcal{M}'$ be the perceptron $(\boldsymbol{w}', b')$. Notice that the dimension of $\mathcal{M}'$ is equal to the number of undefined components of $\boldsymbol{y}$; let us write $m$ this number. It is then clear that $\boldsymbol{y}$ is a sufficient reason of $\boldsymbol{x}$ under $\mathcal{M}$ if and only if every instance of $\mathcal{M}'$ is labeled the same. We can check this as follows. Let $J_1$ be the minimum possible value of $\langle \boldsymbol{w}', \boldsymbol{x}' \rangle$ (for $\boldsymbol{x}' \in \{0, 1\}^m$); $J_1$ can clearly be computed in linear time by setting $x'_\mathrm{i} = 0$ if $w'_\mathrm{i} \geq 0$ and $x'_\mathrm{i} = 1$ otherwise. Similarly we can

compute the maximal possible value $J_2$ of $\langle \boldsymbol{w}', \boldsymbol{x}' \rangle$. Then, every instance of $\mathcal{M}'$ is labeled the same if and only if it is not the case that $J_1 < -b'$ and $J_2 \geq -b'$, thus concluding the proof. □

**Lemma 3.13** The query CheckSufficientReason is co-NP-complete for MLPs.

Proof. We first show membership in co-NP. Let $(\mathcal{M}, \boldsymbol{x}, \boldsymbol{y})$ be an instance of the problem. Then $\boldsymbol{y}$ is a sufficient reason of $\boldsymbol{x}$ under $\mathcal{M}$ if and only if all the completions of $\boldsymbol{y}$ are labeled the same as $\boldsymbol{x}$. Thus, a single completion $\boldsymbol{y}$ with a different verdict than $\boldsymbol{x}$ serves as a certificate for the complemented language.

In order to prove hardness we reduce from TAUT, the problem of checking whether an arbitrary boolean formula is a satisfied by all possible assignments of its variables. This problem is known to be complete for co-NP. Let $\mathcal{F}$ be an arbitrary boolean formula. We use Lemma 3.9 to build an equivalent MLP $\mathcal{M}$ in polynomial time (with the features of $\mathcal{M}$ corresponding to the variables of $\mathcal{F}$). Then $\mathcal{F}$ is a tautology if and only if all completions of the partial instance $\boldsymbol{y} = \perp^n$ are positive instances of $\mathcal{M}$. First, we construct an arbitrary instance $\boldsymbol{x}$ (for instance, the one with all the features being 0), and we reject if $\mathcal{M}(\boldsymbol{x}) = 0$. Then, we accept if $\boldsymbol{y}$ is a sufficient reason of $\boldsymbol{x}$ under $\mathcal{M}$, and we reject otherwise. This concludes the reduction. □

### 3.3.3 The complexity of MinimalSufficientReason

Recall that a partial instance $\boldsymbol{y}$ is said to be a minimal sufficient reason for an instance $\boldsymbol{x}$ under a model $\mathcal{M}$ if $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$ under $\mathcal{M}$ and no proper subset of $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$ wrt. $\mathcal{M}$.

Let us state the main theorem of this subsection.

**Theorem 3.14** The query MinimalSufficientReason is (1) in PTIME for FBDDs, (2) in PTIME for perceptrons, and (3) DP-complete for MLPs.

We start our way towards a proof by stating a folklore lemma that characterizes minimal sufficient reasons in a way that is easier to check. In order to do so, we introduce the notation $\boldsymbol{y} \setminus \{f\}$, for $\boldsymbol{y}$ a partial instance and $f$ a feature, to mean the partial instance $\boldsymbol{y}'$ that is equal to $\boldsymbol{y}$, but for which $y'_f = \perp$. If it was already the case that $y_f = \perp$, then $\boldsymbol{x} = \boldsymbol{x} \setminus \{f\}$.

**Lemma 3.15** (Folklore) A partial instance $\boldsymbol{y}$, that is defined for features $f_1, \ldots f_k$ is a minimal sufficient reason for an instance $\boldsymbol{x}$ under a model $\mathcal{M}$ if and only if $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$ under $\mathcal{M}$ and $\boldsymbol{y} \setminus \{f_{\mathrm{i}}\}$ is not a sufficient reason for $\boldsymbol{x}$ wrt. $\mathcal{M}$ for any $1 \leq \mathrm{i} \leq k$.

A simple proof of this lemma can be found for example in the work of Goldsmith et al. [25].

This simple lemma is enough to prove that checking minimal sufficient reasons is easy for both FBDDs and Perceptrons

**Lemma 3.16** If for a class of models $\mathcal{C}$, the CheckSufficientReason query can be solved in PTIME, then the MinimalSufficientReason query can be solved in PTIME as well.

Proof. Because of Lemma 3.15, it is enough to first verify, given a partial instance $\boldsymbol{y}$, an instance $\boldsymbol{x}$ and a model $\mathcal{M} \in \mathcal{C}$, that $\boldsymbol{y}$ is a sufficient reason for $\mathcal{M}$ (which can be done in PTIME by hypothesis) and then check that no partial instance $\boldsymbol{x} \setminus \{f_i\}$ is not a sufficient reason for $\mathcal{M}$, using again the hypothesis at most $|\mathcal{M}|$ times (which is an upper bound for the number of defined components in $\boldsymbol{x}$). □

**Lemma 3.17** The MinimalSufficientReason query can be solved in PTIME for FBDDs and perceptrons.

Proof. It follows directly from combining Lemma 3.16, Lemma 3.10 and Lemma 3.12. □

We now establish the complexity of this query for MLPs.

**Lemma 3.18** The MinimalSufficientReason query is DP-complete for MLPs.

Proof. Remember that the class DP is the class of languages that correspond to the intersection of a language in NP and a language in co-NP. Based on Lemma 3.15, the inputs that belong to the MinimalSufficientReason language correspond precisely to the intersection between those that belong to the CheckSufficientReason language, and those inputs such that for all removals of a feature $f$ from the input partial instance $y$ one obtains a partial instance that is a sufficient reason for the input instance $\boldsymbol{x}$.

$$\text{MinimalSufficientReason} = \text{CheckSufficientReason} \cap \{(\boldsymbol{x}, \boldsymbol{y}, \mathcal{M}) \mid \forall f \in \boldsymbol{y}, (\boldsymbol{x}, \boldsymbol{y} \setminus \{f\}, \mathcal{M}) \notin \text{CheckSufficientReason}\}$$

While the first language of the intersection is in co-NP because of Lemma 3.13, it is not hard to see that the second one is in NP. Indeed, it is enough to guess one completion for each $\boldsymbol{y} \setminus \{f\}$ that has opposite classification from $\boldsymbol{x}$. We conclude the language is in DP.

In order to prove hardness, we use a result by Goldsmith et al. [25] stating that, given a Boolean formula $\varphi$ and a monomial $C$, it is DP-hard to decide whether $C$ is a prime implicant[2] of $\varphi$. The reduction is straightforward: take an instance $(\varphi, C)$, use Lemma 3.9 to build an MLP $\mathcal{M}_\varphi$, let $\boldsymbol{y}$ be the partial instance naturally induced by $C$, and $\boldsymbol{x}$ an arbitrary completion of $\boldsymbol{y}$.

**Claim 3.19** The monomial $C$ is a prime implicant of $\varphi$ if and only if $\boldsymbol{y}$ is a minimal sufficient reason for $\boldsymbol{x}$ under $\mathcal{M}_\varphi$.

[2] A monomial (that is, a conjunction of literals) $C$ is said to be an implicant of a formula $\varphi$ if the formula $C \implies \varphi$ is valid. An implicant $C$ of a formula $\varphi$ is said to be a prime implicant if no proper subset $C'$ of $C$ is an implicant.

A proof for this claim can be simplified by adopting comfortable notation. In particular, we want to go back and forth from the world of Boolean formulas and monomials to that of models and instances. We thus define the notation $\gamma(\boldsymbol{y}) = C$ as a way to say that $\boldsymbol{y}$ is the uniquely determined partial instance by monomial $C$. As this relation is bijective, we write as well $\gamma^{-1}(C) = \boldsymbol{y}$. Let us note an important property that follows from this notation $(\star)$: for a monomial $C$ and a formula $\varphi$, the formula $C$ is an implicant of $\varphi$ (meaning that $\varphi$ is satisfied by every assignment of its variables that satisfies $C$) if and only if $\mathcal{M}_\varphi(\boldsymbol{x}) = 1$ for every instance $\boldsymbol{x}$ that is a completion of $\gamma^{-1}(C)$. We introduce as well the notation $\boldsymbol{y} \subseteq \boldsymbol{x}$ to reflect that instance $\boldsymbol{x}$ contains partial instance $\boldsymbol{y}$, and similarly, we say $C' \subseteq C$ if the literals in monomial $C'$ are a subset of those in $C$.

Proof of Claim 3.19. For the forward direction, assume $C$ is an implicant of $\varphi$, and then use $(\star)$ to deduce that $\mathcal{M}_\varphi(\boldsymbol{x}) = 1$ for every instance $\boldsymbol{x}$ that is a completion of $\gamma^{-1}(C) = \boldsymbol{y}$. As $\boldsymbol{y} \subseteq \boldsymbol{x}$ by construction, we conclude $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$ under $\mathcal{M}_\varphi$. To see minimality we reason by contradiction. We assume, hoping to retract ourselves, that $\boldsymbol{z} \subsetneq \boldsymbol{y}$ is a sufficient reason for $\mathcal{M}_\varphi$, and we denote $C' = \gamma(\boldsymbol{z}) \subsetneq \gamma(\boldsymbol{y}) = C$. Then, because of $(\star)$, we have that $C'$ must be an implicant of $\varphi$, which contradicts the minimality of $C$ as an implicant of $\varphi$. For the backward direction, as we know that $\mathcal{M}(\boldsymbol{x}) = 1$, and by hypothesis $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$ wrt. $\mathcal{M}$, we have that every completion $\boldsymbol{x}'$ of $\boldsymbol{y} = \varphi^{-1}(C)$ holds $\mathcal{M}_\varphi(\boldsymbol{x}') = 1$, and thus we can use $(\star)$ to obtain that $C$ is an implicant of $\varphi$. To see minimality, assume seeking a contradiction that $C$ is not a prime implicant, but instead there is a monomial $C' \subsetneq C$ that is an implicant of $\varphi$. By using $(\star)$ again, we deduce that $\mathcal{M}(\boldsymbol{x}) = 1$ for every instance $\boldsymbol{x}$ that is a completion of $\gamma^{-1}(C') \subsetneq \gamma^{-1}(C) = \boldsymbol{y}$. But this means that $\gamma^{-1}(C')$ is a sufficient reason for $\boldsymbol{x}$ wrt. $\mathcal{M}$ and strictly contained in $\boldsymbol{y}$, which contradicts the minimality of $\boldsymbol{y}$. □

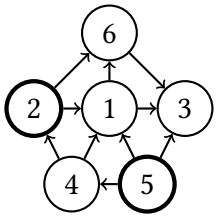
Both membership and hardness being proved, we conclude completeness for the class DP. □

### 3.3.4 The complexity of MinimumSufficientReason

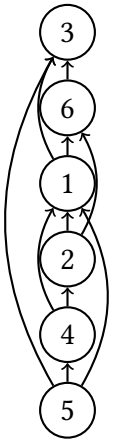
**Lemma 3.20** The MinimumSufficientReason query is NP-complete for FBDDs, and hardness holds already for decision trees.

Proof. Membership in NP is clear, it suffices to guess the instance $\boldsymbol{y}$ and check both $\mathrm{d}(\boldsymbol{x}, \boldsymbol{y}) \leq k$ and $\mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y})$. We will prove that hardness holds already for the particular case of decision trees, and when the input instance $\boldsymbol{x}$ is positive. Hardness of this particular setting implies of course the hardness of the general problem. In order to do so, we will reduce from the problem of determining whether a directed acyclic graph has a dominating set of size at most $k$, which we abbreviate as Dom-DAG. Recall that in a directed graph $G = (V, E)$, a subset of vertices $D \subseteq V$ is said to be dominating if every vertex in $V \setminus D$ has an incoming edge from a vertex in $D$. The problem of Dom-DAG is shown to be NP-complete in [34].

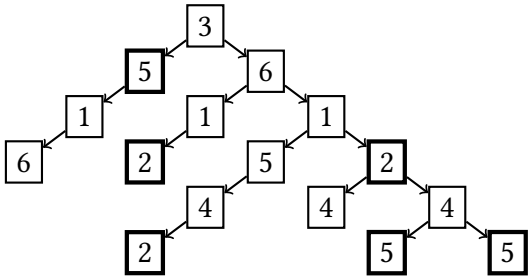
An illustration of the reduction is presented in Figure 3.5. Let $(G = (V, E), k)$ be an instance of Dom-DAG, and let us define $n := |V|$. We start by computing in polynomial time a topological ordering $\varphi = \varphi_1, \ldots, \varphi_n$ of $G$. Next, we will create an instance $(\mathcal{T}, \boldsymbol{x}, k)$ of $k$-SufficientReason such that there is a sufficient reason of size at most $k$ for $\boldsymbol{x}$ under the decision tree $\mathcal{T}$ if and only if $G$ has a dominating set of size at most $k$. We create the decision tree $\mathcal{T}$, of dimension $n$, in 2 steps.




(a) Example of an input DAG. Nodes 2 and 5, corresponding to the minimum dominating set of $G$ are emphasized.




(b) A topological ordering $\varphi$ of $G$.




(c) Resulting decision tree $\mathcal{T}$. Edges to the left of a node are always labeled with 0, and edges to the right with 1. The leaves are not depicted for clarity, but: if a node has no right child in the picture, then its right child is true, and if it has no left child then its left child is false. Note that in every diagonal there is an emphasized node, which is either 2 or 5, implying the partial instance $(\bot, 1, \bot, \bot, 1, \bot)$ is a sufficient reason for the instance $\boldsymbol{x} = (1, 1, 1, 1, 1, 1)$.

Figure 3.5: Illustration of the reduction from Dom-DAG to $k$-SufficientReason over decision trees, for an example graph of 6 nodes.

1. Create nodes $v_1, \ldots, v_n$, where node $v_\mathrm{i}$ is labeled with $\varphi_\mathrm{i}$ The node $v_n$ will be the root of $\mathcal{T}$, and for $2 \leq \mathrm{i} \leq n$, connect $v_\mathrm{i}$ to $v_{\mathrm{i}-1}$ with an edge labeled with 1. Node $v_1$ is connected to a leaf labeled true through an edge labeled with 1. We will denote the path created in this step as $\pi$.

2. For every vertex $\varphi_\mathrm{i}$ create a decision tree $\mathcal{T}_\mathrm{i}$ equivalent to the boolean formula

$$\mathcal{F}_\mathrm{i} = \bigvee_{(\varphi_j, \varphi_\mathrm{i}) \in E} \varphi_j$$

   and create an edge from $v_\mathrm{i}$ to the root of $\mathcal{T}_\mathrm{i}$ labeled with 0. If $\mathcal{F}_\mathrm{i}$ happens to be the empty formula, $\mathcal{T}_\mathrm{i}$ is defined as false. Note that the nodes introduced by this step are all naturally associated with vertices of $G$.

Step 2 takes polynomial time because boolean formulas in 1-DNF can easily be transformed into a decision tree in linear time.

We now check that $\mathcal{T}$ is a decision tree. Since $\mathcal{T}$ has a tree structure, it is enough to check that for every path from the root to a leaf there are no two nodes on the path that have the same label (i.e., to check that $\mathcal{T}$ is a valid FBDD). Note that any path from the root $v_n$ to a leaf goes first to a certain node $v_\mathrm{i}$ in $\pi$, from where it either takes an edge labeled with 0, in case $\mathrm{i} \neq 1$ or it simply goes to a leaf otherwise. In case $\mathrm{i} = 1$, the path from the root goes exactly through $v_n, v_{n-1}, \ldots, v_1$, which all have different labels. In case $\mathrm{i} \neq 1$, the path includes (i) nodes with labels $\varphi_n, \varphi_{n-1}, \ldots, \varphi_\mathrm{i}$, and (ii) a subpath inside $\mathcal{T}_\mathrm{i}$. It is clear that all the labels in (i) are different. And as by construction $\mathcal{T}_\mathrm{i}$ is a decision tree, no two nodes inside (ii) can have the same label. It remains to check that no node in (i) can have the same label of a node in (ii). To see this, consider that all the vertices of $G$ associated to the nodes in (ii) have edges to $\varphi_\mathrm{i}$ in $G$, and thus come before $\varphi_\mathrm{i}$ in the topological order. But (i) is composed precisely by $\varphi_\mathrm{i}$ and the nodes who come after it in the topological ordering, so (i) and (ii) have empty intersection.

Let $\boldsymbol{x} = 1^n$ be the vector of $n$ ones. We claim that $(\mathcal{T}, \boldsymbol{x}, k)$ is a yes-instance of $k$-SufficientReason if and only if $(G, k)$ is a yes-instance of Dom-DAG.

**Forward direction.** Consider that there is a sufficient reason $\boldsymbol{y}$ for $\boldsymbol{x}$ under $\mathcal{T}$ of size at most $k$. As $\boldsymbol{x}$ contains only 1s, $\boldsymbol{y}$ must contain only 1s and $\perp$s. Consider the set $S$ of components i where $y_\mathrm{i} = 1$. Recalling that every vertex of $G$ is canonically associated with a feature of $\mathcal{T}$, we will denote $D_S$ to the set of vertices of $G$ that are associated with the features in $S$. It is clear that $|D_S| \leq k$. We now prove that $D_S$ is a dominating set of $G$. First, in case $D_S = V$, we are trivially done. We know assume $D_S \neq V$. Consider a vertex $v \in V \setminus D_S$, corresponding to $\varphi_\mathrm{i}$ in the topological ordering, and define $\boldsymbol{z}$ as the completion of $\boldsymbol{y}$ where the features $\varphi_j$ such that $j > \mathrm{i}$, are set to 1, and all other features that are undefined by $\boldsymbol{y}$ are set to 0. By hypothesis, $\boldsymbol{z}$ must be a positive instance, and so its path on $\mathcal{T}$ must end in a leaf labeled with true. Note that the path of $\boldsymbol{z}$ in $\mathcal{T}$ necessarily takes the path $\pi$ created in Step 1 of the construction, up to the node $v_\mathrm{i}$, and then enters its subtree $\mathcal{T}_\mathrm{i}$. Let $t$ be the node of $\mathcal{T}_\mathrm{i}$ whose leaf labeled with true ends the path of $\boldsymbol{z}$ in $\mathcal{T}$, and $\varphi_k$ its label and associated vertex in $G$. As feature $t$ is set to 1, we must have either $\varphi_k \in D_S$ (in case $t$ is 1 because of $\boldsymbol{y}$) or $k > \mathrm{i}$ (in case $t$ is 1 by the construction of completion $\boldsymbol{z}$). However, the second case is not actually possible, as if $k > \mathrm{i}$, that means $v_k$ comes before $v_\mathrm{i}$ in path $\pi$, and thus the path of $\boldsymbol{z}$ in $\mathcal{T}$ passes through $v_k$, which has label $\varphi_k$ before passing through $v_\mathrm{i}$. But the path of $\boldsymbol{z}$ in $\mathcal{T}$ passes through $t$ before ending, which also has label $\varphi_k$. This is contradicts the already proven fact that $\mathcal{T}$ is a decision tree. We can therefore assume that $\varphi_k$ belongs to $D_S$. Then, as $t$ is a node of $\mathcal{T}_\mathrm{i}$, there must be an edge $(\varphi_k, \varphi_\mathrm{i})$ in $E$ because of the way $\mathcal{T}_\mathrm{i}$ is constructed. But this means that vertex $v \in V \setminus D_S$ has an edge coming from $\varphi_k \in D_S$, and so $v$ is effectively dominated by the set $D_S$. As this holds for every $v \in V \setminus D_S$, we conclude that $D_S$ is indeed a dominating set of $G$.

**Backward Direction.** Consider that there is a dominating set $D \subseteq V$ of size at most $k$. Let $S_D$ be the set of features associated with $D$. We claim that the partial instance $\boldsymbol{y}$ that has 1 in the features that belong to $S_D$, and is undefined elsewhere, is a sufficient reason for $\boldsymbol{x}$, and by construction its size is at most $k$. Consider an arbitrary completion $\boldsymbol{z}$ of $\boldsymbol{y}$, we need to show that $\boldsymbol{z}$ is a positive instance of $\mathcal{T}$. For $\boldsymbol{z}$ not to be a positive instance, its path on $\mathcal{T}$ would have to reach a leaf labeled with false. This can only happen by either taking the edge labeled wit h 0 from $v_1$ (the last node in path $\pi$ built in the construction), or inside a subtree $\mathcal{T}_\mathrm{i}$, corresponding to a node $v_\mathrm{i}$ whose associated feature in $\boldsymbol{z}$ is set to 0. We show that neither can happen. For the first case, every dominating set must include $\varphi_1$, the vertex in $G$ associated with $v_1$, as it is the first element in the topological ordering of $G$, and thus it must has in-degree 0, which implies $\varphi_1 \in D$. Therefore, it is not possible to take the edge labeled with 0 from $v_1$. On the other hand, suppose the path of $\boldsymbol{z}$ in $\mathcal{T}_\mathrm{i}$ ends in a leaf labeled with false. Then, by construction of $\mathcal{T}_\mathrm{i}$, there is no vertex $\varphi_j$ such that $(\varphi_j, \varphi_\mathrm{i}) \in E$ whose associated feature is set to 1 in $\boldsymbol{z}$. But as $D$ is a dominating set, either there is indeed a $\varphi_j \in D$ such that $(\varphi_j, \varphi_\mathrm{i}) \in E$ or $\varphi_\mathrm{i} \in D$. The first case is in direct contradiction with the previous statement, as $\varphi_j \in D$ implies, by our construction of $\boldsymbol{y}$ that the feature associated with $\varphi_j$ is set to 1. The second case also creates a contradiction, as if $\varphi_\mathrm{i} \in D$, then by construction $\boldsymbol{y}$ would have a 1 in the feature $v_\mathrm{i}$ associated to $\varphi_\mathrm{i}$, which contradicts the assumption of the path of $\boldsymbol{z}$ entering $\mathcal{T}_\mathrm{i}$. □

**Lemma 3.21** The MinimumSufficientReason query can be solved in linear time for perceptrons.

Proof. Let $(\mathcal{M} = (\boldsymbol{w}, b), \boldsymbol{x}, k)$ be an instance of the problem, and let $\mathrm{d}$ be the dimension of the perceptron. We will assume without loss of generality that $\mathcal{M}(\boldsymbol{x}) = 1$. In this proof, what we call a *minimum sufficient reason for* $\boldsymbol{x}$ is a sufficient reason for $\boldsymbol{x}$ that has the least number of components being defined. We show a greedy algorithm that computes a minimum sufficient reason for $\boldsymbol{x}$ under $\mathcal{M}$ in time $O(|\mathcal{M}|)$. For each feature $\mathrm{i}$ of $\boldsymbol{x}$ we define its importance $s(\mathrm{i})$ as $w_{\mathrm{i}}$ if $x_{\mathrm{i}} = 1$ and $-w_{\mathrm{i}}$ otherwise (just as we did in the proof of Lemma 3.10), and its *penalty* $p(\mathrm{i})$ as $\min(0, w_{\mathrm{i}})$. Intuitively, $s$ represents how good it is for a partial instance to be defined in a given feature, and $p$ represents the penalty or cost that a partial instance incurs by not being defined in a given feature. We now assume that $\boldsymbol{x}$ and $\boldsymbol{w}$ have been sorted in decreasing order of score $s$. For example, if originally $\boldsymbol{w} = (3, -5, -2)$ and $\boldsymbol{x} = (1, 0, 1)$, then after the sorting procedure we have $\boldsymbol{w} = (-5, 3, -2)$ and $\boldsymbol{x} = (0, 1, 1)$. We now define a function $\psi$ that takes any partial instance $\boldsymbol{y}$ as input and outputs the worst possible value for a completion of $\boldsymbol{y}$:

$$\psi(\boldsymbol{y}) \coloneqq \min_{\boldsymbol{z}:\, \boldsymbol{z} \text{ is a completion of } \boldsymbol{y}} \langle \boldsymbol{w}, \boldsymbol{z} \rangle = \sum_{y_{\mathrm{i}} \neq \perp} w_{\mathrm{i}} y_{\mathrm{i}} + \sum_{y_{\mathrm{i}} = \perp} p(\mathrm{i}).$$

The second equality is easy to see based on the definition of the function $p$, and the definition of $\psi$ implies that $\psi(\boldsymbol{y}) \geq -b$ exactly when $\boldsymbol{y}$ is a sufficient reason. For $1 \leq l \leq \mathrm{d}$, we define $\boldsymbol{y}^l$ as the partial instance of $\boldsymbol{x}$ such that $y^l_{\mathrm{i}}$ is equal to $x_{\mathrm{i}}$ if $\mathrm{i} \leq l$ and to $\perp$ otherwise. In simple terms, $\boldsymbol{y}^l$ is the partial instance obtained by taking the first $l$ features of $\boldsymbol{x}$; continuing our example with $\boldsymbol{x} = (0, 1, 1)$, we have for instance $\boldsymbol{y}^2 = (0, 1, \perp)$. Let $j$ be the minimum index such that $\psi(\boldsymbol{y}^j) \geq -b$. Such an index always exists, because, since $\boldsymbol{x}$ is a positive instance, taking $j = \mathrm{d}$ is always a valid index. Note that $j$ can be computed in linear time.

We now prove that (†) the partial instance $\boldsymbol{y}^j$ is a minimum sufficient reason for $\boldsymbol{x}$. By definition we have that $\psi(\boldsymbol{y}^j) \geq -b$, so $\boldsymbol{y}^j$ is indeed a sufficient reason for $\boldsymbol{x}$. We now need to show that $\boldsymbol{y}^j$ is minimum. Assume, seeking for a contradiction, that there is a sufficient reason $\boldsymbol{y}'$ of $\boldsymbol{x}$ with strictly less components defined than $\boldsymbol{y}^j$; clearly we can assume without loss of generality that $\boldsymbol{y}'$ has exactly $j - 1$ components defined. We will now show that (⋆) $\boldsymbol{y}^{j-1}$ is a also a sufficient reason for $\boldsymbol{x}$, which is a contradiction since $j$ was assumed to be the minimal index such that $\boldsymbol{y}^j$ is a sufficient reason of $\boldsymbol{x}$, hence proving (†). If $\boldsymbol{y}' = \boldsymbol{y}^{j-1}$, we have that (⋆) is trivially true. Otherwise, and considering that $\boldsymbol{y}'$ and $\boldsymbol{y}^{j-1}$ have the same size, and that $\boldsymbol{y}^{j-1}$ is defined exactly on the first $j - 1$ features, there must be at least a pair of features $(m, n)$, with $m \leq j - 1 < n$, such that $\boldsymbol{y}^{j-1}$ is defined at feature $m$ and $\boldsymbol{y}'$ is not, and on the other hand $\boldsymbol{y}'$ is defined at feature $n$ whereas $\boldsymbol{y}^{j-1}$ is not. In order to finish the proof of (⋆), we will prove a simpler claim that will help us conclude.

**Claim 3.22** Assume that there is a pair of features $(m, n)$ with $m \leq j - 1 < n$ such that $y'_m = \perp, y^{j-1}_m \neq \perp$ and $y'_n \neq \perp, y^{j-1}_n = \perp$, and let $\boldsymbol{y}^*$ be the resulting partial instance that is equal to $\boldsymbol{y}'$ except that $y^*_m \coloneqq y^{j-1}_m$ and $y^*_n \coloneqq \perp$. Then we have that $\psi(\boldsymbol{y}^*) \geq \psi(\boldsymbol{y}')$.

Proof of Claim 3.22. By definition, $\psi(\boldsymbol{y}^*) - \psi(\boldsymbol{y}') = p(n) - p(m) + w_m y^{j-1}_m - w_n y'_n = p(n) - p(m) + w_m x_m - w_n x_n$. But because the features in $y^{j-1}$ are sorted in decreasing order of score, it must hold that $s(m) \geq s(n)$. Using this last inequality and reasoning by cases on the values $x_m, x_n$ and on the signs of $w_m, w_n$, one can tediously check that $\psi(\boldsymbol{y}^*) - \psi(\boldsymbol{y}') \geq 0$ and thus $\psi(\boldsymbol{y}^*) \geq \psi(\boldsymbol{y}')$. □

We now continue with the proof of ($\star$). As a result of Claim 3.22, one can iteratively modify $\boldsymbol{y}'$ until it becomes equal to $\boldsymbol{y}^{j-1}$ in such a way that the value of $\psi$ is never decreased along the process, implying therefore that $\psi(\boldsymbol{y}^{j-1}) \geq \psi(\boldsymbol{y}')$. But $\psi(\boldsymbol{y}') \geq -b$, because $\boldsymbol{y}'$ is assumed to be a sufficient reason, hence we have that $\psi(\boldsymbol{y}^{j-1}) \geq -b$, implying that $\boldsymbol{y}^{j-1}$ is a sufficient reason for $\boldsymbol{x}$, and thus concluding the proof of ($\star$). Therefore, (†) is proven, and since $\boldsymbol{y}^j$ can clearly be computed in polynomial time (in fact, the runtime of the whole procedure is dominated by the sorting subroutine, which again has cost $O(|\mathcal{M}|)$ as it is a classical problem of sorting strings whose total length add up to $\mathcal{M}$ and can be carried with a variant of Bucketsort [12]), this finishes the proof of the lemma; indeed, we can output YES if $j \leq k$ and NO otherwise. □

**Lemma 3.23** The MINIMUMSUFFICIENTREASON query is $\Sigma_2^p$-complete for MLPs.

PROOF. Membership in $\Sigma_2^p$ is clear, as one can non-deterministically guess the value of the $k$ features that would make for a sufficient reason, and then use an oracle in co-NP to verify that no completion of that guess has a different classification. To show hardness, we will reduce from the problem SHORTEST IMPLICANT CORE, defined and proven to be $\Sigma_2^p$-hard in [55, Theorem 1]. First, we need a few definitions in order to present this problem. A formula in *disjunctive normal form* (DNF) is a Boolean formula of the form $\varphi = t_1 \vee t_2 \vee \ldots \vee t_n$, where each *term* $t_i$ is a conjunction of literals (a literal being a variable of the negation thereof). An *implicant* for $\phi$ is a partial assignment of the variables of $\phi$ such that any extension to a full assignment makes the formula evaluate to true; note that we can equivalently see a implicant of $\phi$ as a what we call in this work a sufficient reason of $\phi$. For a partial assignment $C$ of the variables and for a set (or conjunction) of literals $t$, we write $C \subseteq t$ when for every variable $x$, if $x \in t$ then $C(x) = 1$ and if $\neg x \in t$ then $C(x) = 0$ and $C(x)$ is undefined otherwise. An instance of SHORTEST IMPLICANT CORE then consists of a DNF formula $\varphi = t_1 \vee t_2 \vee \ldots \vee t_n$, together with an integer $k$. Such an instance is positive for SHORTEST IMPLICANT CORE when there is an implicant $C$ for $\varphi$ such that $C \subseteq t_n$.[3]

**A reduction that does not work, and how to fix it on an example.** In order to convey the main intuition, we start by presenting a first tentative of a reduction that does not work. Thanks to Lemma 3.9 we know that it is possible to build an MLP $\mathcal{M}_\varphi$ equivalent to $\varphi$. However, doing so directly creates a problem: we would need to find a convenient instance $\boldsymbol{x}$ such that $(\varphi, k) \in$ SHORTEST IMPLICANT CORE if and only if $(\mathcal{M}_\varphi, \boldsymbol{x}, k) \in k$-SUFFICIENTREASON. A natural idea is to consider $t_n$ as a candidate for $\boldsymbol{x}$, but the issue is that $t_n$ does not necessarily include every variable. The next natural idea is to try with $\boldsymbol{x}$ being an arbitrary completion of $t_n$ (interpreting $t_n$ as the partial instance that is uniquely defined by its satisfying assignment). This approach fails because there could be a sufficient reason of size at most $k$ for such a $\boldsymbol{x}$ that relies on features (variables) that are not in $t_n$. We illustrate this with an example for $n = 4$.

$$\varphi := x_1\overline{x_5} \vee \overline{x_2}\,\overline{x_6} \vee x_3x_6 \vee \overline{x_1}\,\overline{x_2}x_4 \vee \underbrace{x_1x_3x_5}_{t_4}$$

While it can be checked that $(\varphi, 2) \notin$ SHORTEST IMPLICANT CORE, we have that the instance $(\mathcal{M}_\varphi, (1,0,1,0,1,1), 2)$ is in fact a positive instance of $k$-SUFFICIENTREASON, as the partial instance that assigns 1 to $x_3$ and $x_6$ and is undefined for the rest of the features, is a sufficient reason of size 2 for $\boldsymbol{x}$. The issue is that we are allowing $x_6$ to be part of the sufficient reason for $\boldsymbol{x}$ even

[3]Note that, in order to keep our notation consistent, we use the symbol $\subseteq$ where Umans uses $\supseteq$.

though $x_6 \notin t_4$. We can avoid this from happening by splitting each variable that is not in $t_n$, such as $x_6$, into $k+1$ variables, in such a way that defining the value of $x_6$ would force us to define the value of all the $k+1$ variables, which is of course unaffordable. Continuing with the example, we build the formula $\varphi'$ as follows:

$$\varphi' \coloneqq \bigwedge_{\mathrm{i}=1}^{3} \left( x_1 \overline{x_5} \vee \overline{x_2^{\mathrm{i}}}\,\overline{x_6^{\mathrm{i}}} \vee x_3 x_6^{\mathrm{i}} \vee \overline{x_1}\,\overline{x_2^{\mathrm{i}}} x_4^{\mathrm{i}} \vee x_1 x_3 x_5 \right)$$

Now we can simply take $(\mathcal{M}_{\varphi'}, \boldsymbol{x}, 1)$ where $\boldsymbol{x}$ is an arbitrary completion of $t_4$ over the new set of variables, for example, one that assigns 1 to the features $1, 3$ and $5$, and $0$ to all other features (variables). Note that $\varphi'$ is not a DNF anymore, but this is not a problem, since we only need to compute $\mathcal{M}_{\varphi'}$. It is then easy to check that this instance is equivalent to the original input instance.

**The reduction.** We now present the correct reduction and prove that it works. Let $(\varphi, k)$ be an instance of Shortest Implicant Core. Let $X_c$ be the set of variables that are not mentioned in $t_n$. We split every variable $x_j \in X_c$ into $k+1$ variables $x_j^1, \ldots x_j^{k+1}$ and for each $\mathrm{i} \in \{1, \ldots, k+1\}$ we build $\varphi^{(\mathrm{i})}$ by replacing every occurrence of a variable $x_j$, that belongs to $X_c$, by $x_j^{\mathrm{i}}$. Finally we define $\varphi'$ as the conjunction of all the $\varphi^{(\mathrm{i})}$. That is,

$$\varphi^{(\mathrm{i})} \coloneqq \varphi[x_j \to x_j^{\mathrm{i}}, \text{ for all } x_j \in X_c] \tag{3.2}$$

$$\varphi' \coloneqq \bigwedge_{\mathrm{i}=1}^{k+1} \varphi^{(\mathrm{i})} \tag{3.3}$$

Observe that any meaningful instance of Shortest Implicant Core has $k < |t_n|$, so we can safely assume that $k$ is given in unary, making this construction polynomial.

We then use Lemma 3.9 to build an MLP $\mathcal{M}_{\varphi'}$ from $\varphi'$, in polynomial time. The features of this model correspond naturally to the variables of $\varphi'$, and thus we refer to both features and variables without distinction. Let $\boldsymbol{y}$ be the instance that assigns 1 to every variable that appears as a positive literal in $t_n$, and 0 to all other variables. We claim that $(\varphi, k) \in$ Shortest Implicant Core if and only if $(\mathcal{M}_{\varphi'}, \boldsymbol{x}, k) \in k$-SufficientReason. For the forward direction, if there is an implicant $C \subseteq t_n$ of $\varphi$, of size at most $k$, then we claim that $C$ is also an implicant of each $\varphi^{(\mathrm{i})}$. This follows from the fact that every assignment $\sigma$ that is consistent with $C$ and satisfies $\varphi$, has a related assignment $\sigma^{\mathrm{i}}$, that for every variable $x_j \in X_c$ assigns $\sigma^{\mathrm{i}}(x_j^{\mathrm{i}}) = \sigma(x_j)$, and that is equal to $\sigma$ for every $x_j \notin X_c$. It is clear that $\sigma^{\mathrm{i}}(\varphi^{(\mathrm{i})}) = \sigma(\varphi)$, which concludes the claim. As $C$ is an implicant of each $\varphi^{(\mathrm{i})}$, it must also be an implicant of $\varphi'$. Then, as $\mathcal{M}_{\varphi'}$ is equivalent to $\varphi'$ (as Boolean functions) by construction, and $\boldsymbol{x}$ is consistent with $C$ because it is consistent with $t_n$, it follows that the partial instance that is induced by $C$ is a sufficient reason for $\boldsymbol{x}$ under $\mathcal{M}_{\varphi'}$. For the backward direction, assume there is a sufficient reason $\boldsymbol{y}$ for $\boldsymbol{x}$ under $\mathcal{M}_{\varphi'}$, whose size is at most $k$, and let $C'$ be its associated implicant for $\varphi'$. We cannot say yet that $C'$ is a proper candidate for being an implicant core of $\varphi$, as $C'$ could contain variables not mentioned by $t_n$.

Let us define $X'_c$ to be the set of variables of $\varphi'$ that are not present in $t_n$. Intuitively, as there are $k+1$ copies of each variable of $X'_c$ in $\varphi'$, no valuation of a variable in $X'_c$, for the formula $\varphi$, can be forced by a sufficient reason of size at most $k$. We will prove this idea in the following claim, allowing us to build an implicant $C$ for which we can assure $C \subseteq t_n$.

**Claim 3.24** Assume that there is an implicant $C'$ of size at most $k$ for $\varphi'$, and let $C$ be the partial valuation that sets every variable $x \in t_n \cap C'$ to $C'(x)$, and that leaves every other variable undefined. Then $C'$ is an implicant of size at most $k$ for $\varphi$.

Proof of Claim 3.24. The set $X'_c$ can be expressed as the union of $k+1$ disjoint sets of variables, namely $X_c^1, \ldots, X_c^{k+1}$, where $X_c^{\mathrm{i}}$ contains all variables of the form $x_j^{\mathrm{i}}$. Since $C'$ contains at most $k$ literals, and there are $k+1$ disjoint sets $X_c^{\mathrm{i}}$, there must exist an index $l$ such that $X_c^l \cap C' = \varnothing$. But then this implies that $C$ is an implicant of $\varphi^{(l)}$. But $\varphi^{(l)}$ is equivalent to $\varphi$ up to renaming of the variables that are not present in $C$, therefore, the fact that $C$ is an implicant of $\varphi^{(l)}$ implies that $C$ must be an implicant of $\varphi$ as well. □

By using Claim 3.24 we get that $C$ is an implicant of $\varphi$. But we have that $C \subseteq t_n$, which is enough to conclude that $(\varphi, k) \in$ Shortest Implicant Core and finishes the proof of Lemma 3.23. □

### 3.3.5 The complexity of CountCompletions

What follows is our main complexity result regarding the query CountCompletions, which yields Theorems 3.1, 3.2, and 3.3 for the case of CC.

**Theorem 3.25** The query CountCompletions is (1) in PTIME for FBDDs, (2) #P-complete for perceptrons, and (3) #P-complete for MLPs.

The first claim follows almost directly from the definition of FBDDs, see [56]. For the second claim, we will rely on the #P-hardness of the counting problem #Knapsack, as defined next:

**Definition 3.26** An input of the problem #Knapsack is simply a group of natural numbers $s_1, \ldots, s_n, k \in \mathbb{N}$. The output is the number of subsets $S \subseteq \{1, \ldots, n\}$ such that $\sum_{\mathrm{i} \in S} s_{\mathrm{i}} \leq k$.

The problem #Knapsack is well known to be #P-complete. Since we were not able to find a proper reference for this fact, we prove it here by using the #P-hardness of the problem #Subset-Sum. An input of the problem #SubsetSum consists of natural numbers $s_1, \ldots, s_n, k \in \mathbb{N}$, and the output is the number of subsets $S \subseteq \{1, \ldots, n\}$ such that $\sum_{\mathrm{i} \in S} s_{\mathrm{i}} = k$. The problem #SubsetSum is shown to be #P-complete in [8, Theorem 4]. From this we can deduce:

**Lemma 3.27** (Folklore) The problem #Knapsack is #P-complete.

Proof. Membership in #P is trivial. We prove hardness by polynomial-time reduction from #Sub-

setSum. Let $(s_1, \ldots, s_n, k) \in \mathbb{N}^{n+1}$ be an input to #SubsetSum. It is clear that #SubsetSum$(s_1, \ldots, s_n, 0) =$ #Knapsack$(s_1, \ldots, s_n, 0)$, and that for $k \geq 1$ we have #SubsetSum$(s_1, \ldots, s_n, k) =$ #Knapsack$(s_1, \ldots, s_n, k) -$ #Knapsack$(s_1, \ldots, s_n, k-1)$, thus establishing the reduction. □

We can now show the second claim of Theorem 3.25.

**Lemma 3.28** The query CountCompletions is #P-complete for perceptrons.

Proof. Membership in #P is trivial. We show hardness by polynomial-time reduction from #Knapsack. Let $(s_1, \ldots, s_n, k)$ be an input of #Knapsack. Let $\mathcal{M}$ be the perceptron with weights $s_1, \ldots, s_n$ and bias $-(k+1)$. Remember that we consider only perceptrons that use the step activation function, so that an instance $\boldsymbol{x} \in \{0,1\}^n$ is positive for $\mathcal{M}$ if and only if $\sum_{i=1}^{n} x_i s_i - (k+1) \geq 0$. It is then clear that #Knapsack$(s_1, \ldots, s_n, k) = 2^n -$ CountPositiveCompletions$(\mathcal{M}, \bot^n)$, thus establishing the reduction. □

Finally, the third claim of Theorem 3.25 simply comes from the fact that MLPs can simulate arbitrary Boolean formulas (Lemma 3.9), and the fact that counting the number of satisfying assignments of a Boolean formula (#SAT) is #P-complete.

Although the query CountCompletions is #P-complete for perceptrons, we can still show that the complexity goes down to PTIME if we assume the weights and biases to be integers given in unary; this is commonly called *pseudo-polynomial time*.

**Theorem 3.29** The query CountCompletions can be solved in pseudo-polynomial time for perceptrons (assuming the weights and biases to be integers given in unary).

The first part of the proof is to show how to transform in polynomial time and arbitrary instance of CountPositiveCompletions for perceptrons (with the weights and bias being integers given in unary) into an instance of #Knapsack that has the same number of solutions.

**Lemma 3.30** Let $\mathcal{M} = (\boldsymbol{w}, b)$ be a perceptron having at least one positive instance, with the weights and bias being integers given in unary, and let $\boldsymbol{x}$ be a partial instance. We can build in polynomial time an input $(s_1, \ldots, s_m, k) \in \mathbb{N}^{m+1}$ of #Knapsack such that CountPositiveCompletions$(\mathcal{M}, \boldsymbol{x}) =$ #Knapsack$(s_1, \ldots, s_m, k)$, with $s_1, \ldots, s_m, k$ written in unary (i.e., their value is polynomial in the input size).

Proof. The first step is to get rid of the components that are defined by $\boldsymbol{x}$, as follows. We define:

- $A \coloneqq \sum_{x_i \neq \bot} x_i w_i$;
- $\boldsymbol{w}' \coloneqq (w_i \mid x_i = \bot)$; and
- $b' \coloneqq b + A$;

and let $\mathcal{M}'$ be the perceptron $(\boldsymbol{w}', b')$. Notice that the dimension of $\mathcal{M}'$ is equal to the number of undefined components of $\boldsymbol{x}$; let us write $m$ this number. It is then clear that CountPositiveCompletions($\mathcal{M}$, is equal to the number of positive instances of $\mathcal{M}'$, that is, of instances $\boldsymbol{x}' \in \{0,1\}^m$ that satisfy

$$\langle \boldsymbol{w}', \boldsymbol{x}' \rangle + b' \geq 0 \tag{3.4}$$

Now, let $J$ be the maximum possible value of $\langle \boldsymbol{w}', \boldsymbol{x}' \rangle$; $J$ can clearly be computed in linear time by setting $x'_{\mathrm{i}} = 1$ if $w'_{\mathrm{i}} \geq 0$ and $x'_{\mathrm{i}} = 0$ otherwise. We then claim that the number of solutions to Equation 3.4 is equal to the number of solutions of

$$\langle \boldsymbol{s}, \boldsymbol{x}' \rangle \leq k, \tag{3.5}$$

where $s_{\mathrm{i}} := |w'_{\mathrm{i}}|$ for $1 \leq \mathrm{i} \leq m$ and $k := J + b'$. Indeed, consider the function $h : \{0,1\}^m \to \{0,1\}^m$ defined componentwise by $h(x'_{\mathrm{i}}) := x'_{\mathrm{i}}$ if $w'_{\mathrm{i}} < 0$ and $h(x'_{\mathrm{i}}) := 1 - x'_{\mathrm{i}}$ otherwise. Then $h$ is a bijection, and we will show that for any $\boldsymbol{x}' \in \{0,1\}^m$, we have that $\boldsymbol{x}'$ satisfies Equation 3.4 if and only if $h(\boldsymbol{x}')$ satisfies Equation 3.5, from which our claim follows. In order to see this, consider that

$$(3) \iff \sum_{\mathrm{i}} w'_{\mathrm{i}} x'_{\mathrm{i}} \geq -b' \iff \sum_{w_{\mathrm{i}} \geq 0} w'_{\mathrm{i}} x'_{\mathrm{i}} + \sum_{w_{\mathrm{i}} < 0} w'_{\mathrm{i}} x'_{\mathrm{i}} \geq -b' \tag{3.6}$$

$$\iff \sum_{w_{\mathrm{i}} \geq 0} |w'_{\mathrm{i}}| x'_{\mathrm{i}} - \sum_{w_{\mathrm{i}} < 0} |w'_{\mathrm{i}}| x'_{\mathrm{i}} \geq -b' \tag{3.7}$$

$$\iff \sum_{w_{\mathrm{i}} < 0} |w'_{\mathrm{i}}| x'_{\mathrm{i}} - \sum_{w_{\mathrm{i}} \geq 0} |w'_{\mathrm{i}}| x'_{\mathrm{i}} \leq b' \tag{3.8}$$

$$\tag{3.9}$$

On the other hand, we have

$$h(\boldsymbol{x}') \text{ satisfies } (4) \iff \sum_{\mathrm{i}} |w'_{\mathrm{i}}| h(x'_{\mathrm{i}}) \leq J + b' \tag{3.10}$$

$$\iff \sum_{w_{\mathrm{i}} < 0} |w'_{\mathrm{i}}| x'_{\mathrm{i}} + \sum_{w_{\mathrm{i}} \geq 0} |w'_{\mathrm{i}}| (1 - x'_{\mathrm{i}}) \leq \sum_{w_{\mathrm{i}} \geq 0} |w'_{\mathrm{i}}| + b' \tag{3.11}$$

$$\iff (7) \tag{3.12}$$

Last, let us observe that we have $k \geq 0$, as otherwise $\mathcal{M}$ would not have any positive instance. Therefore $(s_1, \ldots, s_m, k)$ is a valid input of #Knapsack, which concludes the proof. □

We can now easily combine Lemma 3.30 together with a well-known dynamic programming algorithm solving #Knaspsack in pseudo-polynomial time.

Proof of Theorem 3.29. Let $\mathcal{M} = (\boldsymbol{w}, b)$ be a perceptron, with the weights and bias being integers given in unary, and let $\boldsymbol{x}$ be a partial instance. First, we check that the maximal value of $\langle \boldsymbol{x}, \boldsymbol{w} \rangle$ is greater than $-b$, as otherwise $\mathcal{M}$ has no positive instance and we can simply return 0. We then use Lemma 3.30 to build in polynomial time an instance $(s_1, \ldots, s_m, k) \in \mathbb{N}^{m+1}$

of #Knapsack such that CountPositiveCompletions$(\mathcal{M}, \boldsymbol{x})$ = #Knapsack$(s_1, \ldots, s_m, k)$, and with $s_1, \ldots, s_m, k$ being written in unary (i.e., their value is polynomial in the input size). We can then compute #Knapsack$(s_1, \ldots, s_m, k)$ by dynamic programming as follows. For i $\in \{1, \ldots, m\}$ and $C \in \mathbb{N}$, define the quantity DP[i][$C$] := $|\{S \subseteq \{1, .., \text{i}\} | \sum_{j \in S} s_j \leq C\}|$. We wish to compute DP[$m$][$k$]. We can do so by computing DP[i][$C$] for i $\in \{1, \ldots, m\}$ and $C \in \{0, \ldots, k\}$, using the relation DP[i + 1][$C$] = DP[i][$C$] + DP[i][$C - s_{\text{i}+1}$], and starting with the convention that DP[0][$a$] = 0 for all $a < 0$ and that DP[0][$a$] = 1 for all $a \geq 0$. It is clear that the whole procedure can be done in polynomial time. □

Theorem 3.29 result establishes a difference between perceptrons and MLPs in terms of CC, as this query remains #P-complete for the latter even if weights and biases are given as integers in unary. Another difference is established by the fact that CountCompletions for perceptrons can be efficiently approximated, while this is not the case for MLPs. To present this idea, we briefly recall the notion of *fully polynomial randomized approximation scheme* (FPRAS [32]), which is heavily used to refine the analysis of the complexity of #P-hard problems. Intuitively, an FPRAS is a polynomial time algorithm that computes with high probability a $(1 - \epsilon)$-multiplicative approximation of the exact solution, for $\epsilon > 0$, in polynomial time in the size of the input and in the parameter $1/\epsilon$. We show:

**Theorem 3.31** The problem CountCompletions restricted to perceptrons admits an FPRAS (and the use of randomness is not even needed in this case). This is not the case for MLPs, on the other hand, at least under standard complexity assumptions.

The fact that the query has no FPRAS for MLPs is because MLPs can efficiently simulate Boolean formulas (Lemma 3.9), and it is well known that the problem #SAT (of counting the number of satisfying assignments of a Boolean formula) has no FPRAS unless NP = RP. Hence we only need to prove our claim concerning perceptrons.

Proof of Theorem 3.31 for perceptrons. We can assume without loss of generality that the weights and bias are integers, as we can simply multiply every rational by the lowest common denominator (note that the bit lenght of the lowest common denominator is polynomial, and that it can be computed in polynomial time[4]). We then transform the perceptron and partial instance to an input of #Knapsack with the right number of solutions using Lemma 3.30, by observing that the construction also takes polynomial time when the input weights are given in binary (and by considering that the $s_1, \ldots, s_m, k$ are also computed in binary). We can then apply an FPTAS to this #Knapsack instance, as shown in [27, 47]. □

[4]We need to compute the least common multiple (lcm) of a set of integers $a_1, \ldots, a_n$. Indeed, it is easy to check that $lcm(a_1, \ldots, a_n) = lcm(lcm(a_1, \ldots, a_{n-1}), a_n)$, which reduces inductively the problem to computing the lcm of two numbers in polynomial time. It is also easy to check that $lcm(a_1, a_2) = \frac{a_1 a_2}{gcd(a_1, a_2)}$, where $gcd(a_1, a_2)$ is the greatest common divisor of $a_1$ and $a_2$. As multiplication can clearly be carried in polynomial time, and Euclid's algorithm allows to compute the $gcd$ function in polynomial time, we are done.

# Chapter 4

# The parameterized complexity of interpreting Neural Networks: deeper is harder than shallow

In Section 3.3.1 we proved that for MLPs the MinimumChangeRequired query is in NP, and that this holds no matter the number of layers the MLPs have. Moreover, a careful inspection of our proofs reveals that MCR is already NP-complete for MLPs with only a few layers[1]. This is not something specific to MCR: in fact, all lower bounds for the queries studied in the paper in terms of MLPs hold for a small, fixed number of layers. Henceforth, we cannot differentiate the interpretability of shallow and deep MLPs with the classical complexity classes that we have used so far.

In this section, we show how to construct a gap between the (complexity-based) interpretability of shallow and deep MLPs by considering refined complexity classes in our $c$-interpretability framework. In particular, we use *parameterized complexity* [17, 21], a branch of complexity theory that studies the difficulty of a problem in terms of multiple input parameters. We introduce its main underlying idea in terms of two classical graph problems: VertexCover and Clique. In both problems the input is a pair $(G, k)$ with $G$ a graph and $k$ an integer. In VertexCover we verify if there is a set of nodes of size at most $k$ that includes at least one endpoint for every edge in $G$. In Clique we check if there is a set of nodes of size at most $k$ such that all nodes in the set are adjacent to each other. Both problems are known to be NP-complete. However, this analysis treats $G$ and $k$ at the same level, which might not be fair in some practical situations in which $k$ is much smaller than the size of $G$. Parameterized complexity then studies how the complexity of the problems behaves when the input is only $G$, and $k$ is regarded as a small *parameter*. It happens to be the case that VertexCover and Clique, while both NP-complete, have a different status in terms of parameterized complexity. Indeed, VertexCover can be solved in time $O(2^k \cdot |G|)$, which is polynomial in the size of the input $G$ – with the exponent not depending on $k$ – and, thus, it is called *fixed-parameter tractable* [17]. In turn, it is widely believed that there is no algorithm for Clique with time complexity $O(f(k) \cdot \text{poly}(G))$ – with $f$ being

[1]The presented proof implies hardness for 3 layers, and it is not hard to see that the anti-monotone formula for independent set can be implemented with 2 layers.

any computable function, that depends only on $k$ – and thus it is *fixed-parameter intractable* [17]. To study the notion of fixed-parameter intractability, researchers on parameterized complexity have introduced the $W[t]$ complexity classes (with $t \geq 1$), which form the so called $W$*-hierarchy*. For instance Clique is $W[1]$-complete [17]. A main assumption in parameterized complexity is that $W[t] \subsetneq W[t+1]$, for every $t \geq 1$. We now proceed to explain the required bits of parameterized complexity theory that are needed to prove our results.

## 4.1 Required background in parameterized complexity

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma$ is a finite alphabet. For each element $(x, k)$ of a parameterized problem, the second component is called the *parameter* of the problem. A parameterized problem is said to be *fixed parameter tractable* (FPT) if the question of whether $(x, k)$ belongs to $L$ can be decided in time $f(k) \cdot |x|^{O(1)}$, where $f$ is a computable function.

The FPT class, as well as the other classes we will introduce in this paper, are closed under a particular kind of reductions. A mapping $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ between instances of a parameterized problem $A$ to instances of a parameterized problem $B$ is said to be an *fpt-reduction* if and only if

- $(x, k)$ is a yes-instance of $A \iff \phi(x, k)$ is a yes-instance of $B$.
- $\phi(x, k)$ can be computed in time $|x|^{O(1)} \cdot f(k)$;
- There exists a computable function $g$ such that $k' \leq g(k)$, where $k'$ is the parameter of $\phi(x)$.

We will define the complexity classes that are relevant for this article in terms of circuits. Recall that a circuit is a rooted directed acyclic graph[2] where nodes of in-degree 0 are called *input gates*, and that the root of the circuit is called the *output gate*. Internal gates can be either Or, And, or Not gates. All Not nodes have in-degree 1. Nodes of types And, Or can either have in-degree at most 2, in which case they are said to be *small* gates[3], or in-degree bigger than 2, in which case they are said to be *large* gates. The *depth* of a circuit is defined as the length (number of edges) of the longest path from any input node to the output node. The *weft* of a circuit is defined as the maximum amount of large gates in any path from an input node to the output node. An *assignment* of a circuit $C$ is a function from the set of input gates in $C$ to $\{0, 1\}$. The weight of an assignment is defined as the number of input gates that are assigned 1. Assignments of a circuit naturally induce a value for each gate of the circuit, computed according to the label of the gate. We say an assignment *satisfies* a circuit if the value of the output gate is 1 under that assignment.

The main classes we will be dealing with are those composing the $W$-hierarchy and the $W(Maj)$-hierarchy, a variant proposed in [19] by Fellows et al. These complexity classes can be defined upon the Weighted Circuit Satisfiability problem, parameterized by specific classes $\mathcal{C}$ of circuits, as defined below.

[2]Note that under this definition multiple parallel edges are allowed. Although this is not useful for circuits with (Or, And, Not)-gates, it allows circuits with more complex gates, such as *majority gates* (that are satisfied when more than half its inputs are 1) to receive multiple times a same input.

[3]The And, Or gates with in-degree 1 compute the unary identity function.

| | |
|---|---|
| Problem: | WEIGHTED CIRCUIT SATISFIABILITY($\mathcal{C}$), abbreviated WCS($\mathcal{C}$) |
| Input: | A circuit $C \in \mathcal{C}$ |
| Parameter: | An integer $k$ |
| Output: | YES, if there is a satisfying assignment of weight exactly $k$ for $C$, and NO otherwise. |

We consider two restricted classes of circuits. First, $C_{t,\mathrm{d}}$, the class of circuits using the connectives AND, OR, NOT that have weft at most $t$ and depth at most $\mathrm{d}$. On the other hand, we consider $M_{t,\mathrm{d}}$, the class of circuits that use the majority connective MAJ (that is satisfied exactly when more than half of its inputs are true), have weft at most $t$ and depth at most $\mathrm{d}$. Note that in the case of majority gates, a gate is said to be small if its fan-in is at most 3.

We can then define each class $W[t]$ as the set of problems that can be fpt-reduced to WCS$(C_{t,\mathrm{d}})$ for some constant $\mathrm{d}$. Similarly, each class $W(Maj)[t]$ corresponds to the set of problems that can be fpt-reduced to WCS$(M_{t,\mathrm{d}})$ for some constant $\mathrm{d}$.

Note that the notion of *can be fpt-reduced* is transitive, and thus the classes $W[t]$ and $W(\mathsf{Maj})[t]$ are closed under fpt-reductions. Showing that a parameterized problem $A$ is $W[t]$-hard (resp., $W(\mathsf{Maj})[t]$-hard) is usually complicated, as using this definition one would have to show that, for every fixed $\mathrm{d} \in \mathbb{N}$, there exists an fpt-reduction $f_{\mathrm{d}}$ from WCS$(C_{t,\mathrm{d}})$ (resp., from WCS$(M_{t,\mathrm{d}})$) to $A$. Instead, it is usually more convenient to prove first some form of *normalization theorem* stating that a particular class of circuits, for which one knows the value of $\mathrm{d}$, is already hard for $W[t]$ (or $W(\mathsf{Maj})[t]$). Useful normalization theorems for the $W$-hierarchy are proved in the work of Downey et al. [16, 18], or Buss et al. [11]. We provide a loose normalization theorem for the $W(\mathsf{Maj})$-hierarchy in Lemma 4.4, that is used to derive hardness.

## 4.2 Statement of a parameterized result and sketch of proof

Given an MLP $\mathcal{M}$, with the dimension of the layers being $\mathrm{d}_0, \ldots, \mathrm{d}_k$, we define its *graph size* as $N := \sum_{\mathrm{i}=0}^{k} \mathrm{d}_\mathrm{i}$. We say an MLP with graph size $N$ is restricted (abbreviated as rMLP) if each of its weights and biases can be represented as a decimal number with at most $\log(N)$ digits. More precisely, represented as $\sum_{\mathrm{i}=-K}^{K} a_\mathrm{i} 10^\mathrm{i}$, for integers $0 \leq a_\mathrm{i} \leq 9$ and $K \in O(\log N)$. Note that all numbers expressible in this way are also expressible by fractions, where the numerator is an arbitrary integer bounded by a polynomial in $N$, and the denominator is a power of 10 whose value is bounded as well by a polynomial in $N$.

We now explicit the parameterized problem, or more precisely, a family of parameterized problems indexed by an integer $t \geq 1$.

| | |
|---|---|
| Problem: | $t$-MINIMUMCHANGEREQUIRED, abbreviated $t$-MCR |
| Input: | An rMLP $\mathcal{M}$ with at most $t$ layers, an instance $\boldsymbol{x}$ |
| Parameter: | An integer $k$ |
| Output: | YES, if there exists an instance $\boldsymbol{y}$ with $\mathrm{d}(\boldsymbol{x}, \boldsymbol{y}) \leq k$ and $\mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y})$, and NO otherwise |

We now have enough notation to explicit our results.

**Theorem 4.1** For every $t \geq 1$, the $(3t+3)$-MCR problem is $W(\mathsf{Maj})[t]$-hard and is contained in $W(\mathsf{Maj})[3t+7]$.

As the proof of Theorem 4.1 is quite involved, we first present a sketch proof that summarizes the process.

**Hardness.** We start by showing in Lemma 4.4 that the problem $WCS(M_{3t+2,3t+3})$ is $W(\mathsf{Maj})[t]$-hard; the main difficulty here is to reduce the depth $\mathrm{d}$ of the majority circuits, for any fixed $\mathrm{d} \in \mathbb{N}$, to a depth of at most $3t+3$. We then show in Lemma 4.2 that rMLPs can simulate majority circuits, without increasing the depth of the circuit. In Theorem 4.5 we use this construction to show an fpt-reduction from $WCS(M_{3t+2,3t+3})$ to $(3t+3)$-MCR. This is enough to conclude hardness for $W(\mathsf{Maj})[t]$.

**Membership.** Presented in Theorem 4.7, the proof consists of 4 steps. We first show in Lemma 4.3 how to transform a given rMLP $\mathcal{M}$ that into an MLP $\mathcal{M}'$ that uses only step activation functions and that has the same number of layers. Then, as a second step, we build an MLP $\mathcal{M}''$, with $3t+4$ layers and again using only the step activation function, such that $\mathcal{M}''$ has a satisfying assignment of weight $k$ if and only if $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of the $t$-MCR problem. The third step is to use a result of circuit complexity [24] stating that circuits with weighted thresholds gates (which are equivalent to biased step functions), can be transformed into circuits using only majority gates, increasing the depth by no more than 1. This yields a circuit $C_{\mathcal{M}''}$ with $3t+5$ layers. However, the circuit $C_{\mathcal{M}''}$, resulting from the construction of Goldmann et al. [24], has both positive variables and negated variables as inputs, as their model needs to be able to represent non-monotone functions. For the fourth and last step, we build a circuit $C^*_{\mathcal{M}''}$ based on $C_{\mathcal{M}''}$, that fits the description of majority circuits as defined by [19, 20] (i.e., the one that we use). This circuit $C^*_{\mathcal{M}''}$ has weft $3t+7$, and we prove that $(C^*_{\mathcal{M}''}, k+1)$ is a positive instance of the Weighted Circuit Satisfiability problem that characterizes the class $W(\mathsf{Maj})[t]$ if and only if $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of the $(3t+3)$-MCR problem. The whole construction being an fpt-reduction, this will be enough to conclude membership in $W(\textsc{Maj})[3t+7]$.

### 4.2.1 The graph interpretation of (r)MLPs

We remark here that (r)MLPs can be interpreted as well as rooted directed acyclic graphs, with weighted edges and where each node is associated a layer according to its (unweighted) distance from the root. Every node in a certain layer $\ell$ is connected to every node in layers $\ell-1$ and $\ell+1$. This equivalent interpretation turns out to be more handy for some of the proofs in this section.

## 4.3 Constructions and transformations

In this subsection we present the relevant constructions and transformations between the different kind of circuits and MLPs that are involved in the proof of Theorem 4.1.

We first prove that a circuit containing only majority gates can be simulated by an rMLP.

**Lemma 4.2** Given a circuit $C$ containing only majority gates, we can build in polynomial time an rMLP that is equivalent to $C$ (as a Boolean function) and whose number of layers is equal to

the depth of $C$.

Proof. First, note that we can assume that circuit $C$ does not contain parallel edges by replacing each gate $g$ having $p$ edges to a gate $g'$ by $p$ copies $g_1, \ldots, g_p$ with single edges to $g'$. We then build a layerized circuit (remember the definition of a layerized circuit from Lemma 3.9) $C'$ from $C$, by applying the same construction that we used in Lemma 3.9 to layerize a circuit, but using unary majority gates as identity gates instead. Note that the depth of $C'$ is the same as that of $C$.

Next, we show how each non-output majority gate can be simulated by using two $\mathrm{relu}$-gates (again, remember the definition of a relu gate from Lemma 3.9). First, note that (†) for any non-negative integers $x, n \in \mathbb{N}$, the function

$$f_n(x) \coloneqq \mathrm{relu}(x - \lfloor \frac{n}{2} \rfloor) - \mathrm{relu}(x - \lfloor \frac{n}{2} \rfloor - 1)$$

is equal to

$$\mathsf{Maj}_n(x) = \begin{cases} 1 & \text{if } x > \frac{n}{2} \\ 0 & \text{otherwise} \end{cases}.$$

We will use (†) to transform the majority circuit $C'$ into a circuit $C''$ that has only relu gates for the non-output gates, and that is equivalent to $C'$ in a sense that we will explain next. For every non-output majority gate $g$ of $C'$, we create two relu gates $g'_1, g'_2$ of $C''$. The idea is that (⋆) for any valuation of the input gates (we identify the input gates of $C'$ with those of $C''$), the Boolean value of any non-output gate $g$ in $C'$ will be equal to the (not necessarily Boolean) value of gate $g'_1$ (in $C''$) minus the value of the gate $g'_1$ (in $C''$). We now explain what are the biases of these new gates $g'_1, g'_2$ for every majority gate $g$ of $C'$. Letting $n$ be the in-degree of a majority gate $g$ in $C'$, the bias of $g'_1$ is $-\lfloor \frac{n}{2} \rfloor$, and that of $g'_2$ is $-\lfloor \frac{n}{2} \rfloor - 1$. Next, we explain what are the weights of these new gates $g'_1, g'_2$ and how we connect them to the other relu gates. We do this by a bottom-up induction on $C'$, that is, on the level of the gates of $C'$ (since $C'$ is layerized), and we will at the same time show that (⋆) is satisfied. To connect the gates $g'_1, g'_2$ to the gates of the preceding layer, we differentiate two cases:

**Base case.** The inputs of the gate $g$ are variable gates; in other words, the level of $g$ in $C'$ is $1$ (remember that variable gates have level $0$). We then set these variable gates to be an input of both $g'_2$ and $g'_2$, and set all the weights to $1$. It is clear that (⋆) is satisfied for the gates $g, g'_1, g'_2$, thanks to (†).

**Inductive case.** The inputs of the gate $g$ are other majority gates; in other words, the level of $g$ in $C'$ is $> 1$. Then, let ${}^1g, \ldots, {}^m g$ be the inputs[4] (majority gates) of the gate $g$ in $C'$, and consider their associated pairs of relu gates $({}^1g'_1, {}^1g'_2), \ldots, ({}^m g'_1, {}^m g'_2)$ in $C''$. We then set all the gates ${}^1g'_1, \ldots, {}^m g_1$ to be input gates of both gates $g'_1$ and $g'_2$, with a weight of $1$, and set all the gates ${}^1g_2, \ldots, {}^m g_2$ to be input gates of both gates $g'_1$ and $g'_2$, with a weight of $-1$. By induction hypothesis, and using again (†), it is clear that (⋆) is satisfied.

Finally, based on the output gate $r$ of $C'$, we create a step gate $r'$ in $C''$ in the following way. Let ${}^1g, \ldots, {}^m g$ be the inputs of $r$, and $({}^1g'_1, {}^1g'_2), \ldots, ({}^m g'_1, {}^m g'_2)$ their associated pairs in $C''$. Then wire each gate ${}^{\mathrm{i}}g'_1$ to $r'$ with weight $1$, and also wire each gate ${}^{\mathrm{i}}g'_2$ to $r'$ with weight $-1$. Let $-\lfloor \frac{n}{2} \rfloor - 1$ be the bias of $r'$.

[4] We hope that the reader will excuse us for using left superscripts.

We have constructed a circuit $C''$ whose output gate is a step gate, and all other gates are relu gates. Consider now a valuation $\boldsymbol{x}$ of the input gates of $C'$, which we identify as well as a valuation $\boldsymbol{x}'$ of the input gates of $C''$. We claim that $C'(\boldsymbol{x}) = 1$ if and only if $C''(\boldsymbol{x}') = 1$. But this simply comes from the fact that for $x, n \in \mathbb{N}$, we have $x > \frac{n}{2} \iff x \geq \lfloor \frac{n}{2} \rfloor + 1$, and from the fact that $(\star)$ is satisfied for the input gates of $r$ and of $r'$.

The last thing that we have to do is to transform the circuit $C''$, that uses only relu gates except for its output step gate, into a valid MLP. This can be done easily as in the proof of Lemma 3.9 by adding dummy connections with weights zero, because $C''$ is layerized. The resulting MLP $\mathcal{M}_C$ is then equivalent to $C$, it is clearly an rMLP, its number of layers is exactly the depth of $C$, and, since we have constructed it in polynomial time, this concludes the proof.

□

We now prove that MLPs using only step activation functions are powerful enough to simulate MLPs that use $\mathrm{relu}$ activation functions in the internal layers (and a step function for the output neuron), if we assume all weights and biases to be rationals and representable with a bounded number of bits. The construction is polynomial on the width (maximal number of neurons in a layer) of the given $\mathrm{relu}$-MLP, but exponential on its depth (number of layers). We show:

**Lemma 4.3** Given an rMLP $\mathcal{M}$ with $\mathrm{relu}$ activation functions, there is an equivalent rMLP $\mathcal{M}'$ that uses only step activation functions. Moreover, if the number of layers of $\mathcal{M}$ is bounded by a constant, then $\mathcal{M}'$ can be computed in polynomial time.

Proof. Let $(\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(\ell)})$, $(\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(\ell)})$ and $(f^{(1)}, \ldots, f^{(\ell)})$ be the sequences of weights, biases, and activation functions of the rMLP $\mathcal{M}$. Note that $f^{(\mathrm{i})}$ for $1 \leq \mathrm{i} \leq \ell - 1$ is $\mathrm{relu}$ and that $f^{(\ell)}$ is the step activation function. The first step of the proof is to transform every weight and bias into an integer. To this end, let $L \in \mathbb{N}$, $L > 0$ be the lowest common denominator of all the weights and biases, and let $\mathcal{M}'$ be the MLP that is exactly equal to $\mathcal{M}$ except that all the weights have been multiplied by $L$, and all the biases of layer $\mathrm{i}$ have been multiplied by $L^{\mathrm{i}}$. Observe that $\mathcal{M}'$ has only integer weights and biases. When $w$ (resp., $b$) is a weight (resp., bias) of $\mathcal{M}$, we write $w'$ (resp., $b'$) the corresponding value in $\mathcal{M}'$. We claim that $\mathcal{M}$ and $\mathcal{M}'$ are equivalent, in the sense that for every $\boldsymbol{x} \in \{0,1\}^n$, it holds that $\mathcal{M}(\boldsymbol{x}) = \mathcal{M}'(\boldsymbol{x})$. Indeed, for $0 \leq \mathrm{i} \leq \ell$, let $\boldsymbol{h}^{(\mathrm{i})}$ and $\boldsymbol{h}'^{(\mathrm{i})}$ be the vectors of values for the layers of $\mathcal{M}$ and $\mathcal{M}'$, respectively, as defined by Equation 3.1. We will show that $(\star)$ for all $1 \leq \mathrm{i} \leq \ell - 1$ we have $\boldsymbol{h}'^{(\mathrm{i})} = L^{\mathrm{i}} \times \boldsymbol{h}^{(\mathrm{i})}$. The base case of $\mathrm{i} = 0$ (i.e., the inputs) is trivially true. For the inductive case, assume that $(\star)$ holds up to $\mathrm{i}$ and let us show that it holds for $\mathrm{i}+1$. We have:

$$
\begin{aligned}
\boldsymbol{h}'^{(\mathrm{i}+1)} &= \mathrm{relu}(\boldsymbol{h}'^{(\mathrm{i})}\boldsymbol{W}'^{(\mathrm{i}+1)} + \boldsymbol{b}'^{(\mathrm{i}+1)}) \\
&= \mathrm{relu}(L \times \boldsymbol{h}'^{(\mathrm{i})}\boldsymbol{W}^{(\mathrm{i}+1)} + L^{\mathrm{i}+1} \times \boldsymbol{b}^{(\mathrm{i}+1)}) \text{ by the definition of } \mathcal{M}' \\
&= \mathrm{relu}(L^{\mathrm{i}+1} \times \boldsymbol{h}^{(\mathrm{i})}\boldsymbol{W}^{(\mathrm{i}+1)} + L^{\mathrm{i}+1} \times \boldsymbol{b}^{(\mathrm{i}+1)}) \text{ by inductive hypothesis} \\
&= L^{\mathrm{i}+1} \times \mathrm{relu}(\boldsymbol{h}^{(\mathrm{i})}\boldsymbol{W}^{(\mathrm{i}+1)} + \boldsymbol{b}^{(\mathrm{i}+1)}) \text{ by the linearity of } \mathrm{relu} \\
&= L^{\mathrm{i}+1} \times \boldsymbol{h}^{(\mathrm{i}+1)},
\end{aligned}
$$

and ($\star$) is proven. Since the step function (used for the output neuron) satisfies $\mathrm{step}(cx) = c\,\mathrm{step}(x)$ for $c > 0$, we indeed have that $\mathcal{M}(\boldsymbol{x}) = \mathcal{M}'(\boldsymbol{x})$.

We now show how to build a model $\mathcal{M}''$ that uses only step activation functions and that is equivalent to $\mathcal{M}'$. The first step is to prove an upper bound for the values in $\boldsymbol{h}'$. We start by bounding the values in $\boldsymbol{h}$. Let $D$ be width of $\mathcal{M}$, that is, the maximal dimension of a layer of $\mathcal{M}$, and let $C$ be the maximal absolute value of a weight or bias in $\mathcal{M}$; note that the value of $C$ is asymptotically bounded by $|\mathcal{M}|^{O(1)}$ because $\mathcal{M}$ is an rMLP. For every instance $\boldsymbol{x}$, we have that

$$0 \leq \boldsymbol{h}_j^{(\mathrm{i})} = \mathrm{relu}(\sum_k \boldsymbol{h}_k^{(\mathrm{i}-1)} \boldsymbol{W}_{k,j}^{(\mathrm{i})} + \boldsymbol{b}_j^{(\mathrm{i})}) \leq DC \max_k \boldsymbol{h}_k^{(\mathrm{i}-1)} + C \leq (D+1)C \max(1, \max_k \boldsymbol{h}_k^{(\mathrm{i}-1)})$$

Using this inequality, and the fact that $\max_k \boldsymbol{h}_k^{(0)} \leq 1$, we obtain inductively that $0 \leq \boldsymbol{h}_j^{(\mathrm{i})} \leq ((D+1)C)^{\mathrm{i}}$. By ($\star$), this implies that $0 \leq \boldsymbol{h}_j'^{(\mathrm{i})} \leq ((D+1)CL)^{\mathrm{i}}$.

As all values (weights, biases and the $\boldsymbol{h}'$ vectors) in $\mathcal{M}'$ consist only of integers, and are all bounded by the integer $S \coloneqq ((D+1)CL)^{\ell}$, then each relu in $\mathcal{M}'$ with bias $b$ becomes equivalent to the following function $f^*$:

$$f^*(x+b) \coloneqq [x+b \geq 1] + [x+b \geq 2] + \ldots + [x+b \geq S] \tag{4.1}$$

Where $[y \geq j] \coloneqq 1$ if $y \geq j$ and 0 otherwise. Hence, in order to finish the proof, it is enough to show how activation functions of the form $f^*$ can be simulated with step activation functions. Namely, we show how to build $\mathcal{M}''$, that uses only step activation functions, from $\mathcal{M}'$, in such a way that both models are equivalent. In order to do so, we replace each $f^{(\mathrm{i})}, \boldsymbol{W}'^{(\mathrm{i})}, \boldsymbol{b}'^{(\mathrm{i})}$ for $1 \leq \mathrm{i} \leq \ell$ in the following way. If $\mathrm{i} = \ell$, then nothing needs to be done, as $f^{(\ell)}$ is already assumed to be a step activation function. When $1 \leq \mathrm{i} < \ell$, we replace the weights, activations and biases in a way that is better described in terms of the underlying graph of the MLP. We split every internal node, with bias $b$ into $S$ copies, all of which will have the same incoming and outgoing edges as the original nodes, with the same weights. The $j$-th copy will have a bias equal to $b - j$. We illustrated this step in Figure 4.1. This construction is an exact simulation of the function $f^*$ defined in Equation 4.1.

The computationally expensive part of the algorithm is the replacement of each node in $\mathcal{M}'$ by $S$ nodes, which takes time at most $S = ((D+1)CL)^{\ell} \in O(|\mathcal{M}|^{\ell}(CL)^{\ell})$ per node and thus at most $O(|\mathcal{M}|^{\ell+1}(CL)^{\ell})$ in total. Since $\ell$ is a constant, and $C$ is bounded by a polynomial on $\mathcal{M}$, we only need to argue that $L$ is bounded as well. Indeed, as $\mathcal{M}$ is an rMLP, each weight and bias can be assumed to be represented as a fraction whose denominator is a power of 10 of value polynomial in the graph size $N$ of $\mathcal{M}$. But the lowest common multiple of a set of powers of 10 is exactly the largest power of 10 in the set. Therefore $L \leq 10^p$, where $p \in O(\log N)$, and thus $L \in O(N^c) \subseteq O(|\mathcal{M}|^c)$ for some constant $c$. We conclude from this that the construction takes polynomial time. □

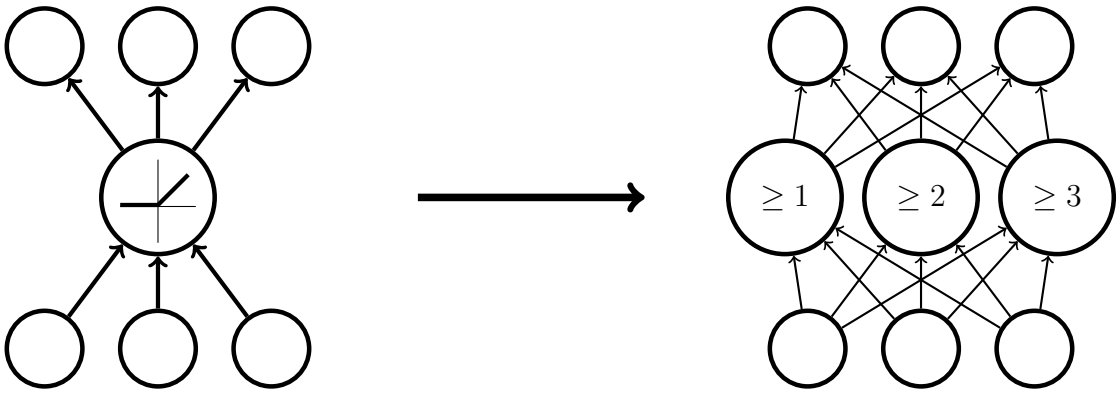



Figure 4.1: Illustration of the conversion from a relu activation function to step activation functions, for $S = 3$. The weights are unchanged, and if the bias of the original neuron was $b$ then the bias in the $j$-th copy of that neuron becomes $b - j$.

## 4.4 A (*perhaps way too long*) proof for Theorem 4.1

Based on the definitions presented in Section 4.1, the only hard problem for the class $W(\mathsf{Maj})[t]$ we know of is that of $WCS(M_{t,\mathrm{d}})$. A priori, the constant $\mathrm{d}$ required to get hard instances could be arbitrarily high, and so we start by proving a modest normalization theorem, that states that $\mathrm{d} = 3t + 3$ is enough to have hard instances for the class $W(\mathsf{Maj})[t]$.

**Lemma 4.4** (Normalization) The problem $WCS(M_{3t+2,3t+3})$ is $W(\mathsf{Maj})[t]$-hard.

Proof. A significant part of this proof is based on techniques due to Fellows et al. [19] and to Buss et al. [11]. Let $C$ be an arbitrary majority circuit of weft at most $t$ and depth at most $\mathrm{d} \geq t$ for some constant $\mathrm{d}$, and let $k$ be the parameter of the input instance. We define a *small sub-circuit* as a maximally connected sub-circuit comprising only small gates. Now, consider a path $\pi$ from an arbitrary input node of $C$ to its output gate. We claim that $\pi$ intersects at most $t + 1$ small sub-circuits. Indeed, there must be at least one large gate separating every pair of small sub-circuits intersected by $\pi$, as otherwise the maximality assumption would be broken. But in $\pi$, as in any path, there are at most $t$ large gates, because of the weft restriction, from where we conclude the claim. Now, for each small sub-circuit $S$, consider the set $I_S$ of its inputs (that may be either large gates or input nodes of $C$). As small gates have fan-in at most 3, and the depth of each small sub-circuit is at most $\mathrm{d}$, we have that $|I_S| \leq 3^{\mathrm{d}}$. We can thus enumerate in constant time all the satisfying assignments of $S$. We identify each assignment with the set of variables to which it assigns the value 1. We keep a set $\Gamma$ with the satisfying assignments among $I_S$ that are minimal with respect to $\subseteq$. Then, because of the fact that majority circuits are monotone, $S$ can be written in monotone DNF as

$$S \equiv \bigvee_{\gamma \in \Gamma} \bigwedge_{x \in \gamma} x$$

Note that the size of $\Gamma$ is trivially bounded by the constant $2^{3^{\mathrm{d}}}$. We then build a circuit $C'$, based on $C$, by following these steps:

1. Add $3^{\mathrm{d}}(k + 1)$ extra input nodes. We distinguish the first, that we denote as $u$, from

the $3^{\mathrm{d}}(k+1)-1$ remaining, that we refer to by $N$.

2. Add a new output gate that is a binary majority between the old output gate and the node $u$.

3. Replace every small sub-circuit $S$ by its equivalent monotone DNF formula, consisting of one large $\vee$-gate and many large $\wedge$-gates.

4. Relabel every large $\vee$-gate, of fan-in $\ell \leq 2^{3^{\mathrm{d}}}$ created in the previous step to be a majority gate with the same inputs, but to which one wires as well $\ell$ parallel edges from the input node $u$.

5. Relabel every large $\wedge$-gate $g$, of fan-in $\ell \leq 3^{\mathrm{d}}$, to be a majority gate. If $g$ had edges from gates $g_1, \ldots, g_\ell$, then replace each edge coming from a $g_{\mathrm{i}}$ by $k+1$ parallel edges, and finally, wire $\ell(k+1)-1$ nodes in $N$ to $g$.

An illustration of the transformation ins presented in Figure 4.2. We now check that $C'$ is a (majority) circuit in $M_{3t+2,3t+3}$. To bound the depth and weft of $C'$ we need to account for all the sub-circuits of depth 2 that we introduced in steps 3–5 to replace each small sub-circuit of $C$. Note that two small sub-circuits that were parallel in $C$ (meaning no input-output path could intersect both) have corresponding sub-circuits that are parallel in $C'$. Consider now an arbitrary path $\pi$ from a variable to the root of $C$, and let $\pi'$ be the corresponding path in $C'$ (that goes to the new root of $C'$). The path $\pi$ contains one variable gate, at most $t$ large gates, and intersects at most $t+1$ small sub-circuits. The corresponding path $\pi'$ in $C'$ still contains the variable gate, the (at most $t$) large gates that were in $\pi$, and for each of the $\leq t+1$ small-subcircuits that $\pi$ intersected, $\pi'$ now contains exactly 2 large gate (and $\pi'$ also contains the new output gate of $C'$). Therefore, the length of $\pi'$ is at most $1+t+2(t+1)+1-1 = 3t+3$, and it contains at most $t+2(t+1) = 3t+2$ large gates. Since every path $\pi'$ in $C'$ from a variable to the root of $C'$ corresponds to such a path $\pi$ in $C$, we obtain that the depth of $C'$ is at most $3t+3$ and its weft is at most $3t+2$. Hence, $C'$ is indeed a majority circuit in $M_{3t+2,3t+3}$.

We now prove that ($\star$) there is a satisfying assignment of weight $k+1$ for $C'$ if and only if there is a satisfying assignment of weight $k$ for $C$, which would conclude our fpt-reduction. The proof for this claim is based on how the constructions in step 4 and 5 actually simulate large $\vee$-gates and $\wedge$-gates, respectively.[5] We prove each direction in turn.

**Forward direction.** Let us assume that there exists a satisfying assignment of weight $k+1$ for $C'$. First, because input node $u$ is directly connected to the output gate through a binary majority, it must be assigned to 1 in order to satisfy $C'$. Let $C''$ be the sub-circuit of $C'$ formed by all the nodes that descend from the old output-gate in $C'$. Then $C''$ needs to be satisfied in order to satisfy $C'$. Since $u$ is not present in $C''$, an assignment of weight $k+1$ that satisfies $C'$ is made by assigning 1 to $u$ and to exactly $k$ other input gates. In order to prove the claim, we will show that (†) an assignment of weight $k$ for the inputs of $C''$ satisfies $C''$ if and only if its restriction to the inputs of $C$ satisfies $C$, assuming $u$ is assigned to 1. As $C''$ only differs from $C$ because of the replacement of each small sub-circuit $S$ by its equivalent DNF, and the additional inputs in $N$, we only need to prove that steps 4 and 5 actually compute large $\vee$ and $\wedge$ gates. Consider a gate $g$

[5] Although this technique can already be found in the work of Fellows et al. [19], we include it here for completeness.

introduced in step 4, having edges from gates $g_1, \ldots, g_\ell$ and $\ell$ edges from node $u$. Therefore, $g$ has fan-in $2\ell$, and as $u$ always contributes with a value of $\ell$ to $g$, we have that $g$ is satisfied exactly when at least one of the gates $g_1, \ldots, g_\ell$ is satisfied. Consider now a gate $g$ introduced in step 5. By construction, $g$ has fan-in equal to $2\ell(k+1)-1$, from which we deduce that if all gates $g_1, \ldots, g_\ell$ are satisfied, then $g$ is indeed satisfied in $C''$. On the other hand, if an assignment of weight $k$ does not satisfy every gate $g_\mathrm{i}$, then $g$ receives at most $(\ell-1)(k+1)$ units from the gates $g_\mathrm{i}$, and as the assignment has weight $k$, it receives at most $k$ from the nodes in $N$. Thus, $g$ receives at most $(k+1)\ell - 1$ units, which is less than half of its fan-in, and thus, $g$ is not satisfied. Thus, we have proved (†). However, notice that the restriction of the assignment might have a weight of strictly less than $k$ in $C$. But it is clear that, since the circuit is monotone, we can increase the weight by setting to 1 some variables of $C$, until the weight becomes equal to $k$. This proves the forward direction.

**Backward direction.** Let us now assume an assignment of weight $k$ for $C$. We then we extend such an assignment to $C'$ by assigning 0 to the inputs in $N$ and 1 to $u$. Thanks to (†), this is a satisfying assignment of weight $k+1$ for $C'$, which proves the backward direction of ($\star$) and thus concludes the proof of Lemma 4.4. □

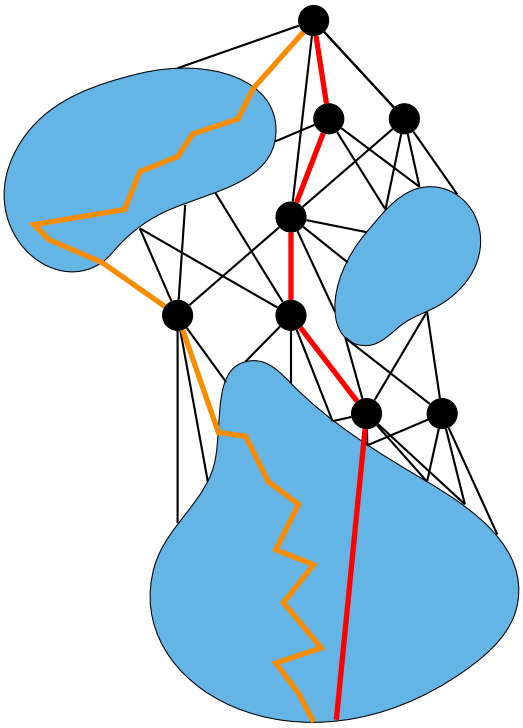
We now prove a reduction from the problem $WCS(M_{3t+2,3t+3})$ to the $(3t+3)$-MCR problem, implying that the latter is $W(\mathsf{Maj})[t]$ hard.

**Theorem 4.5** (Hardness) There is an fpt-reduction from the problem $WCS(M_{3t+2,3t+3})$ to the $(3t+3)$-MCR problem.

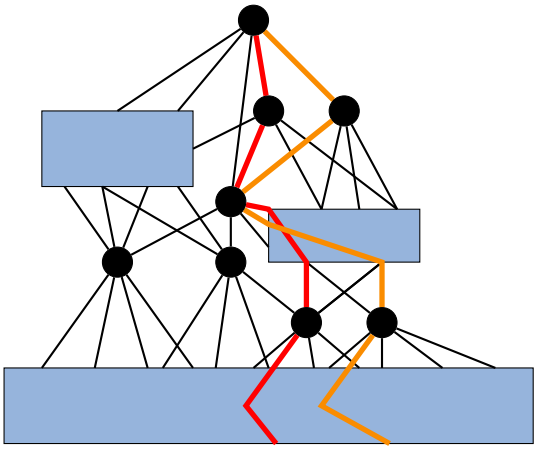
Proof. We will in fact show an fpt-reduction from $WCS(M_{t,t})$ to $t$-MCR, which gives the claim when applied to $3t+3$, noting of course that $WCS(M_{3t+3,3t+3})$ is trivially at least as hard as $WCS(M_{3t+2,3t+3})$. Let $(C,k)$ be an instance of $WCS(M_{t,t})$. We first build an MLP $\mathcal{M}_C$ equivalent to $C$ (as Boolean functions) by using Lemma 4.2. The MLP $\mathcal{M}_C$ has $t$ layers. Then, we build an MLP $\mathcal{M}'_C$, that is based on $\mathcal{M}_C$, by following the steps described below:

1. Initialize $\mathcal{M}'_C$ to be an exact copy of $\mathcal{M}_C$.
2. Add an extra input, that we call $v_1$, to $\mathcal{M}'_C$. This means that if $\mathcal{M}_C$ had dimension $n$, then $\mathcal{M}'_C$ has dimension $n+1$.
3. Create nodes $v_2, \ldots, v_t$, all having a bias of 0, and for each $1 \leq \mathrm{i} < t$, connect node $v_\mathrm{i}$ to node $v_{\mathrm{i}+1}$ with an edge of weight 1.
4. Let $r$ be the root of $\mathcal{M}'_C$, and let $m$ be its fan-in. We connect node $v_t$ to $r$ with an edge of weight $m$. Moreover, if the bias of $r$ in $\mathcal{M}_C$ was $b$, we set it to be $b-m$ in $\mathcal{M}'_C$.
5. Observe that $\mathcal{M}'_C$ is layerized. To make it a valid MLP (where all the neurons of a layer are connected to all the neurons of the adjacent layers), we do as in the proof of Lemma 3.9 by adding dummy null weights.

It is clear that the construction of $\mathcal{M}'_C$ takes polynomial time, and that its number of layers is



(a) A majority circuit where *small sub-circuits* are represented with blue blobs, and black nodes correspond to large majority gates. The path determining the *weft* is colored red. The longest path, determining the *depth* of the circuit, is colored orange.



(b) The majority circuit where *small sub-circuits* have been replaced by depth-2 majority circuits, corresponding to their equivalent DNF. The equivalent DNF depth-2 sub-circuits are represented by rectangles. Once again, the path determining the *weft* is colored red. The longest path, determining the *depth* of the circuit, is colored orange.

Figure 4.2: Illustration of the Normalization Lemma (4.4). In a nutshell, by paying a controlled increase in weft, the depth of the circuit can be substantially reduced.

again $t$. We now prove a claim describing the behavior of $\mathcal{M}'_C$.

**Claim 4.6** For any instance $\boldsymbol{x}'$ of $\mathcal{M}'_C$, expressed as the concatenation of a feature $\boldsymbol{x}'_1$ (for the extra input node $v_1$) and an instance $\boldsymbol{x}$ of $\mathcal{M}_C$, we have that $\boldsymbol{x}'$ is a positive instance of $\mathcal{M}'_C$ if and only if $\boldsymbol{x}'_1 = 1$ and $\boldsymbol{x}$ is a positive instance of $\mathcal{M}_C$

Proof. Consider that, by construction, an instance $\boldsymbol{x}'$ is positive for $\mathcal{M}'_C$ if and only if

$$\sum_{\mathrm{i}=1}^{n+1} \boldsymbol{h}_{\mathrm{i}}^{\prime(t-1)} \boldsymbol{W}_{\mathrm{i}}^{\prime(t)} = m\boldsymbol{h}_1^{\prime(t-1)} + \sum_{\mathrm{i}=2}^{n+1} \boldsymbol{h}_{\mathrm{i}}^{\prime(t-1)} \boldsymbol{W}_{\mathrm{i}}^{\prime(t)} \geq -b + m$$

But by construction $\boldsymbol{h}_1^{\prime(t-1)} = \boldsymbol{x}'_1$, and $\sum_{\mathrm{i}=2}^{m+1} \boldsymbol{h}_{\mathrm{i}}^{\prime(t-1)} \boldsymbol{W}_{\mathrm{i}}^{\prime(t)} = \sum_{\mathrm{i}=1}^{m} \boldsymbol{h}_{\mathrm{i}}^{(t-1)} \boldsymbol{W}_{\mathrm{i}}^{(t)}$. This means that $\boldsymbol{x}'$ is a positive instance of $\mathcal{M}'_C$ if and only if

$$m\boldsymbol{x}'_1 + \sum_{\mathrm{i}=1}^{m} \boldsymbol{h}_{\mathrm{i}}^{(t-1)} \boldsymbol{W}_{\mathrm{i}}^{(t)} \geq -b + m$$

Note that if $\boldsymbol{x}'_1 = 1$ and $\boldsymbol{x}$ is a positive instance of $\mathcal{M}_C$, this inequality is achieved, making $\boldsymbol{x}'$ a positive instance. For the other direction, it is clear that it holds if $\boldsymbol{x}'_1 = 1$. We show that in fact $\boldsymbol{x}'_1 = 0$ is not possible. Indeed, by the construction of $\mathcal{M}_C$, we have that $0 \leq \sum_{\mathrm{i}=1}^{m} \boldsymbol{h}_{\mathrm{i}}^{(t-1)} \boldsymbol{W}_{\mathrm{i}}^{(t)} \leq m$, and also that $-b \geq 1$, which makes the inequality unfeasible.

This concludes the proof of the claim. □

This claim has two important consequences:

1. As satisfying assignments of $C$ correspond to positive instance of $\mathcal{M}_C$, we have that there is a satisfying assignment of weight exactly $k$ for $C$ if and only if there is a positive instance of weight exactly $k+1$ for $\mathcal{M}'_C$.

2. The instance $0^{n+1}$ is negative for $\mathcal{M}'_C$

This consequences will allow us to finish the reduction. Consider now the instance $(\mathcal{M}'_C, 0^{n+1}, k+1)$ of $t$-MCR. We claim that this is a positive instance for the problem if and only if $(C, k)$ is a positive instance of $WCS(M_t)$.

For the forward direction, consider $(\mathcal{M}'_C, 0^{n+1}, k+1)$ to be a positive instance of $t$-MCR. This means there is an instance $\boldsymbol{x}^*$ that has the opposite classification as $0^{n+1}$ under $\mathcal{M}'_C$, and differs from it in at most $k+1$ features. By the second consequence of the claim, $\boldsymbol{x}^*$ must be a positive instance. Also, differing in at most $k+1$ features from $0^{n+1}$ means that $\boldsymbol{x}^*$ has weight at most $k+1$. But as majority gates are monotone connectives, majority circuits are monotones as well, so the existence of a positive instance $\boldsymbol{x}^*$ of weight at most $k+1$ implies the existence of a positive instance $\boldsymbol{x}'^*$ of weight exactly $k+1$. Therefore, by the first consequence of the claim, there is a satisfying assignment of weight exactly $k$ for $C$, which implies $(C, k)$ is a positive instance of $WCS(M_{t,t})$

For the backward direction, consider $(C, k)$ to be a positive instance of $WCS(M_{t,t})$. This means, by the first consequence of the claim, that there is a positive instance $\boldsymbol{x}^*$ of weight exactly $k+1$ for $\mathcal{M}'_C$. But based on the second consequence of the claim, $0^{n+1}$ is a negative instance for $\mathcal{M}'_C$. As $\boldsymbol{x}^*$ differs from $0^{n+1}$ in no more than $k+1$ features, and they have opposite classifications, we have that $(\mathcal{M}'_C, 0^{n+1}, k+1)$ is a positive instance of $t$-MCR.

As the whole construction takes polynomial time, and the reduction changes the parameter in a computable way, from $k$ to $k+1$, it is an fpt-reduction. This concludes the proof. □

We now proceed to prove membership in $W(\mathsf{Maj})[3t+7]$.

**Theorem 4.7** (Membership) There is an fpt-reduction from $t$-MCR to $WCS(M_{t+4,t+4})$, implying $(3t+3)$-MCR belongs to $W(\mathsf{Maj})[3t+7]$.

Proof. Let $(\mathcal{M}, \boldsymbol{x}, k)$ be an instance of $t$-MCR. During this reduction we assume that $n > 2k$, as otherwise the result can be achieved trivially; if $n \leq 2k$ then trying all instances that differ by at most $k$ from $\boldsymbol{x}$ takes only $O(k^k)$, and thus we can solve the entire problem in fpt-time and return a constant-size instance of $WCS(M_{t+2})$, completing the reduction.

We start by applying Lemma 4.3 to build an equivalent MLP $\mathcal{M}'$ that uses only step activation functions. As $t$ is constant, this construction takes polynomial time, and its resulting MLP $\mathcal{M}'$ has $t$ layers as well. If $\boldsymbol{x}$ is a negative instance of $\mathcal{M}'$ (and thus of $\mathcal{M}$) we do nothing. This can trivially be checked in polynomial time, evaluating $\boldsymbol{x}$ in $\mathcal{M}'$. But if $\boldsymbol{x}$ happens to be a positive instance of $\mathcal{M}'$, then we change the definition of $\mathcal{M}'$ negating its root perceptron[6], and thus making $\boldsymbol{x}$ a negative instance. As a result, we can safely assume $\boldsymbol{x}$ to be a negative instance of $\mathcal{M}'$. We can also, in the same fashion that we assumed $n > 2k$, discard the case where the instance $0^n$ is a positive instance of $\mathcal{M}'$ that differs by at most $k$ from $\boldsymbol{x}$, as in such scenario we could also solve the problem in fpt-time. The same can be done for $1^n$.

We now build an MLP $\mathcal{M}''$, that still uses only step activation functions, such that $\mathcal{M}''$ has a positive instance of weight exactly $k$ if and only if $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of $t$-MCR.

Let $\mathcal{M}''$ be a copy of $\mathcal{M}'$ to which we add one extra layer at the bottom. For each $1 \leq \mathrm{i} \leq n$, we connect the i-th input node of $\mathcal{M}''$ to what was the i-th input node of $\mathcal{M}'$, but is now an internal node in $\mathcal{M}''$. If $\boldsymbol{x}_\mathrm{i} = 0$ then the node in $\mathcal{M}''$ corresponding to the i-th input node of $\mathcal{M}'$ has a bias of 1, and the weight of the edge coming from the i-th input node of $\mathcal{M}''$ is also 1. On the other hand, if $\boldsymbol{x}_\mathrm{i} = 1$, then hen the node in $\mathcal{M}''$ corresponding to the i-th input node of $\mathcal{M}'$ has a bias of 0, and the weight of the connection added to it is $-1$. After doing this, we add $k-1$ more input nodes to $\mathcal{M}''$, a new node $p$ in the $t$-th layer and a new root node $r''$, that is placed in the layer $t+1$. We connect $r'$, the previous root node, to $r''$ of $\mathcal{M}'$ with weight 1, and all input nodes to node $p$ with weights of 1. In case $p$ is more than one layer above the new input nodes, we connect them through paths of identity gates, as shown in Lemma 3.9. We set the bias of $r''$ to $-2$, and the bias of $p$ to $-k$. All non-input nodes added in the construction use step activation

[6]Let $\mathcal{P} = (\boldsymbol{w}, \boldsymbol{b})$ be the perceptron at the root of $\mathcal{M}'$, which contain only integer values by construction. Then, the negation of $\mathcal{P}$ is simply $\bar{\mathcal{P}} = (-\boldsymbol{w}, -\boldsymbol{b}+1)$, as $-\boldsymbol{w}\boldsymbol{x} \geq -\boldsymbol{b}+1$ precisely when $\boldsymbol{w}\boldsymbol{x} \leq \boldsymbol{b}-1$, which occurs over the integers exactly when it is not true that $\boldsymbol{w}\boldsymbol{x} \geq \boldsymbol{b}$.

functions.

We now prove a claim stating that $\mathcal{M}''$ has exactly the intended behavior.

**Claim 4.8** The MLP $\mathcal{M}''$ has a positive instance of weight exactly $k$ if and only if $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of $t$-MCR.

Proof. For the forward direction, assume $\mathcal{M}''$ has a positive instance $\boldsymbol{x}'$ of weight exactly $k$. As the root $r''$ has a bias of $-2$, and two incoming edges with weight 1, and given that the output of any node is bounded by 1, as only step activation functions are used, we conclude that both $p$ and $r'$, the children of $r''$, must have a value of 1 on $\boldsymbol{x}'$. The fact that $r'$ has a value of 1 on $\boldsymbol{x}'$ implies that $\boldsymbol{x}^s$, the restriction of $\boldsymbol{x}$ that considers only nodes that descend from $r'$, must be a positive instance for the submodel $\mathcal{M}^s$ induced by considering only nodes that descend from $r'$. But one can easily check that by construction, we have that $\mathcal{M}^s(\boldsymbol{x}^s) = \mathcal{M}'(\boldsymbol{x}^s \oplus \boldsymbol{x})$, where $\oplus$ represents the bitwise-xor. Thus, $\boldsymbol{x}^s \oplus \boldsymbol{x}$ is a positive instance for $\mathcal{M}$, and consequently for $\mathcal{M}$. As $\boldsymbol{x}^s \oplus \boldsymbol{x}$ differs from $\boldsymbol{x}$ by exactly the weight of $\boldsymbol{x}^s$, as 0 is the neutral element of $\oplus$, and the weight of $\boldsymbol{x}^s$ is by definition no more than the weight of $\boldsymbol{x}'$, which is in turn no more than $k$ by hypothesis, we conclude that $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of $t$-MCR.

For the backward direction, assume there is a positive instance $\boldsymbol{x}'$ of $\mathcal{M}$ that differs from $\boldsymbol{x}$ in at most $k$ positions. This means that $\boldsymbol{x}'' = \boldsymbol{x} \oplus \boldsymbol{x}'$ has weight at most $k$. By the same argument used in the forward direction, $\mathcal{M}^s(\boldsymbol{x}'') = \mathcal{M}'(\boldsymbol{x}'' \oplus \boldsymbol{x}) = \mathcal{M}'(\boldsymbol{x}')$, as $\boldsymbol{x} \oplus \boldsymbol{x}' \oplus \boldsymbol{x} = \boldsymbol{x} \oplus \boldsymbol{x} \oplus \boldsymbol{x}' = \boldsymbol{x}'$, because $\oplus$ is both commutative and its own inverse. But the fact that $\boldsymbol{x}'$ is a positive instance of $\mathcal{M}$ implies that it is also a positive instance for $\mathcal{M}'$. As we are assuming $\boldsymbol{x}'| \neq 0^n$, we have that $k - |\boldsymbol{x}'| \leq k - 1$. Thus, we can create an instance $\boldsymbol{x}''$ for $\mathcal{M}''$ that is equal to $\boldsymbol{x}'$ on its corresponding features, and that sets $k - |\boldsymbol{x}'|$ arbitrary extra input nodes to 1, among those created in the construction of $\mathcal{M}''$. As the instance $\boldsymbol{x}''$ has weight exactly $k$, it satisfies the submodel descending from $p$, and as $\boldsymbol{x}''$ its equal to $\boldsymbol{x}'$ on the submodel descending from $r'$, and $\boldsymbol{x}'$ is a positive instance of $\mathcal{M}'$, we have that this submodel must be satisfied as well. Both submodels being satisfied, the whole model $\mathcal{M}''$ is satisfied, hence we conclude the proof. □

We thus have a model $\mathcal{M}''$ with step activation functions, and $t + 2$ layers, such that if that model has a satisfying assignment of weight exactly $k$, then $(\mathcal{M}, \boldsymbol{x}, k)$ is a positive instance of $t$-MCR.

Note that step activation functions with bias are equivalent to weighted threshold gates. We then use a result by Goldmann and Karpinski [24, Corollary 12] to build a circuit $C_{\mathcal{M}''}$ that is equivalent (as Boolean functions) to $\mathcal{M}''$ but uses only majority gates. The construction of Goldmann et al. can be carried in polynomial time, and guarantees that $C_{\mathcal{M}''}$ will have at most $t + 3$ layers.

There is however a caveat to surpass: although not explicitly stated in the work of Goldmann et al. [24], their definition of majority circuit must assume that for representing a Boolean function from $\{0,1\}^n$ to $\{0,1\}$, the circuit is granted access to $2n$ input variables $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \overline{\boldsymbol{x}_1}, \ldots, \overline{\boldsymbol{x}_n}$, as it is usual in the field, and described for example in the work of Allender [4]. We thus assume that the circuit $C_{\mathcal{M}''}$ resulting from the construction of Goldmann et al. has this structure, which

does not match the required structure of the majority circuits defining the $W(\mathsf{Maj})$-hierarchy as defined by Fellows et al [19, 20]. In order to solve this, we adapt a technique from Fellows et al. [20, p. 17]. We build a circuit $C^*_{\mathcal{M}''}$ that does fit the required structure. Let $n$ be the dimension of $\mathcal{M}''$ (which exceeds by $k-1$ that of $\mathcal{M}$). We now describe the steps one needs to apply to $C_{\mathcal{M}''}$ in order to obtain $C^*_{\mathcal{M}''}$.

1. Add a new layer with $n+1$ input nodes $\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_{n+1}$, below what previously was the layer of $2n$ input nodes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \overline{\boldsymbol{x}_1}, \ldots, \overline{\boldsymbol{x}_n}$.

2. For every $1 \leq \mathrm{i} \leq n$, connect input node $\boldsymbol{x}'_\mathrm{i}$ with its corresponding node $\boldsymbol{x}_\mathrm{i}$ in the second layer, making $\boldsymbol{x}_\mathrm{i}$ a unary majority, with the same outgoing edges it had as an input node. This enforces $\boldsymbol{x}_\mathrm{i} = \boldsymbol{x}'_\mathrm{i}$.

3. Create a new root $r'$ for the circuit, and let $r'$ be a binary majority between the input node $\boldsymbol{x}'_{n+1}$ and the previous root $r$.

4. Replace each previous input node $\overline{\boldsymbol{x}_\mathrm{i}}$ by a majority gates $m_\mathrm{i}$ that has $n+1-2k$ incoming edges from $\boldsymbol{x}'_{n+1}$, and one incoming edge from each $\boldsymbol{x}'_j$ with $j \notin \{\mathrm{i}, n+1\}$. The outgoing edges are preserved.

It is clear that the circuit $C^*_{\mathcal{M}''}$ is a valid majority circuit in the sense defining the $W(\mathsf{Maj})$-hierarchy. And it has 2 layers more than $C_{\mathcal{M}''}$, yielding a total of $t+5$ layers, where the last one has a small gate. However, it is not evident what this new circuit does. We now prove a tight relationship between the circuit $C^*_{\mathcal{M}''}$ and $\mathcal{M}''$.

**Claim 4.9** The circuit $C^*_{\mathcal{M}''}$ has a satisfying assignment of weight exactly $k+1$ if and only if $\mathcal{M}''$ has a positive instance of weight exactly $k$.

Proof. **Forward Direction.** Assume $C^*_{\mathcal{M}''}$ has a satisfying assignment of weight $k+1$. By step 3 of the construction, in order to satisfy $C^*_{\mathcal{M}''}$, the assignment must set $\boldsymbol{x}'_{n+1}$ to 1.

As we assume that node $\boldsymbol{x}'_{n+1}$ is set to 1, the assignment must set to 1 exactly $k$ input nodes among $\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_n$ and thus the sum of inputs set to 1 of each majority gate $m_\mathrm{i}$ constructed in step 4, is exactly equal to

$$n+1-2k+\sum_{j \notin \{\mathrm{i}, n+1\}} \boldsymbol{x}'_j = n+1-2k+(k-\boldsymbol{x}'_\mathrm{i}) = n+1-k-\boldsymbol{x}'_\mathrm{i}$$

and its fan-in is exactly equal to $2n-2k$. Therefore $m_\mathrm{i}$ is activated when $n+1-k-\boldsymbol{x}'_\mathrm{i} > n-k$, which happens precisely when $\boldsymbol{x}'_\mathrm{i} = 0$. This way, each gate $m_\mathrm{i}$ corresponds to the negation of $\boldsymbol{x}'_\mathrm{i}$.

This way, the subcircuit induced by considering only the nodes that descend from $r'$ computes the same Boolean function that $C_{\mathcal{M}''}$ computes, under the natural mapping of their variables. Therefore, a satisfying assignment of weight $k+1$ for $C^*_{\mathcal{M}''}$ implies the existence of a satisfying assignment for $C_{\mathcal{M}''}$ that chooses exactly $k$ positive variables, and thus a positive instance of weight $k$ for $\mathcal{M}''$.

**Backward Direction.** Assume $\mathcal{M}''$ has a positive instance of weight exactly $k$. That implies

that $C_{\mathcal{M}''}$ has a satisfying assignment $\sigma$ that sets at most $k$ positive variables to 1. Let us consider the assignment $\sigma'$ for $C^*_{\mathcal{M}''}$ that sets to 1 the same variables that $\sigma$ does, and additionally sets $\boldsymbol{x}_{n+1}$ to 1. The assignment $\sigma'$ has weight exactly $k+1$. By the same argument used in the forward direction, under assignment $\sigma'$ the gates $m_{\mathrm{i}}$ behave like negations. Thus, the assignment $\sigma'$ induces an assignment over the second layer of $C^*_{\mathcal{M}''}$ that corresponds precisely to a satisfying assignment of $C_{\mathcal{M}''}$, and thus makes the value of $r$ equal to 1. As both $r$ and $\boldsymbol{x}_{n+1}$ have value 1 under assignment $\sigma'$, it follows that the value of $r'$, and thus of circuit $C^*_{\mathcal{M}''}$, are 1 under $\sigma'$ as well. This means that assignment $\sigma'$, which by construction has weight $k+1$, is a satisfying assignment for $C^*_{\mathcal{M}''}$, and thus concludes the proof. □

By combining Claim 4.8 and Claim 4.9, and noting again that circuit $C^*_{\mathcal{M}''}$ is a valid majority circuit, in the sense that defines the $W(\mathsf{Maj})$-hierarchy, and has weft at most $t+4$, we conclude the reduction of Theorem 4.7. □

## 4.5 Application of the parameterized result

Let us restate the parameterized theorem we proved in Section 4.4.

**Theorem** For every $t \geq 1$ the MinimumChangeRequired query over rMLPs with $3t+3$ layers is $W(\mathsf{Maj})[t]$-hard and is contained in $W(\mathsf{Maj})[3t+7]$.

By assuming that the $W(\mathsf{Maj})$-hierarchy is strict, we can use Theorem 4.1 to provide separations for rMLPs with different numbers of layers. For instance, instantiating the above result with $t = 1$ we obtain that for rMLPs with 6 layers, the MCR problem is in $W(\mathsf{Maj})[3t+7] = W(\mathsf{Maj})[10]$. Moreover, instantiating it with $t = 11$ we obtain that for rMLPs with 36 layers, the MCR problem is $W(\mathsf{Maj})[11]$-hard. Thus, assuming that $W(\mathsf{Maj})[10] \subsetneq W(\mathsf{Maj})[11]$ we obtain that rMLPs with 6 layers are strictly more c-interpretable than rMLPs with 36 layers. We generalize this observation in the following result.

**Theorem 4.10** Assume that the $W(\mathsf{Maj})$-hierarchy is strict. Then for every $t \geq 1$ we have that rMLPs with $3t+3$ layers are strictly more c-interpretable than rMLPs with $9t+27$ layers wrt. MCR.

Proof. Based on Theorem 4.1, we know that interpreting an rMLP (for the problem MCR) with $9t+27 = 3(3t+8)+3$ is $W(\mathsf{Maj})[3t+8]$-hard. On the other hand, by using the same theorem, the problem of interpreting an rMLP with $3t+3$ layers is contained in $W(\mathsf{Maj})[3t+7]$. But by hypothesis, $W(\mathsf{Maj})[3t+7] \subsetneq W(\mathsf{Maj})[3t+8]$, which is enough to conclude the proof. □

# Chapter 5

# Applications to Fairness and Bias Detection

The increasing use of AI in sensitive domains has brought increasing attention to its implications in perpetuating biases, disparity and unfairness. For example, just during June 2020[1], two significant examples of the perils of biased systems have made it to the mainstream media. On the one hand, presented in Figure 5.1, a generative system designed to upscale pixelated images has been widely critiqued for its racial bias [3], on the other hand, an important image dataset was discovered to contain racist and misogynistic labels, and consequently taken down [1]. An older example, but of immense importance, is that of COMPAS, a proprietary system widely used in the United States Justice system for parole and bail decisions, that predicts recidivism risk based on +130 factors. COMPAS has been accused of being racially biased [2], but precisely because of its proprietary nature, it is difficult to assess whether the model depends directly on race [49], and even to estimate the dependency on features like *age*, when its creators claimed a linear dependency [49]. In the words of Rudin et al. "Lack of transparency makes it difficult to assess any of the myriads forms of fairness. [...] It can hide bias toward underrepresented groups, or conversely, it can make fair models seem biased" [49]. In previous work by Rudin [48], we find a positive framing of this same perspective "It can be much easier to detect and debate possible bias or unfairness with an interpretable model than with a black box. Similarly, it could be easier to detect and avoid data privacy issues with interpretable models than black boxes". We take this perspective to analyze how our previous results have a correlate with the complexity of finding bias or unfairness in different classes of bias.

## 5.1 About the notion of (un)fairness used

The concepts of bias and unfairness are multidimensional, and require multidisciplinary perspectives and a constant dialogue with the social sciences [42]. We take a rather simplistic approach to fairness, by considering only a particular kind; *fairness through unawareness* [22, 37]. In this notion, a certain subset of the features (that is, components of the input) is said to be *protected*, and consist of features that intuitively should not be used for decision taking. This usually includes

[1]That is, in the last 2 weeks from the moment this sentence is being written.

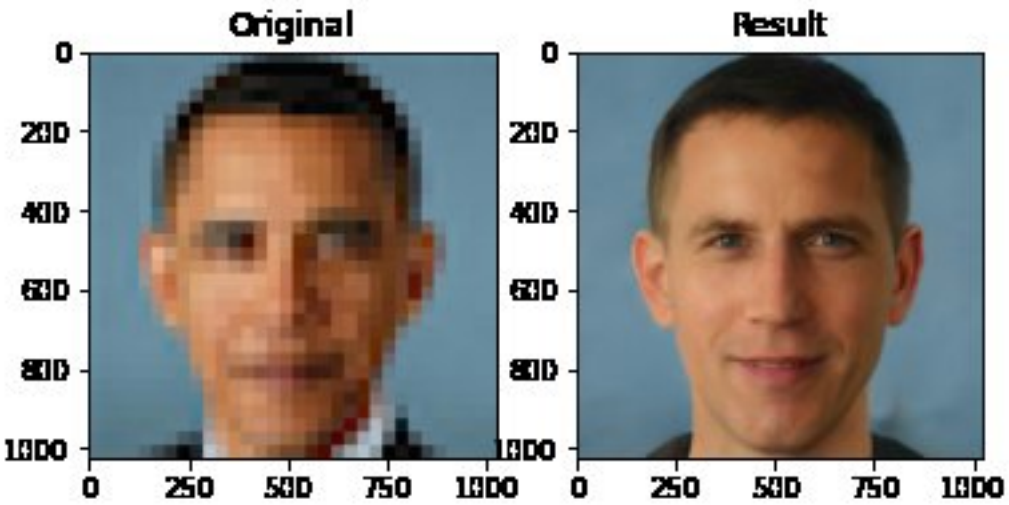



Figure 5.1: Pixelated picture of ex-president Barack Obama reconstructed as a white man by the PULSE algorithm. From Twitter user @Chicken3gg.

features like gender, age, marital status, etc.

**Definition 5.1** (Fairness through unawareness) A model is said to achieve fairness through unawareness if protected features are not explicitly used in the prediction process.

As pointed out by Barocas et al. [7], the direct effect of protected variables is insufficient as a measure of discrimination on its own. It cannot detect any form of *proxy discrimination*, as for example, a bank could choose to not ask for the age of loan applicants, but infer it from the vocabulary used in a written application.

However, the ability to formally establish whether certain protected features where or not causal for decisions taken by models can still be of great impact, as discussed in the case of COMPAS [49]. It is also plausible that we will face the need of retroactively analyze this sort of fairness, as features that are not deemed problematic as of today could be deemed protected in the future.

## 5.2 Definitions of bias

We use the formalization of Darwiche et al. [13], merely adapting the terminology to our setting.

**Definition 5.2** (Biased decision) Given a model $\mathcal{M}$, of input size $n$ and a set of protected features $\mathcal{P} \subseteq \{1, \ldots, n\}$, an instance $\boldsymbol{x}$ is said to be a biased decision of $\mathcal{M}$ if and only if there exists an instance $\boldsymbol{y}$ such that $\boldsymbol{x}$ and $\boldsymbol{y}$ differ only on features in $\mathcal{P}$ and $\mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y})$.

This definition is then trivially extended to models.

**Definition 5.3** (Biased model) A model $\mathcal{M}$ is said to be biased if and only if there is an instance

$\boldsymbol{x}$ that is a biased decision of $\mathcal{M}$.

**Example 2** (Related to Example 1) Consider that among the features used by a bank to judge loan applications we find both the *criminal record* and the *marital status* of applicants. If the law forbids said considerations, deeming the features *Criminal Record* and *Married* to be protected, then individuals could rebut decisions they received that could have unfairly been based on such features. Moreover, one would be interesting in analyzing a posteriori, given a trained model, whether there is actually an individual who could have been classified unfairly.

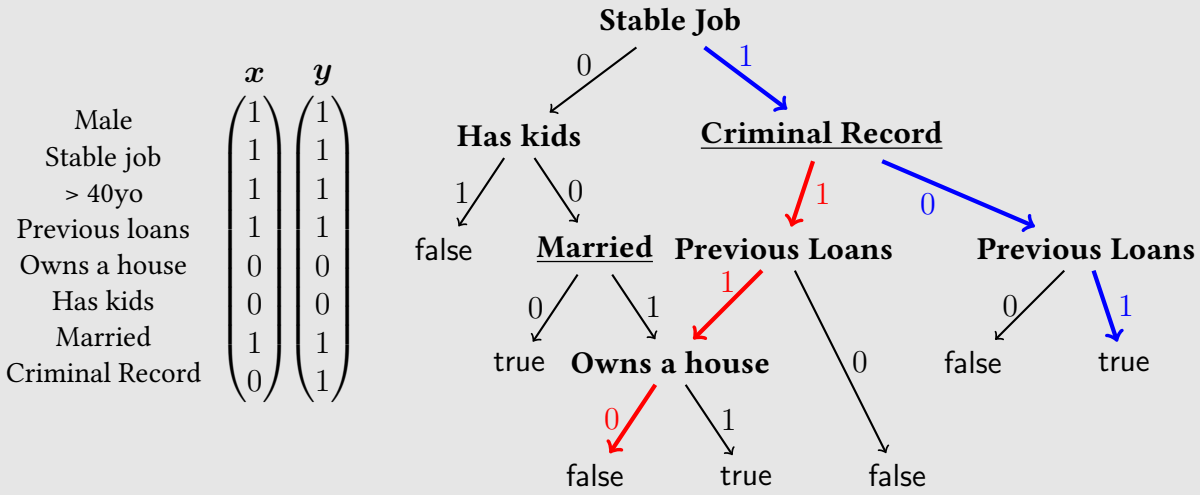



Figure 5.2: Illustration of the bias in an FBDD used for judging loan applications. Instance $\boldsymbol{x}$ follows the blue path, while instance $\boldsymbol{y}$ follows the red path. Protected features are underlined.

As illustrated in Figure 5.2, this particular FBDD is biased, as it presents a biased decision. The instance $\boldsymbol{x}$ is a positive instance of the model, while the negative instance $\boldsymbol{y}$ differs from it only on the protected feature corresponding to the criminal record.

## 5.3 On the complexity of bias detection

Following the same principles of the presented framework, we study the complexity of computational problems associated with the previous definitions of bias over different models (MLPs, FBDDs and Perceptrons).

Let us define the precise computational problems concerned by our study.

| | |
|---|---|
| Problem: | DecisionBias |
| Input: | A model $\mathcal{M}$, a set of protected features $\mathcal{P}$ and an instance $\boldsymbol{x}$ |
| Output: | Yes, if $\boldsymbol{x}$ is a biased decision of $\mathcal{M}$ and No otherwise |

| | FBDDs | Perceptrons | MLPs |
|---|---|---|---|
| DecisionBias | PTIME (5.4) | PTIME (5.5) | NP-complete (5.7) |
| ModelBias | Open, partial results (5.8) & (5.11) | NP-complete (5.16) | NP-complete (5.18) |

Table 5.1: Complexity results for bias detection.

| | |
|---|---|
| Problem: | ModelBias |
| Input: | A model $\mathcal{M}$ and a set of features $\mathcal{P}$ |
| Output: | Yes, if $\mathcal{M}$ is a biased and No otherwise |

As summarized in Table 5.1, we confirm once again that bias detection is harder over MLPs than it is over FBDDs and perceptrons, while in this case, FBDDs present a significant advantage over perceptrons when it comes to judge the fairness of the model as a whole. Even though the general case of FBDDs remains open, we prove in Theorem 5.8 that in the particular case of decision trees, there is an algorithm that runs in polynomial time for the ModelBias problem. We show as well a non-trivial randomized algorithm for the natural parameterized problem.

**Theorem 5.4** The DecisionBias problem can be solved in linear time for FBDDs

A proof is presented in the work of Darwiche et al. [13], however, we present a couple of new ones based on our previous proofs in Chapter 3, showing their applicability to new problems.

As a first example, Theorem 3.25 says that it is possible to count the number of positive completions of a given partial instance in polynomial time. We can use result in the following way. Given an instance $\boldsymbol{x}$ and protected features $\mathcal{P}$, define the partial instance $\boldsymbol{x}'$ such that $x'_{\mathrm{i}} = x_{\mathrm{i}}$ if $\mathrm{i} \notin \mathcal{P}$ and $x'_{\mathrm{i}} = \perp$ if $\mathrm{i} \in \mathcal{P}$. Then, if $\boldsymbol{x}$ is a positive instance of the input model, and $\boldsymbol{x}'$ has less than $2^{|\mathcal{P}|}$ positive completions, we have that at least one completion $\boldsymbol{x}''$ is negative, and as $\boldsymbol{x}''$ differs from $\boldsymbol{x}$ only on protected features, we conclude that $\boldsymbol{x}$ is a biased decision. Otherwise, all completions of $\boldsymbol{x}'$ are positive, and thus no instance differing from $\boldsymbol{x}$ only on protected features is negative, so we conclude $\boldsymbol{x}$ is not a biased decision. If $\boldsymbol{x}$ happened to be a negative instance, it is enough to check whether $\boldsymbol{x}'$ has at least 1 positive completion.

For an alternative proof, consider the proof of Lemma 3.5, where we show how to compute the minimum change required for an FBDD in linear time. Well, we can make a trivial modification to the recursive formula so only protected features can be changed. Namely

$$\mathrm{mcr}^{\mathcal{P}}_u(\boldsymbol{x}) = \begin{cases} \min\left([x_u = 1] + \mathrm{mcr}^{\mathcal{P}}_{u_0}(\boldsymbol{x}), [x_u = 0] + \mathrm{mcr}^{\mathcal{P}}_{u_1}(\boldsymbol{x})\right) & \text{if } u \in \mathcal{P} \\ {}^{2}\mathrm{mcr}^{\mathcal{P}}_{u_{x_u}}(x) & \text{otherwise} \end{cases}$$

It is clear, after the proof of Lemma 3.5, that this recursive formula is correct. Finally, considering $r$ the root of the given FBDD, it is enough to check whether $\mathrm{mcr}^{\mathcal{P}}_r(\boldsymbol{x}) \neq \infty$ to decide whether

[2]Hopefully, the reader will excuse the use of such a complicated sub-index. The variable $u_{x_u}$ means simply the child of node $u$ going along the edge with label $x_u$. Note as well that such a notation allows for a sudden appearance of the emoticon uwu when considering an instance named $w$.

$\boldsymbol{x}$ is a biased decision or not.

**Theorem 5.5** The DecisionBias problem can be solved in linear time for perceptrons.

Proof. Consider an input instance $(\mathcal{M} = (\boldsymbol{w}, b), \boldsymbol{x}, \mathcal{P})$, where we assume $\boldsymbol{x}$ to be a positive instance wlog. We define the instance $\boldsymbol{x}'$ as

$$x'_{\mathrm{i}} = \begin{cases} x_{\mathrm{i}} & \text{if i} \notin \mathcal{P} \\ 1 & \text{else if } w_{\mathrm{i}} < 0 \\ 0 & \text{otherwise} \end{cases}$$

We claim that $\boldsymbol{x}'$ is a negative instance of $\mathcal{M}$ if and only if $(\mathcal{M} = (\boldsymbol{w}, b), \boldsymbol{x}, \mathcal{P})$ is a positive instance of DecisionBias. The forward direction is trivial, as $\boldsymbol{x}'$ is an instance that only differs from $\boldsymbol{x}$ on protected features and has opposite classification. For the backward direction, assume for the sake of a contradiction that we are dealing with a positive instance of DecisionBias and yet $\boldsymbol{x}'$ is a positive instance of $\mathcal{M}$. The fact that we are dealing with a positive instance of DecisionBias implies there is an instance $\boldsymbol{x}''$ that differs with $\boldsymbol{x}$ only on protected features and such that $\mathcal{M}(\boldsymbol{x}'') = 0$. We will prove in an instant that $\langle \boldsymbol{w}, \boldsymbol{x}'' \rangle \geq \langle \boldsymbol{w}, \boldsymbol{x}' \rangle$ which is enough to conclude that the assumption $\mathcal{M}(\boldsymbol{x}'') = 0$ implies $\mathcal{M}(\boldsymbol{x}') = 0$, a contradiction with our hypothesis of $\boldsymbol{x}'$ being a positive instance for $\mathcal{M}$.

**Claim 5.6** $\langle \boldsymbol{w}, \boldsymbol{x}'' \rangle \geq \langle \boldsymbol{w}, \boldsymbol{x}' \rangle$

In order to prove this claim, and finish the whole proof, it is enough to consider an arbitrary index i and check that $w_{\mathrm{i}} x''_{\mathrm{i}} \geq w_{\mathrm{i}} x'_{\mathrm{i}}$. We do so by cases, if $\mathrm{i} \notin \mathcal{P}$, equality must be achieved as neither $\boldsymbol{x}'$ nor $\boldsymbol{x}''$ can differ from $\boldsymbol{x}$ on unprotected features. If $w_{\mathrm{i}} < 0$, then $w_{\mathrm{i}} x'_{\mathrm{i}} = w_{\mathrm{i}} \geq w_{\mathrm{i}} x''_{\mathrm{i}}$, as $x''_{\mathrm{i}}$ can only be 0 or 1. Finally, if $w_{\mathrm{i}} \geq 0$, then $w_{\mathrm{i}} x'_{\mathrm{i}} = 0 \geq w_{\mathrm{i}} x''_{\mathrm{i}}$. □

**Theorem 5.7** The DecisionBias problem is NP-complete for MLPs.

Proof. Membership is easy; it is enough to guess an instance $\boldsymbol{y}$ that differs from the input instance $\boldsymbol{x}$ only on protected features and then check that they receive opposite classifications by the input model $\mathcal{M}$. For hardness, consider the particular case when every feature is protected. That is, if $n$ is the input size of $\mathcal{M}$, then we establish $\mathcal{P} = \{1, \ldots, n\}$. It follows from Definition 5.2 that $\boldsymbol{x}$ is biased if and only if $\mathcal{M}$ is not trivial[3], as any instance with different classification than $\boldsymbol{x}$ would prove that $\boldsymbol{x}$ is a biased decision. We then use Lemma 3.9 to reduce in polynomial time from the problem of checking whether a Boolean formula in CNF is not trivial, meaning either unsatisfiable or a tautology. As it can be checked in polynomial time whether a CNF formula is a a tautology or not, the problem of deciding whether a Boolean formula in CNF is trivial must be *co*NP-hard, as deciding whether a CNF formula is satisfiable is known to be NP-hard. As the problem of

[3] A trivial model is one that classifies every input the same.

deciding whether a CNF formula is trivial is *co*NP-hard, and we have shown a reduction from DecisionBias to its complement, we have that DecisionBias is NP-hard. □

**Theorem 5.8** The ModelBias problem can be solved in polynomial time for Decision Trees

Proof. Let $\mathcal{T}$ be a decision tree, and $\mathcal{P}$ the set of protected features. We say a node $u$ is a *biased splitter* of $\mathcal{T}$ if there are two instances $\boldsymbol{x}$ and $\boldsymbol{y}$ that differ only on protected features such that both instances take the same path on $\mathcal{T}$ until $u$, and then their paths split, in such a way that one of them reaches a true leaf, while the other reaches a false leaf. It naturally follows that $u$ must be a protected feature, but most importantly, that is a biased splitter in $\mathcal{T}$ if and only if $\mathcal{T}$ is biased. The forward direction is trivial. For the forward direction, consider that two instances that differ on protected features and have opposite classification must take different paths on $\mathcal{T}$, and thus, their paths have a common prefix (even if only the root of $\mathcal{T}$) whose last element is precisely $u$. It is thus enough to be able to check whether a given node $v$ of $\mathcal{T}$, labeled with a protected feature, is a biased splitter of $\mathcal{M}$. This way, we can iterate over all the possible nodes $v$, and if any of them happens to be a biased splitter, we can confirm model $\mathcal{T}$ is biased. If no bias splitter is found, we conclude that the decision tree $\mathcal{T}$ is not biased. As a consequence, let us focus now in the problem BiasedSplitter($v$), that aims to check whether the given node $v$ is a biased splitter or not.

Considere an arbitrary node $v$ and its associated sub-tree $\mathcal{T}_v$. For an arbitrary instance $\boldsymbol{x}$, we call $\pi^v_{\boldsymbol{x}}$ its path over $\mathcal{T}_v$. By definition, node $v$ is a biased splitter if there are two leaves, $\ell_1$ and $\ell_2$ such that, without loss of generality, $\ell_1$ is labeled true while $\ell_2$ is labeled false, and there are instances $\boldsymbol{x}$ and $\boldsymbol{y}$, such that $\pi^v_{\boldsymbol{x}}$ ends in $\ell_1$, $\pi^v_{\boldsymbol{y}}$ ends in $\ell_2$, and $\pi^v_{\boldsymbol{x}}$ and $\pi^v_{\boldsymbol{y}}$ differ only on protected features[4]. As $\mathcal{T}$ is a tree, given a leaf $\ell_1$, we can get its unique path from node $v$ in linear time. Thus, in order to check whether $v$ is indeed a biased splitter, we can simply iterate over all the, at most $\binom{|\mathcal{T}|}{2}$, pairs of leaves $\ell_1$ and $\ell_2$ (with labels true and false respectively), and check whether the paths $v \rightsquigarrow \ell_1$ and $v \rightsquigarrow \ell_2$ differ only in protected features.

The total complexity of the algorithm is bounded by $O(|\mathcal{T}|^4)$, as at most $|\mathcal{T}|$ nodes are tested to be biased splitters, and for each fixed node, we test at most $|\mathcal{T}|^2$ pairs of leafs, where each test takes time $|\mathcal{T}|$. □

An important problem studied for many classes of Boolean functions is that of deciding equivalence. To the best of our knowledge, the problem of deciding whether two FBDDs are equivalent has not be shown to be in PTIME. However, Blum et al. designed an algorithm in randomized polynomial[5] time for this problem [10]. We now show by a trivial reduction that a deterministic polynomial time algorithm for ModelBias over FBDDs would imply the equivalence problem to be in PTIME as well.

**Theorem 5.9** If the ModelBias problem can be solved in PTIME for FBDDs, then equivalence (as boolean functions) of FBDDs can be solved in PTIME as well.

[4]We can represent paths as $n_1 \mathrm{e}_1 n_2 \dots \mathrm{e}_k n_{k+1}$, where $n_\mathrm{i} \mathrm{e}_\mathrm{i} n_{\mathrm{i}+1}$ means the path takes the edge with label $\mathrm{e}_\mathrm{i}$ from node $n_\mathrm{i}$ to node $n_{\mathrm{i}+1}$.

[5]A precise definition is out of our current scope, but a motivated reader can refer to Arora and Barak [5].

PROOF. Consider an input instance of the equivalence problem with FBDDs $\mathcal{M}_1$ and $\mathcal{M}_2$, over features $\{1, \ldots, n\}$.[6]

Let $\mathcal{M}$ be an FBDD over features $\{1, \ldots, n+1\}$ such that its root node $r$ is labeled with feature $n+1$, and it has $\mathcal{M}_1$ as its sub-model along the edge labeled with 0 and $\mathcal{M}_2$ as its sub-model along the edge labeled with 1. Consider $n+1$ to be the only protected feature.

**Claim 5.10** The constructed model $\mathcal{M}$ is biased if and only if $\mathcal{M}_1$ is *not* equivalent to $\mathcal{M}_2$.

For the forward direction, as $\mathcal{M}$ is biased there exist instances $\boldsymbol{x}, \boldsymbol{y}$, differing only on feature $n+1$, such that $\mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y})$. Without loss of generality, $x_{n+1} = 0$ while $y_{n+1} = 1$. As $\mathcal{M}_1$ does not mention the feature $n+1$, the instance $\boldsymbol{x}'$ that is equal to the restriction $\boldsymbol{x}$ to features $\{1, \ldots, n\}$ holds that $\mathcal{M}_1(\boldsymbol{x}') = \mathcal{M}(\boldsymbol{x})$. Analogously, $\mathcal{M}_2(\boldsymbol{y}') = \mathcal{M}(\boldsymbol{y})$. But as $\boldsymbol{x}' = \boldsymbol{y}'$, which implies $\mathcal{M}_1(\boldsymbol{x}') = \mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y}) = \mathcal{M}_2(\boldsymbol{x}')$. Thus, models $\mathcal{M}_1$ and $\mathcal{M}_2$ are not equivalent, as they differ on instance $\boldsymbol{x}'$. For the backward direction, the existence of an instance $\boldsymbol{x}'$ such that $\mathcal{M}_1(\boldsymbol{x}') \neq \mathcal{M}_2(\boldsymbol{x}')$ implies that $\boldsymbol{x} = \boldsymbol{x}' \odot 0$, where $\odot$ represents concatenation, is a biased decision of $\mathcal{M}$, as $\mathcal{M}(\boldsymbol{x}' \odot 0) = \mathcal{M}_1(\boldsymbol{x}') \neq \mathcal{M}_2(\boldsymbol{x}') = \mathcal{M}(\boldsymbol{x}' \odot 1)$. Thus, $\boldsymbol{x}' \odot 0$ and $\boldsymbol{x}' \odot 1$ are instance of $\mathcal{M}$ that differ only on a protected feature and yet have opposite classification. This makes $\mathcal{M}$ a biased model. □

As a result, a polynomial algorithm for the MODELBIAS problem over FBDDs would be of significant theoretical importance.

We now give a positive theoretical result for the case when $|\mathcal{P}|$, the number of protected features, is relatively small. Consider the following parameterized problem.

| | |
|---|---|
| Problem: | p-MODELBIAS |
| Input: | An FBDD $\mathcal{M}$, a set of protected features $\mathcal{P}$ |
| Parameter: | $\mathcal{P}$ |
| Output: | YES, $\mathcal{M}$ is biased and NO otherwise. |

**Theorem 5.11** There is an algorithm for the p-MODELBIAS problem that runs in FPT time, and that always gives the right answer for unbiased input models, while having a probability of error of at most $1/2$ for biased input models.

The proof heavily uses the probabilistic algorithm for FBDD equivalence by Blum et al. We state it next.

**Theorem 5.12** (Blum et al. [10]) Given FBDDs $\mathcal{M}_1$ and $\mathcal{M}_2$, there is a randomized algorithm that runs in polynomial time such that if $\mathcal{M}_1 \equiv \mathcal{M}_2$ the algorithm outputs YES correctly every single time, whereas if $\mathcal{M}_1 \not\equiv \mathcal{M}_2$, the algorithm outputs NO correctly with probability at least $1/2$.

[6]Note that two FBDDs that mention different features cannot be equivalent in general, which can be easily checked in polynomial time. Thus, meaningful instances of the problem refer only to models over a common set of features.

We can use this result to prove Theorem 5.11.

Proof of Theorem 5.11. Let $\mathcal{M}$ be an FBDD, and $k := |\mathcal{P}|$. Consider, as in the proof for Theorem 5.8, the problem BiasedSplitter that given the model $\mathcal{M}$, a node $u$, and the set of protected features $\mathcal{P}$, decides whether $u$ is a biased splitter of $\mathcal{M}$. We claim that solving this problem with high enough probability is enough to solve the p-ModelBias problem.

**Claim 5.13** Assume there is an algorithm $A$ for the BiasedSplitter problem that runs in polynomial time, and such that when the algorithm says Yes it is always correct, and when it says No it has a probability of error of at most $1/2$. Then, we can solve the p-ModelBias problem in FPT time, with guaranteed success on unbiased input models, and probability of error at most $1/2$ for biased input models.

Proof of Claim 5.13. Recall that a model $\mathcal{M}$ is biased if and only if there is a biased splitter. Let $B$ be the algorithm for p-ModelBias that iterates over all $k$ nodes labeled with protected features, and if for at least one of them $A$ says it is a biased splitter, then $B$ says $\mathcal{M}$ is biased, otherwise $B$ says $\mathcal{M}$ is not biased. As $A$ is correct when it says a node is biased splitter, $B$ only be wrong if there is a biased splitter that $A$ does not recognize. In the worst case, only $1$ node $u$ in $\mathcal{M}$ is a biased splitter, and thus the probability of failure of $B$ is at most $1/2$, as $A$ fails to recognize $u$ with probability at most $1/2$. □

Before presenting an algorithm $A$ satisfying the requirements of Claim 5.13, we define the notation $\mathcal{M}|\sigma$ for an arbitrary model $\mathcal{M}$ and $\sigma$ an arbitrary assignment of subset of the features of $\mathcal{M}$. Given $\mathcal{M}$ and $\sigma$, the model $\mathcal{M}|\sigma$ is the restriction of $\mathcal{M}$ that assumes every feature takes the value prescribed by $\sigma$. Namely, if a feature $p \in Dom(\sigma)$ is the label of a node $u$, then $u$ is deleted, and every node that had $u$ as a child gets now connected to $u_{\sigma(p)}$. Note that given a model $\mathcal{M}$ and a partial assignment $\sigma$, one can compute $\mathcal{M}|\sigma$ in polynomial time. If we define the support of a model as the set of features it checks, then the support of $\mathcal{M}|\sigma$ is equal to the support of $\mathcal{M} \setminus Dom(\sigma)$, as variables of $Dom(\sigma)$ have been removed by the *conditioning* process of $\sigma$.

We now present an algorithm that satisfies the requirements of Claim 5.13.

**Algorithm 1** Probabilistic Biased Splitter Check

**Precondition:** Let PrEquiv$(\mathcal{M}, \mathcal{M}')$ the algorithm for checking probabilistic equivalence of Theorem 5.12

**function** BiasedSplitter($\mathcal{M}, u, \mathcal{P}$)
  $\mathcal{M}_1 \leftarrow \mathcal{M}_{u_0}$ ▷ the sub-model along the 0-edge of $u$
  $\mathcal{M}_2 \leftarrow \mathcal{M}_{u_1}$ ▷ the sub-model along the 1-edge of $u$
  **for** $\sigma_1 \leftarrow$ assignment of all features in $\mathcal{P}$ **do**
    **for** $\sigma_2 \neq \sigma_1 \leftarrow$ assignment of all features in $\mathcal{P}$ **do**
      **if not** PrEquiv$(\mathcal{M}_1|\sigma_1, \mathcal{M}_2|\sigma_2)$ **then**
        **return** $u$ is a biased splitter
  **return** $u$ is not a biased splitter

Given that PrEquiv runs in polynomial time, and the $\mathcal{M}_{\mathrm{i}}|\sigma_{\mathrm{i}}$ can be computed in polynomial time, Algorithm 1 runs in time $|\mathcal{M}|^{O(1)}2^{2k}$. Let us now analyze its correctness.

**Claim 5.14** If Algorithm 1 returns on line 7, then it must be correct.

Proof of Claim 5.14. For Algorithm 1 to return on line 7, the algorithm PrEquiv must have said that $\mathcal{M}_1|\sigma_1$ is not equivalent to $\mathcal{M}_2|\sigma_2$, and because of Theorem 5.12, that means $\mathcal{M}_1|\sigma_1 \not\equiv \mathcal{M}_2|\sigma_2$. Thus, there is an instance $\boldsymbol{x}$ such that $\mathcal{M}_1|\sigma_1(\boldsymbol{x}) \neq \mathcal{M}_2|\sigma_2(\boldsymbol{x})$, which implies $\mathcal{M}_1(\boldsymbol{x} \odot \sigma_1) \neq \mathcal{M}_2(\boldsymbol{x} \odot \sigma_2)$, where $\boldsymbol{x} \odot \sigma_{\mathrm{i}}$ means the instance over all features of $\mathcal{M}_{\mathrm{i}}$ that takes the values of $\boldsymbol{x}$ on unprotected features, and the values prescribed by $\sigma$ on protected features. Let $\pi_u(\mathcal{M})$ be any path from the root of $\mathcal{M}$ down to node $u$. As $\mathcal{M}$ is an FBDD, no feature present in $\pi_u(\mathcal{M})$ can be present in $\mathcal{M}_1$ nor $\mathcal{M}_2$ and thus, we can define instances $\pi_u(\mathcal{M}) \odot (\mathrm{i}-1) \odot \boldsymbol{x} \odot \sigma_{\mathrm{i}}$ that take the values prescribed by $\pi_u(\mathcal{M})$ on the features in that path, have value $(\mathrm{i}-1)$ on the feature that labels $u$, and have the values prescribed by $\boldsymbol{x}$ and $\sigma_{\mathrm{i}}$, while haviing arbitrary values in the features that are not prescribed by any of these. The defined instance $\pi_u(\mathcal{M}) \odot \boldsymbol{x} \odot \sigma_{\mathrm{i}}$ are valid instances of $\mathcal{M}$. We have that

$$\mathcal{M}\big(\pi_u(\mathcal{M}) \odot 0 \odot \boldsymbol{x} \odot \sigma_1\big) = \mathcal{M}_1(\boldsymbol{x} \odot \sigma_1) \neq \mathcal{M}_2(\boldsymbol{x} \odot \sigma_2) = \mathcal{M}\big(\pi_u(\mathcal{M}) \odot 1 \odot \boldsymbol{x} \odot \sigma_2\big)$$

Thus, we have found two instances that differ only on protected features, whose paths split at $u$, and for which $\mathcal{M}$ has opposite classification. Therefore $u$ is correctly labeled as a biased splitter. □

**Claim 5.15** If Algorithm 1 returns on line 8, then its probability of error is at most $1/2$.

Proof of Claim 5.15. For Algorithm 1 to err on line 8, node $u$ must be a biased splitter, which means there is at least one pair of instances $\boldsymbol{x}$ and $\boldsymbol{y}$ such that wlog[7] $\mathcal{M}_1(\boldsymbol{x}) \neq \mathcal{M}_2(\boldsymbol{y})$. Let $\sigma_1^*$ be the assignment of protected features induced by $\boldsymbol{x}$ and $\sigma_2^*$ the one induced by $\boldsymbol{y}$. As we assume that Algorithm 1 returned on line 8, the call PrEquiv$(\mathcal{M}_1|\sigma_1^*, \mathcal{M}_2|\sigma_2^*)$ must have returned true. But this is a mistake, as we knew that $\mathcal{M}_1(\boldsymbol{x}) \neq \mathcal{M}_2(\boldsymbol{y})$. Thus, for Algorithm 1 to err on line 8, we need that every call to PrEquiv with instances $\boldsymbol{x}$ and $\boldsymbol{y}$ that verify $u$ as a biased splitter fail. As in the worst case there is only one such pair, the probability of error of Algorithm 1 at most that of PrEquiv. □

By combining Claim 5.13, Claim 5.14 and Claim 5.15 we conclude the proof of Theorem 5.11. □

Let us recall the reader that a probability of error of $1/2$ can be trivially reduced to $1/2^t$ by running the algorithm $t$ times in a row, this is a standard amplification technique [5].

We can now resume our analysis of the complexity of the ModelBias problems for perceptrons and MLPs.

[7] we are assuming here $\boldsymbol{x}$ has the feature that labels node $u$ with value 0, if this is not the case one can simply swap $\boldsymbol{x}$ and $\boldsymbol{y}$.

**Theorem 5.16** The ModelBias problem is NP-complete for perceptrons.

Membership is easy; one can guess a biased decision and then use Theorem 5.5 to verify it. In order to show hardness we will reduce from the subset sum problem, which is well known to be NP-hard. Recall that the subset sum problem consists on, given natural numbers $s_1, \ldots, s_n, k \in \mathbb{N}$, to decide whether there is a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} s_i = k$. Let us proceed with the reduction. Based on a subset sum instance $s_1, \ldots, s_n, k$, we create a perceptron with $n$ unprotected features (that we assume to have indices 1 through $n$) with associated weights $s_1, \ldots, s_n$ and a single protected feature, with index $n+1$ and weight 1. Given the described weights, let $\mathcal{M}$ be the resulting perceptron that has those weights and bias[8] $-k-1$. The following claim is enough to establish the reduction.

**Claim 5.17** The perceptron $\mathcal{M}$ is biased if and only if $s_1, \ldots, s_n, k$ is a positive instance of the subset sum problem.

For the forward direction, consider $\mathcal{M}$ to be biased. That means there are instances $\boldsymbol{x}$ and $\boldsymbol{y}$ such that $\mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y})$, that differ only on the $n+1$-th feature, as it is the only protected one. Assume wlog that $\mathcal{M}(\boldsymbol{x}) = 1$ and $\mathcal{M}(\boldsymbol{y}) = 0$, by swapping the variables if it is not already the case. This implies $\langle \boldsymbol{w}, \boldsymbol{x} \rangle \geq k+1$ and $\langle \boldsymbol{w}, \boldsymbol{y} \rangle < k+1$. As $\langle \boldsymbol{w}, \boldsymbol{x} \rangle \geq \langle \boldsymbol{w}, \boldsymbol{y} \rangle$, and $\boldsymbol{x}$ differs from $\boldsymbol{y}$ only on the $n+1$-th feature, it must hold that $x_{n+1} = 1$ and $y_{n+1} = 1$, as $w_{n+1} = 1$. Let $P$ be the set of unprotected features of $\boldsymbol{x}$ (and thus $\boldsymbol{y}$) that are set to 1. Then we can write $\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \left(\sum_{i \in P} s_i\right) + 1$, as each weight $w_i$ was chosen to be equal to $s_i$. We thus have, considering $\boldsymbol{x}$ that $\left(\sum_{i \in P} s_i\right) + 1 \geq k+1$ and, by considering $\boldsymbol{y}$, that $\left(\sum_{i \in P} s_i\right) < k+1$, from which we deduce that $\sum_{i \in P} s_i = k$. We have found a subset of $\{s_1, \ldots, s_n\}$ that adds up to $k$, which is enough to conclude the forward direction of the proof. For the backward direction, consider an arbitrary set $P \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in P} s_i = k$. It is then easy to verify that the instance $\boldsymbol{x}$ that has a 1 in every feature whose index belongs to $P$, a 1 in the $n+1$-th feature, and 0 on the rest, is a positive instance of $\mathcal{M}$. Furthermore, $\boldsymbol{y}$ that differs from $\boldsymbol{x}$ only in the $n+1$-th feature can be checked to be a negative instance. As we have found a pair of instances that differ only on protected features, and yet have opposite classifications, the model $\mathcal{M}$ must be biased.

**Theorem 5.18** The ModelBias problem is NP-complete for MLPs.

Proof. Hardness is a trivial consequence of Theorem 5.16, as a perceptron is a particular case of an MLP. Membership comes from the fact that one can guess a pair of instances $\boldsymbol{x}, \boldsymbol{y}$ and then check in polynomial time both that they differ only on protected features and that they receive opposite classifications by $\mathcal{M}$. □

[8] Recall that the bias of a perceptron has nothing to do with the notion of bias that relates to fairness.

# Discussion

## A About our results

We have proposed a theoretical framework that allows to formally compare the interpretability of different classes of models. Our results show significant correlate with three common assumptions in the literature [6, 23, 29, 30, 33, 36, 41, 48, 50, 57]:

1. Tree-based and Rule-based models are more interpretable than MLPs
2. Linear models are more interpretable than MLPs
3. The highly nested and recursive structure of deep MLPs is what makes them difficult to interpret.

It is however interesting to recall that, as discussed in Chapter 4, our hardness results for MLPs apply even to very shallow ones. With hindsight, one can notice that all our basic hardness results derive from the ability of MLPs to compactly represent arbitrary boolean formulas. The ability of MLPs to learn a wide variety of functions is often used to argue their potential as general model for learning, and our results seem to suggest that a lack of interpretability is strictly related with this capacity for generalization.

The results concerning parameterized complexity in Chapter 4 suggest that, even with the goal of finding small explanations, that are of particular interest for practitioners, it is unlikely that we get to develop practical algorithms for such a task. The deeper the MLP, the more unlikely it will be to obtain a practical algorithm, according to traditional parameterized complexity assumptions.

On the other hand, our results in the complexity of bias detection, presented in Chapter 5, coupled with our results in the complexity of model interpretability, presented in Chapter 3, suggest that there is indeed a close relationship between our ability to interpret models and to detect and prevent bias in them, as proposed by Rudin et al. [48, 49].

Finally, it is relevant to stress once again, as pointed out by Miller [38], Lipton [36] and others, that interpretability is a complex concept, in constant dialogue with social sciences and depending in human behavior. Our work does not attempt to reduce such complex nature to a mathematical formalism, but rather to present a mathematical formalism that could serve to back up human perception, or have a better and more refined understanding of it. More precisely, the notion of *c-interpretability* proposed in our work does not aim to subsume that of interpretability, but to

support it, and even though our results are not to change, the concept of c-interpretability is to be revisited as the notion of interpretability dynamically changes.

# B Related work

Our work is tightly related to that of Darwiche et al. [13, 51, 52, 53, 54]. Their work presents definitions of different kind of explanations, for example in a recent article [13] they present the concept of *robustness*, to which we came independently as *minimum change required*. Furthermore, the name *(minimum/minimal)*[1] *sufficient reason* coined[2] by Darwiche and Hirth [13], and used throughout our work, comes to replace what was before called a *prime implicant explanation*, which relates the classical concept of a prime implicant in logic [55] with explanations for a model's decision. Throughout many different articles, Darwiche et al. explore how several particular cases of Binary Decision Diagrams allow for efficient solutions to certain queries. Moreover, they show how knowledge compilation allows for transforming Neurons/Neural Networks into equivalent BDDs improving on a naive approach.[3] Interestingly, the result presented in Lemma 3.5 establishes a barrier to knowledge compilation approaches to explainability based on BDDs by presenting a query that is hard to answer even over decision trees.

On the other hand, it is relevant to acknowledge the work of Ramaswamy [45], which studies the complexity of different problems associated with understanding the behavior of a particular kind of Neural Networks, more biologically inspired. The studied networks allow for cycles in their graph structure, and have a *temporal* behaviour, which makes them closer to *Recurrent Neural Networks*. It is interesting to notice that, while our work is mostly concerned with detecting subset of the input that are relevant for the decisions a model takes, the work of Ramaswamy considers the relevance of arbitrary nodes in networks.

# C Open problems and future research directions

A reasonable objection against the framework presented in this thesis is to note its dependency on the chosen queries. That is, it could be that the presented queries (some of which are not original to this work [13, 51, 53, 55]) allow for the fruitful results obtained in Chapter 3, but a different choice of queries would have made for a completely different story. A general way to address this concern is to consider a *logic* that subsumes not only the presented queries, but an infinite number of them. Then, by proving that answering queries in such a logic is harder for MLPs than it is for FBDDs or perceptrons one could get rid of the dependency on the queries, at the cost of depending on the components of the chosen logic. This is part of our ongoing work, and aims to extend the scope and generality of the results presented.

An interesting line of work, considering the widespread use of (deep) MLPs is that of getting a better characterization of easy and hard instances, that could allow for tractability results, or approximability results, which could be helpful for practitioners and researchers. For example,

[1] The qualifiers are added in our treatment for clarity. The work of Darwiche and Hirth uses the term *sufficient reason* always referring to what we call a *minimal sufficient reason*.

[2] To the best of our knowledge.

[3] Their runtimes are still exponential, but much better than a brute-force solution.

finding conditions on the weights, activation function, or number of non-zero connections that make the studied queries tractable could be an interesting step to make our work more applicable.

Another extension of our work would be to distinguish between different modern architectures for MLPs. For example, the work of Pérez et al. [44] explores the Turing Completeness of certain architectures assuming arbitrary precision for internal representations. It would be interesting to study whether in a fixed precision setting, or limited precision setting (as discussed in Chapter 4), certain interpretability queries could become undecidable.

Yet another direction is to study interpretability queries that aim to understand the impact of inner perceptrons inside an MLP. The idea of interpreting the internal representation of MLPs has been explored in the literature [6, 23], and could lead to interesting research directions.

Finally, expanding on broader formal characterizations of bias and fairness could allow for a more detailed study of the correlate between fairness and interpretability. Generalizing our results so they encompass more diverse forms and sources of bias is part of our ongoing work as well, and a promising research direction.

# Bibliography


[1] MIT takes down 80 Million Tiny Images data set due to racist and offensive content, url = https://venturebeat.com/2020/07/01/mit-takes-down-80-million-tiny-images-data-set-due-to-racist-and-offensive-content/, note = Accessed for the last time on 2020-07-05.

[2] How We Analyzed the COMPAS Recidivism Algorithm, url = https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm, note = Accessed for the last time on 2020-07-05.

[3] What a machine learning tool that turns Obama white can (and can't) tell us about AI bias, url = https://www.theverge.com/21298762/face-depixelizer-ai-machine-learning-tool-pulse-stylegan-obama-bias, note = Accessed for the last time on 2020-07-05.

[4] E. Allender. A note on the power of threshold circuits. In *30th Annual Symposium on Foundations of Computer Science*. IEEE, 1989.

[5] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[6] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[7] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. fairmlbook.org, 2019. `http://www.fairmlbook.org`.

[8] Gerardo Berbeglia and Geňa Hahn. Counting feasible solutions of the traveling salesman problem with pickups and deliveries is# P-complete. *Discrete Applied Mathematics*, 157(11):2541–2547, 2009.

[9] Or Biran and Courtenay V. Cotton. Explanation and Justification in Machine Learning : A Survey. 2017.

[10] Manuel Blum, Ashok K. Chandra, and Mark N. Wegman. Equivalence of Free Boolean Graphs can be Decided Probabilistically in Polynomial Time. *Inf. Process. Lett.*, 10:80–82, 1980.

[11] Jonathan F. Buss and Tarique Islam. Simplifying the weft hierarchy. *Theoretical Computer Science*, 351(3):303–313, 2006.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[13] Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. *arXiv preprint arXiv:2002.09284*, 2020.

[14] Finale Doshi-Velez and Been Kim. A Roadmap for a Rigorous Science of Interpretability. *CoRR*, abs/1702.08608, 2017.

[15] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer New York, 1999.

[16] Rod G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM Journal on Computing*, 24(4):873–921, August 1995.

[17] Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

[18] Rodney G. Downey, Michael R. Fellows, and Kenneth W. Regan. Parameterized circuit complexity and the W hierarchy. *Theoretical Computer Science*, 191(1-2):97–115, January 1998.

[19] Michael Fellows, Danny Hermelin, Moritz Müller, and Frances Rosamond. A purely democratic characterization of W[1]. In *Parameterized and Exact Computation*, pages 103–114. Springer Berlin Heidelberg.

[20] Michael R. Fellows, Jörg Flum, Danny Hermelin, Moritz Müller, and Frances A. Rosamond. Combinatorial circuits and the W-hierarchy. 2007.

[21] Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[22] Pratik Gajane and Mykola Pechenizkiy. On Formalizing Fairness in Prediction with Machine Learning, 2017.

[23] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2018.

[24] Mikael Goldmann and Marek Karpinski. Simulating threshold circuits by majority circuits. *SIAM Journal on Computing*, 27(1):230–246, 1998.

[25] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF minimization and isomorphism testing for monotone formulas. *Information and Computation*, 206(6):760–775, 2008.

[26] Bryce Goodman and Seth Flaxman. European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Magazine*, 38(3):50–57, October 2017.

[27] Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Štefankovic, Santosh Vempala, and Eric Vigoda. An FPTAS for #knapsack and related counting problems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 817–826. IEEE, 2011.

[28] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5).

[29] David Gunning and David Aha. DARPA's explainable artificial intelligence (XAI) program. *AI Magazine*, 40(2):44–58, 2019.

[30] Tameru Hailesilassie. Rule Extraction Algorithm for Deep Neural Networks: A Review. *ArXiv*, abs/1610.05267, 2016.

[31] Bernease Herman. The Promise and Peril of Human Evaluation for Model Interpretability, 2017.

[32] Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *TCS*, 43:169–188, 1986.

[33] Been Kim, Justin Gilmer, Fernanda Viegas, Ulfar Erlingsson, and Martin Wattenberg. TCAV: Relative concept importance testing with Linear Concept Activation Vectors. 11 2017.

[34] James Alexander King. *Approximation algorithms for guarding 1.5 dimensional terrains*. PhD thesis, 2005.

[35] Isaac Lage, Andrew Ross, Samuel J Gershman, Been Kim, and Finale Doshi-Velez. Human-in-the-Loop Interpretability Prior. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10159–10168. Curran Associates, Inc., 2018.

[36] Zachary C Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

[37] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A Survey on Bias and Fairness in Machine Learning, 2019.

[38] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, February 2019.

[39] Christoph Molnar. *Interpretable Machine Learning*. 2019. `https://christophm.github.io/interpretable-ml-book/`.

[40] W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.

[41] Tung D Nguyen, Kathryn E Kasmarik, and Hussein A Abbass. Towards interpretable deep neural networks: An exact transformation to multi-class multivariate decision trees. *arXiv*, pages arXiv–2003, 2020.

[42] Eirini Ntoutsi, Pavlos Fafalios, Ujwal Gadiraju, Vasileios Iosifidis, Wolfgang Nejdl, Maria-Esther Vidal, Salvatore Ruggieri, Franco Turini, Symeon Papadopoulos, Emmanouil Krasanakis, Ioannis Kompatsiaris, Katharina Kinder-Kurlanda, Claudia Wagner, Fariba Karimi, Miriam Fernandez, Harith Alani, Bettina Berendt, Tina Kruegel, Christian Heinze, Klaus Broelemann, Gjergji Kasneci, Thanassis Tiropanis, and Steffen Staab. Bias in data-driven artificial intelligence systems—An introductory survey. *WIREs Data Mining and Knowledge Discovery*, 10(3), February 2020.

[43] Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3(0):96–146, 2009.

[44] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*, 2019.

[45] Venkatakrishnan Ramaswamy. An Algorithmic Barrier to Neural Circuit Understanding. *bioRxiv*, 2019.

[46] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?". In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

[47] Romeo Rizzi and Alexandru I Tomescu. Faster FPTASes for counting and random generation of Knapsack solutions. *Information and Computation*, 267:135–144, 2019.

[48] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[49] Cynthia Rudin, Caroline Wang, and Beau Coker. The Age of Secrecy and Unfairness in Recidivism Prediction. *Harvard Data Science Review*, 2(1), 3 2020. https://hdsr.mitpress.mit.edu/pub/7z10o269.

[50] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. *ITU Journal: ICT Discoveries - Special Issue 1 - The Impact of Artificial Intelligence (AI) on Communication Networks and Services*, 1:1–10, 10 2017.

[51] Weijia Shi, Andy Shih, Adnan Darwiche, and Arthur Choi. On tractable representations of binary neural networks. *arXiv preprint arXiv:2004.02082*, 2020.

[52] Andy Shih, Arthur Choi, and Adnan Darwiche. Formal verification of Bayesian network classifiers. In *International Conference on Probabilistic Graphical Models*, pages 427–438, 2018.

[53] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining Bayesian network classifiers. *arXiv preprint arXiv:1805.03364*, 2018.

[54] Andy Shih, Adnan Darwiche, and Arthur Choi. Verifying binarized neural networks by Angluin-style learning. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 354–370. Springer, 2019.

[55] Christopher Umans. The minimum equivalent DNF problem and shortest implicants. *Journal*

*of Computer and System Sciences*, 63(4):597–611, 2001.

[56] Ingo Wegener. BDDs—design, analysis, complexity, and applications. *Discrete Applied Mathematics*, 138(1-2):229–251, 2004.

[57] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. DeepRED – Rule Extraction from Deep Neural Networks. In *Discovery Science*, pages 457–473. Springer International Publishing, 2016.