Query Languages for Neural Networks

Martin Grohe
RWTH Aachen University, Aachen, Germany

Christoph Standke
RWTH Aachen University, Aachen, Germany

Juno Steegmans
Data Science Institute, UHasselt, Diepenbeek, Belgium

Jan Van den Bussche
Data Science Institute, UHasselt, Diepenbeek, Belgium

Abstract

We lay the foundations for a database-inspired approach to interpreting and understanding neural network models by querying them using declarative languages. Towards this end we study different query languages, based on first-order logic, that mainly differ in their access to the neural network model. First-order logic over the reals naturally yields a language which views the network as a black box; only the input–output function defined by the network can be queried. This is essentially the approach of constraint query languages. On the other hand, a white-box language can be obtained by viewing the network as a weighted graph, and extending first-order logic with summation over weight terms. The latter approach is essentially an abstraction of SQL. In general, the two approaches are incomparable in expressive power, as we will show. Under natural circumstances, however, the white-box approach can subsume the black-box approach; this is our main result. We prove the result concretely for linear constraint queries over real functions definable by feedforward neural networks with a fixed number of hidden layers and piecewise linear activation functions.

2012 ACM Subject Classification Theory of computation → Database query languages (principles)

Keywords and phrases Expressive power of query languages, Machine learning models, languages for interpretability, explainable AI

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.9

Related Version *Full Version*: `https://arxiv.org/abs/2408.10362` [14]

Funding *Martin Grohe*: Funded by the European Union (ERC, SymSim, 101054974). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Christoph Standke: Funded by the German Research Foundation (DFG) under grants GR 1492/16-1 and GRK 2236 (UnRAVeL).
Juno Steegmans: Supported by the Special Research Fund (BOF) of UHasselt.
Jan Van den Bussche: Partially supported by the Flanders AI Program (FAIR).

1 Introduction

Neural networks [11] are a popular and successful representation model for real functions learned from data. Once deployed, the neural network is "queried" by supplying it with inputs then obtaining the outputs. In the field of databases, however, we have a much richer conception of querying than simply applying a function to given arguments. For example, in querying a database relation Employee(name, salary), we can not only ask for Anne's salary; we can also ask how many salaries are below that of Anne's; we can ask whether no two employees have the same salary; and so on.

In this paper, we consider the querying of neural networks from this more general perspective. We see many potential applications: obvious ones are in explanation, verification, and interpretability of neural networks and other machine-learning models [8, 2, 25]. These are huge areas [31, 7] where it is important [29, 22] to have formal, logical definitions for the myriad notions of explanation that are being considered. Another potential application is in managing machine-learning projects, where we are testing many different architectures and training datasets, leading to a large number of models, most of which become short-term legacy code. In such a context it would be useful if the data scientist could search the repository for earlier generated models having certain characteristics in their architecture or in their behavior, which were perhaps not duly documented.

The idea of querying machine learning models with an expressive, declarative query language comes naturally to database researchers, and indeed, Arenas et al. already proposed a language for querying boolean functions over an unbounded set of boolean features [3]. In the modal logic community, similar languages are being investigated [26, references therein].

In the present work, we focus on real, rather than boolean, functions and models, as is indeed natural in the setting of verifying neural networks [2].

The constraint query language approach. A natural language for querying real functions on a fixed number of arguments (features) is obtained by simply using first-order logic over the reals, with a function symbol F representing the function to be queried. We denote this by FO($\mathbf{R}$). For example, consider functions F with three arguments. The formula $\forall b' \, |F(a,b,c) - F(a,b',c)| < \epsilon$ expresses that the output on (a,b,c) does not depend strongly on the second feature, i.e., $F(a,b',c)$ is ϵ-close to $F(a,b,c)$ for any b'. Here, a, b, c and ϵ can be real constants or parameters (free variables).

The language FO($\mathbf{R}$) (also known as FO + Poly) and its restriction FO($\mathbf{R}_{\text{lin}}$) to linear arithmetic (aka FO + Lin) were intensively investigated in database theory around the turn of the century, under the heading of *constraint query languages*, with applications to spatial and temporal databases. See the compendium volume [21] and book chapters [24, chapter 13], [13, chapter 5]. Linear formulas with only universal quantifiers over the reals, in front of a quantifier-free condition involving only linear arithmetic (as the above example formula), can already model many properties considered in the verification of neural networks [2]. This universal fragment of FO($\mathbf{R}_{\text{lin}}$) can be evaluated using linear programming techniques [2].

Full FO($\mathbf{R}$) allows alternation of quantifiers over the reals, and multiplication in arithmetic. Because the first-order theory of the reals is decidable [5], FO($\mathbf{R}$) queries can still be effectively evaluated on any function that is *semi-algebraic*, i.e., itself definable in first-order logic over the reals. Although the complexity of this theory is high, if the function is presented as a quantifier-free formula, FO($\mathbf{R}$) query evaluation actually has polynomial-time *data* complexity; here, the "data" consists of the given quantifier-free formula [18].

Functions that can be represented by feedforward neural networks with ReLU hidden units and linear output units are clearly semi-algebraic; in fact, they are piecewise linear. For most of our results, we will indeed focus on this class of networks, which are widespread in practice [11], and denote them by ReLU-FNN.

The SQL approach. Another natural approach to querying neural networks is to query them directly, as graphs of neurons with weights on the nodes and edges. For this purpose one represents such graphs as relational structures with numerical values and uses SQL to query them. As an abstraction of this approach, in this paper, we model neural networks

as weighted finite structures. As a query language we use FO(SUM): first-order logic over weighted structures, allowing order comparisons between weight terms, where weight terms can be built up using rational arithmetic, if-then-else, and, importantly, summation.

Based on logics originally introduced by Grädel and Gurevich [12], the language FO(SUM) is comparable to the relational calculus with aggregates [19] and, thus, to SQL [23]. Logics close to FO(SUM), but involving arithmetic in different semirings, were recently also used for unifying different algorithmic problems in query processing [35], as well as for expressing hypotheses in the context of learning over structures [36]. The well-known FAQ framework [28], restricted to the real semiring, can be seen as the conjunctive fragment of FO(SUM).

To give a simple example of an FO(SUM) formula, consider ReLU-FNNs with a single input unit, one hidden layer of ReLU units, and a single linear output unit. The following formula expresses the query that asks if the function evaluation on a given input value is positive:

$$0 < b(\mathrm{out}) + \sum_{x:E(\mathrm{in},x)} w(x,\mathrm{out}) \cdot \mathrm{ReLU}(w(\mathrm{in},x) \cdot \mathit{val} + b(x)).$$

Here, E is the edge relation between neurons, and constants in and out hold the input and output unit, respectively. Thus, variable x ranges over the neurons in the hidden layer. Weight functions w and b indicate the weights of edges and the biases of units, respectively; the weight constant *val* stands for a given input value. We assume for clarity that ReLU is given, but it is definable in FO(SUM).

Just like the relational calculus with aggregates, or SQL select statements, query evaluation for FO(SUM) has polynomial time data complexity, and techniques for query processing and optimization from database systems directly apply.

Comparing expressive powers. Expressive power of query languages has been a classical topic in database theory and finite model theory [1, 24], so, with the advent of new models, it is natural to revisit questions concerning expressivity. The goal of this paper is to understand and compare the expressive power of the two query languages FO($\mathbf{R}$) and FO(SUM) on neural networks over the reals. The two languages are quite different. FO($\mathbf{R}$) sees the model as a black-box function F, but can quantify over the reals. FO(SUM) can see the model as a white box, a finite weighted structure, but can quantify only over the elements of the structure, i.e., the neurons.

In general, indeed the two expressive powers are incomparable. In FO(SUM), we can express queries about the network topology; for example, we may ask to return the hidden units that do not contribute much to the function evaluation on a given input value. (Formally, leaving them out of the network would yield an output within some ϵ of the original output.) Or, we may ask whether there are more than a million neurons in the first hidden layer. For FO($\mathbf{R}$), being a black box language, such queries are obviously out of scope.

A more interesting question is how the two languages compare in expressing *model agnostic* queries: these are queries that return the same result on any two neural networks that represent the same input–output function. For example, when restricting attention to networks with one hidden layer, the example FO(SUM) formula seen earlier, which evaluates the network, is model agnostic. FO($\mathbf{R}$) is model agnostic by design, and, indeed, serves as a very natural declarative benchmark of expressiveness for model-agnostic queries. It turns out that FO(SUM), still restricting to networks of some fixed depth, can express model-agnostic queries that FO($\mathbf{R}$) cannot. For example, for any fixed depth d, we will show that FO(SUM) can express the integrals of a functions given by a ReLU-FNNs of depth d. In contrast, we will show that this cannot be done in FO($\mathbf{R}$) (Theorem 6.1).

The depth of a neural network can be taken as a crude notion of "schema". Standard relational query languages typically cannot be used without knowledge of the schema of the data. Similarly, we will show that without knowledge of the depth, FO(SUM) cannot express any nontrivial model-agnostic query (Theorem 6.2). Indeed, since FO(SUM) lacks recursion, function evaluation can only be expressed if we known the depth. (Extensions with recursion is one of the many interesting directions for further research.)

When the depth is known, however, for model-agnostic queries, the expressiveness of FO(SUM) exceeds the benchmark of expressiveness provided by FO($\mathbf{R}_{\text{lin}}$). Specifically, we show that every FO($\mathbf{R}_{\text{lin}}$) query over functions representable by ReLU-FNNs is also expressible in FO(SUM) evaluated on the networks directly (Theorem 7.1). This is our main technical result, and can be paraphrased as *"SQL can verify neural networks."* The proof involves showing that the required manipulations of higher-dimensional piecewise linear functions, and the construction of cylindrical cell decompositions in $\mathbb{R}^n$, can all be expressed in FO(SUM). To allow for a modular proof, we also develop the notion of FO(SUM) translation, generalizing the classical notion of first-order interpretations [16].

This paper is organized as follows. Section 2 provides preliminaries on neural networks. Section 3 introduces FO($\mathbf{R}$). Section 4 introduces weighted structures and FO(SUM), after which Section 5 introduces white-box querying. Section 6 considers model-agnostic queries. Section 7 presents the main technical result. Section 8 concludes with a discussion of topics for further research.

A full version of this paper with full proofs is available [14].

2 Preliminaries on neural networks

A feedforward neural network [11], in general, could be defined as a finite, directed, weighted, acyclic graph, with some additional aspects which we discuss next. The nodes are also referred to as *neurons* or *units*. Some of the source nodes are designated as *inputs*, and some of the sink nodes are designated as *outputs*. Both the inputs, and the outputs, are linearly ordered. Neurons that are neither inputs nor outputs are said to be *hidden*. All nodes, except for the inputs, carry a weight, a real value, called the *bias*. All directed edges also carry a weight.

In this paper, we focus on *ReLU-FNNs*: networks with ReLU activations and linear outputs. This means the following. Let $\mathcal{N}$ be a neural network with m inputs. Then every node u in $\mathcal{N}$ represents a function $F_u^{\mathcal{N}} : \mathbb{R}^m \to \mathbb{R}$ defined as follows. We proceed inductively based on some topological ordering of $\mathcal{N}$. For input nodes u, simply $F_u^{\mathcal{N}}(x_1, \ldots, x_m) := x_i$, if u is the ith input node. Now let u be a hidden neuron and assume $F_v^{\mathcal{N}}$ is already defined for all *predecessors* v of u, i.e., nodes v with an edge to u. Let $v_1, \ldots, v_l$ be these predecessors, let $w_1, \ldots, w_l$ be the weights on the respective edges, and let b be the bias of u. Then

$$F_u^{\mathcal{N}}(\boldsymbol{x}) := \text{ReLU}(b + \sum_i w_i F_{v_i}^{\mathcal{N}}(\boldsymbol{x})),$$

where $\text{ReLU} : \mathbb{R} \to \mathbb{R} : z \mapsto \max(0, z)$.

Finally, for an output node u, we define $F_u^{\mathcal{N}}$ similarly to hidden neurons, except that the application of ReLU is omitted. The upshot is that a neural network $\mathcal{N}$ with m inputs and n outputs $u_1, \ldots, u_n$ represents a function $F^{\mathcal{N}} : \mathbb{R}^m \to \mathbb{R}^n$ mapping $\boldsymbol{x}$ to $(F_{u_1}^{\mathcal{N}}(\boldsymbol{x}), \ldots, F_{u_n}^{\mathcal{N}}(\boldsymbol{x}))$. For any node u in the network, $F_u^{\mathcal{N}}$ is always a continuous piecewise linear function. We denote the class of all continuous piecewise linear functions $F : \mathbb{R}^m \to \mathbb{R}$ by $\mathcal{PL}(m)$; that is, continuous functions F that admit a partition of $\mathbb{R}^m$ into finitely many polytopes such that F is affine linear on each of them.

Hidden layers. Commonly, the hidden neurons are organized in disjoint blocks called *layers*. The layers are ordered, such that the neurons in the first layer have only edges from inputs, and the neurons in any later layer have only edges from neurons in the previous layer. Finally, outputs have only edges from neurons in the last layer.

We will use $\mathbf{F}(m, \ell)$ to denote the class of layered networks with m inputs of depth ℓ, that is, with an input layer with m nodes, $\ell - 1$ hidden layers, and an output layer with a single node. Recall that the nodes on all hidden layer use ReLU activations and the output node uses the identity function.

It is easy to see that networks in $\mathbf{F}(1, 1)$ just compute linear functions and that for every $\ell \geq 2$ we have $\{F^{\mathcal{N}} \mid \mathcal{N} \in \mathbf{F}(1, \ell)\} = \mathcal{PL}(1)$, that is, the class of functions $\mathbb{R} \to \mathbb{R}$ that can be computed by a network in $\mathbf{F}(1, \ell)$ is the class of all continuous piecewise linear functions. The well-known *Universal Approximation Theorem* [9, 17] says that every continuous function $f : K \to \mathbb{R}$ defined on a compact domain $K \subseteq \mathbb{R}^m$ can be approximated to any additive error by a network in $\mathbf{F}(m, 2)$.

3 A black-box query language

First-order logic over the reals, denoted here by $\mathrm{FO}(\mathbf{R})$, is, syntactically, just first-order logic over the vocabulary of elementary arithmetic, i.e., with binary function symbols $+$ and $\cdot$ for addition and multiplication, binary predicate $<$, and constant symbols 0 and 1 [5]. Constants for rational numbers, or even algebraic numbers, can be added as an abbreviation (since they are definable in the logic).

The fragment $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}})$ of *linear* formulas uses multiplication only for scalar multiplication, i.e., multiplication of variables with rational number constants. For example, the formula $y = 3x_1 - 4x_2 + 7$ is linear, but the formula $y = 5x_1 \cdot x_2 - 3$ is not. In practice, linear queries are often sufficiently expressive, both from earlier applications for temporal or spatial data [21], as well as for querying neural networks (see examples to follow). The only caveat is that many applications assume a distance function on vectors. When using distances based on absolute value differences between real numbers, e.g., the Manhattan distance or the max norm, we still fall within $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}})$.

We will add to $\mathrm{FO}(\mathbf{R})$ extra relation or function symbols; in this paper, we will mainly consider $\mathrm{FO}(\mathbf{R}, F)$, which is $\mathrm{FO}(\mathbf{R})$ with an extra function symbol F. The structure on the domain $\mathbb{R}$ of reals, with the arithmetic symbols having their obvious interpretation, will be denoted here by $\mathbf{R}$. Semantically, for any vocabulary τ of extra relation and function symbols, $\mathrm{FO}(\mathbf{R}, \tau)$ formulas are interpreted over structures that expand $\mathbf{R}$ with additional relations and functions on $\mathbb{R}$ of the right arities, that interpret the symbols in τ. In this way, $\mathrm{FO}(\mathbf{R}, F)$ expresses queries about functions $F \colon \mathbb{R}^m \to \mathbb{R}$.

This language can express a wide variety of properties (queries) considered in interpretable machine learning and neural-network verification. Let us see some examples.

▶ **Example 3.1.** To check whether $F \colon \mathbb{R}^m \to \mathbb{R}$ is robust around an m-vector $\boldsymbol{a}$ [32], using parameters ϵ and δ, we can write the formula $\forall \boldsymbol{x}(d(\boldsymbol{x}, \boldsymbol{a}) < \epsilon \Rightarrow |F(\boldsymbol{x}) - F(\boldsymbol{a})| < \delta)$. Here $\boldsymbol{x}$ stands for a tuple of m variables, and d stands for some distance function which is assumed to be expressible.

▶ **Example 3.2.** Counterfactual explanation methods [37] aim to find the closest $\boldsymbol{x}$ to an input $\boldsymbol{a}$ such that $F(\boldsymbol{x})$ is "expected," assuming that $F(\boldsymbol{a})$ was unexpected. A typical example is credit denial; what should we change minimally to be granted credit? Typically we can define expectedness by some formula, e.g., $F(\boldsymbol{x}) > 0.9$. Then we can express the counterfactual explanation as $F(\boldsymbol{x}) > 0.9 \wedge \forall \boldsymbol{y}(F(\boldsymbol{y}) > 0.9 \Rightarrow d(\boldsymbol{x}, \boldsymbol{a}) \leq d(\boldsymbol{y}, \boldsymbol{a}))$.

▶ **Example 3.3.** We may define the contribution of an input feature i on an input $\boldsymbol{a} = (a_1, \ldots, a_m)$ as the inverse of the smallest change we have to make to that feature for the output to change significantly. We can express that r is such a change by writing (taking $i = 1$ for clarity) $r > 0 \wedge (d(F(a_1 - r, a_2, \ldots, a_m), F(\boldsymbol{a})) > \epsilon \vee d(F(a_1 + r, a_2, \ldots, a_m), F(\boldsymbol{a})) > \epsilon)$. Denoting this formula by $\mathit{change}(r)$, the smallest change is then expressed as $\mathit{change}(r) \wedge \forall r'(\mathit{change}(r') \Rightarrow r \leq r')$.

▶ **Example 3.4.** We finally illustrate that $\mathrm{FO}(\mathbf{R}, F)$ can express gradients and many other notions from calculus. For simplicity assume F to be unary. Consider the definition $F'(a) = \lim_{x \to c}(F(x) - F(c))/(x - c)$ of the derivative in a point c. So it suffices to show how to express that $l = \lim_{x \to c} G(x)$ for a function G that is continuous in c. We can write down the textbook definition literally as $\forall \epsilon > 0\, \exists \delta > 0\, \forall x(|x - c| < \delta \Rightarrow |G(x) - l| < \epsilon)$.

Evaluating FO(R) queries. Black box queries can be effectively evaluated using the decidability and quantifier elimination properties of $\mathrm{FO}(\mathbf{R})$. This is the constraint query language approach [18, 21], which we briefly recall next.

A function $f\colon \mathbb{R}^m \to \mathbb{R}$ is called semialgebraic [5] (or semilinear) if there exists an $\mathrm{FO}(\mathbf{R})$ (or $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}})$) formula $\varphi(x_1, \ldots, x_m, y)$ such that for any m-vector $\boldsymbol{a}$ and real value b, we have $\mathbf{R} \models \varphi(\boldsymbol{a}, b)$ if and only if $F(\boldsymbol{a}) = b$.

Now consider the task of evaluating an $\mathrm{FO}(\mathbf{R}, F)$ formula ψ on a semialgebraic function f, given by a defining formula φ. By introducing auxiliary variables, we may assume that the function symbol F is used in ψ only in subformulas for the form $z = F(u_1, \ldots, u_m)$. Then replace in ψ each such subformula by $\varphi(u_1, \ldots, u_m, z)$, obtaining a pure $\mathrm{FO}(\mathbf{R})$ formula χ.

Now famously, the first-order theory of $\mathbb{R}$ is decidable [33, 5]. In other words, there is an algorithm that decides, for any $\mathrm{FO}(\mathbf{R})$ formula $\chi(x_1, \ldots, x_k)$ and k-vector $\boldsymbol{c}$, whether $\mathbf{R} \models \chi(\boldsymbol{c})$. Actually, a stronger property holds, to the effect that every $\mathrm{FO}(\mathbf{R})$-formula is equivalent to a quantifier-free formula. The upshot is that there is an algorithm that, given a $\mathrm{FO}(\mathbf{R}, F)$ query $\psi(x_1, \ldots, x_k)$ and a semialgebraic function f given by a defining formula, outputs a quantifier-free formula defining the result set $\{\boldsymbol{c} \in \mathbb{R}^k \mid \mathbf{R}, f \models \psi(\boldsymbol{c})\}$. If f is given by a quantifier-free formula, the evaluation can be done in polynomial time in the length of the description of f, so polynomial-time data complexity. This is because there are algorithms for quantifier elimination with complexity $p(n) \cdot e(q)$, where n is the size of the formula, p is a polynomial, q is the number of quantifiers, and e is a doubly exponential function [18, 5].

Complexity. Of course, we want to evaluate queries on the functions represented by neural networks. From the definition given in Section 2, it is clear that the functions representable by ReLU-FNNs are always semialgebraic (actually, semilinear). For every output feature j, it is straightforward to compile, from the network, a quantifier-free formula defining the jth output component function. In this way we see that $\mathrm{FO}(\mathbf{R}, F)$ queries on ReLU-FNNs are, in principle, computable in polynomial time.

However, the algorithms are notoriously complex, and we stress again that $\mathrm{FO}(\mathbf{R}, F)$ should be mostly seen as a *declarative benchmark* of expressiveness. Moreover, we assume here for convenience that ReLU is a primitive function. ReLU can be expressed in $\mathrm{FO}(\mathbf{R})$ using disjunction, but this may blow up the query formula, e.g., when converting to disjunctive normal form [2]. Symbolic constraint solving algorithms for the reals have been extended to deal with ReLU natively [2].

▶ Remark 3.5. In closing this Section we remark that, to query the entire network function, we would not strictly use just only a single function symbol F, but rather the language $\mathrm{FO}(\mathbf{R}, F_1, \ldots, F_n)$, with function symbols for the n outputs. In this paper, for the sake of clarity, we will often stick to a single output, but our treatment generalizes to multiple outputs.

4 Weighted structures and FO(SUM)

Weighted structures are standard abstract structures equipped with one or more weight functions from tuples of domain elements to values from some separate, numerical domain. Here, as numerical domain, we will use $\mathbb{R}_\perp = \mathbb{R} \cup \{\perp\}$, the set of "lifted reals" where $\perp$ is an extra element representing an undefined value. Neural networks are weighted graph structures. Hence, since we are interested in declarative query languages for neural networks, we are interested in logics over weighted structures. Such logics were introduced by Grädel and Gurevich [12]. We consider here a concrete instantiation of their approach, which we denote by FO(SUM).

Recall that a (finite, relational) vocabulary is a finite set of function symbols and relation symbols, where each symbol comes with an arity (a natural number). We extend the notion of vocabulary to also include a number of *weight function symbols*, again with associated arities. We allow 0-ary weight function symbols, which we call *weight constant symbols*.

A (finite) *structure* $\mathcal{A}$ over such a vocabulary Υ consists of a finite domain A, and functions and relations on A of the right arities, interpreting the standard function symbols and relation symbols from Υ. So far this is standard. Now additionally, $\mathcal{A}$ interprets every weight function symbol w, of arity k, by a function $w^{\mathcal{A}} : A^k \to \mathbb{R}_\perp$.

The syntax of FO(SUM) formulas (over some vocabulary) is defined exactly as for standard first order logic, with one important extension. In addition to *formulas* (taking Boolean values) and *standard terms* (taking values in the structure), the logic contains *weight terms* taking values in $\mathbb{R}_\perp$. Weight terms t are defined by the following grammar:

$$t ::= \perp \mid w(s_1, \ldots, s_n) \mid r(t, \ldots, t) \mid \mathsf{if}\ \varphi\ \mathsf{then}\ t\ \mathsf{else}\ t \mid \sum_{\boldsymbol{x}:\varphi} t$$

Here, w is a weight function symbol of arity n and the s_i are standard terms; r is a rational function applied to weight terms, with rational coefficients; φ is a formula; and $\boldsymbol{x}$ is a tuple of variables. The syntax of weight terms and formulas is mutually recursive. As just seen, the syntax of formulas φ is used in the syntax of weight terms; conversely, weight terms t_1 and t_2 can be combined to form formulas $t_1 = t_2$ and $t_1 < t_2$.

Recall that a rational function is a fraction between two polynomials. Thus, the arithmetic operations that we consider are addition, scalar multiplication by a rational number, multiplication, and division.

The *free variables* of a weight term are defined as follows. The weight term $\perp$ has no free variables. The free variables of $w(s_1, \ldots, s_n)$ are simply the variables occurring in the s_i. A variable occurs free in $r(t_1, \ldots, t_n)$ if it occurs free in some t_i. A variable occurs free in 'if φ then t_1 else t_2' if it occurs free in t_1, t_2, or φ. The free variables of $\sum_{\boldsymbol{x}:\varphi} t$ are those of φ and t, except for the variables in $\boldsymbol{x}$. A formula or (weight) term is *closed* if it has no free variables.

We can evaluate a weight term $t(x_1, \ldots, x_k)$ on a structure $\mathcal{A}$ and a tuple $\boldsymbol{a} \in A^k$ providing values to the free variables. The result of the evaluation, denoted by $t^{\mathcal{A},\boldsymbol{a}}$, is a value in $\mathbb{R}_\perp$, defined in the obvious manner. In particular, when t is of the form $\sum_{\boldsymbol{y}:\varphi} t'$, we have

$$t^{\mathcal{A},\boldsymbol{a}} = \sum_{\boldsymbol{b}:\mathcal{A}\models\varphi(\boldsymbol{a},\boldsymbol{b})} t'^{\mathcal{A},\boldsymbol{a},\boldsymbol{b}}.$$

Division by zero, which can happen when evaluating terms of the form $r(t, \ldots, t)$, is given the value $\perp$. The arithmetical operations are extended so that $x + \perp$, $q\perp$ (scalar multiply), $x \cdot \perp$, and $x/\perp$ and $\perp/x$ always equal $\perp$. Also, $\perp < a$ holds for all $a \in \mathbb{R}$.

5 White-box querying

For any natural numbers m and n, we introduce a vocabulary for neural networks with m inputs and n outputs. We denote this vocabulary by $\Upsilon^{\text{net}}(m, n)$, or just Υ^{net} if m and n are understood. It has a binary relation symbol E for the edges; constant symbols $\text{in}_1, \ldots, \text{in}_m$ and $\text{out}_1, \ldots, \text{out}_n$ for the input and output nodes; a unary weight function b for the biases, and a binary weight function symbol w for the weights on the edges.

Any ReLU-FNN $\mathcal{N}$, being a weighted graph, is an Υ^{net}-structure in the obvious way. When there is no edge from node u_1 to u_2, we put $w^{\mathcal{N}}(u_1, u_2) = 0$. Since inputs have no bias, we put $b^{\mathcal{N}}(u) = \perp$ for any input u.

Depending on the application, we may want to enlarge Υ^{net} with some additional parameters. For example, we can use additional weight constant symbols to provide input values to be evaluated, or output values to be compared with, or interval bounds, etc.

The logic FO(SUM) over the vocabulary Υ^{net} (possibly enlarged as just mentioned) serves as a "white-box" query language for neural networks, since the entire model is given and can be directly queried, just like an SQL query can be evaluated on a given relational database. Contrast this with the language $\text{FO}(\mathbf{R}, F)$ from Section 3, which only has access to the function F represented by the network, as a black box.

▶ **Example 5.1.** While the language $\text{FO}(\mathbf{R}, F)$ cannot see inside the model, at least it has direct access to the function represented by the model. When we use the language FO(SUM), we must compute this function ourselves. At least when we know the depth of the network, this is indeed easy. In the Introduction, we already showed a weight term expressing the evaluation of a one-layer neural network on a single input and output. We can easily generalize this to a weight term expressing the value of any of a fixed number of outputs, with any fixed number m of inputs, and any fixed number of layers. Let $val_1, \ldots, val_m$ be additional weight constant symbols representing input values. Then the weight term $\text{ReLU}(b(u) + w(\text{in}_1, u) \cdot val_1 + \cdots + w(\text{in}_m, u) \cdot val_m)$ expresses the value of any neuron u in the first hidden layer (u is a variable). Denote this term by $t_1(u)$. Next, for any subsequent layer numbered $l > 1$, we inductively define the weight term $t_l(u)$ as

$$\text{ReLU}(b(u) + \sum_{x:E(x,u)} w(x, u) \cdot t_{l-1}(x)).$$

Here, $\text{ReLU}(c)$ can be taken to be the weight term $\mathsf{if}\ c > 0\ \mathsf{then}\ c\ \mathsf{else}\ 0$. Finally, the value of the jth output is given by the weight term $eval_j := b(\text{out}_j) + \sum_{x:E(x,\text{out}_j)} w(x, \text{out}_j) \cdot t_l(x)$, where l is the number of the last hidden layer.

▶ **Example 5.2.** We can also look for useless neurons: neurons that can be removed from the network without altering the output too much on given values. Recall the weight term $eval_j$ from the previous example; for clarity we just write $eval$. Let z be a fresh variable, and let $eval'$ be the term obtained from $eval$ by altering the summing conditions $E(x, u)$ and $E(x, \text{out})$ by adding the conjunct $x \neq z$. Then the formula $|eval - eval'| < \epsilon$ expresses that z is useless. (For $|c|$ we can take the weight term $\mathsf{if}\ c > 0\ \mathsf{then}\ c\ \mathsf{else}\ -c$.)

Another interesting example is computing integrals. Recall that $\mathbf{F}(m, \ell)$ is the class of networks with m inputs, one output, and depth ℓ.

▶ **Lemma 5.3.** *Let m and ℓ be natural numbers. There exists an* FO(SUM) *term t over $\Upsilon^{\text{net}}(m,1)$ with m additional pairs of weight constant symbols min_i and max_i for $i \in \{1,\dots,m\}$, such that for any network $\mathcal{N}$ in $\mathbf{F}(m,\ell)$, and values a_i and b_i for the min_i and max_i, we have $t^{\mathcal{N},a_1,b_1,\dots,a_m,b_m} = \int_{a_1}^{b_1} \cdots \int_{a_m}^{b_m} F^{\mathcal{N}}\, dx_1 \dots dx_m$.*

Proof (sketch). We sketch here a self-contained and elementary proof for $m = 1$ and $\ell = 2$ (one input, one hidden layer). This case already covers all continuous piecewise linear functions $\mathbb{R} \to \mathbb{R}$.

Every hidden neuron u may represent a "quasi breakpoint" in the piecewise linear function (that is, a point where its slope may change). Concretely, we consider the hidden neurons with nonzero input weights to avoid dividing by zero. Its x-coordinate is given by the weight term $break_x(u) := -b(u)/w(\text{in}_1, u)$. The y-value at the breakpoint is then given by $break_y(u) := eval_1(break_x(u))$, where $eval_1$ is the weight term from Example 5.1 and we substitute $break_x(u)$ for val_1.

Pairs (u_1, u_2) of neurons representing successive breakpoints are easy to define by a formula $succ(u_1, u_2)$. Such pairs represent the pieces of the function, except for the very first and very last pieces. For this proof sketch, assume we simply want the integral between the first breakpoint and the last breakpoint.

The area (positive or negative) contributed to the integral by the piece (u_1, u_2) is easy to write as a weight term: $area(u_1, u_2) = \frac{1}{2}(break_y(u_1) + break_y(u_2))(break_x(u_2) - break_x(u_1))$. We sum these to obtain the desired integral. However, since different neurons may represent the same quasi breakpoint, we must divide by the number of duplicates. Hence, our desired term t equals $\sum_{u_1,u_2:succ(u_1,u_2)} area(u_1,u_2)/(\sum_{u_1',u_2':\gamma} 1)$, where γ is the formula $succ(u_1', u_2') \wedge break_x(u_1') = break_x(u_1) \wedge break_x(u_2') = break_x(u_2)$. ◀

▶ **Example 5.4.** A popular alternative to Example 3.3 for measuring the contribution of an input feature i to an input $\boldsymbol{y} = (y_1, \dots, y_m)$ is the Shap score [27]. It assumes a probability distribution $\mathbb{P}$ on the input space and quantifies the change to the expected value of $F^{\mathcal{N}}$ caused by fixing input feature i to y_i in a random fixation order of the input features:

$$\text{Shap}(i) = \sum_{I \subseteq \{1,\dots,m\} \setminus \{i\}} \frac{|I|!(m-1-|I|!)}{m!} \Big(\mathbb{E}\big(F^{\mathcal{N}}(\boldsymbol{x}) \mid \boldsymbol{x}_{I\cup\{i\}} = \boldsymbol{y}_{I\cup\{i\}}\big) - \mathbb{E}\big(F^{\mathcal{N}}(\boldsymbol{x}) \mid \boldsymbol{x}_I = \boldsymbol{y}_I\big) \Big).$$

When we assume that $\mathbb{P}$ is the product of uniform distributions over the intervals (a_j, b_j), we can write the conditional expectation $\mathbb{E}\big(F^{\mathcal{N}}(\boldsymbol{x}) \mid \boldsymbol{x}_J = \boldsymbol{y}_J\big)$ for some $J \subseteq \{1,\dots,m\}$ by setting $\{1,\dots m\} \setminus J =: \{j_1,\dots,j_r\}$ as follows.

$$\mathbb{E}\big(F^{\mathcal{N}}(\boldsymbol{x}) \mid \boldsymbol{x}_J = \boldsymbol{y}_J\big) = \prod_{k=1}^{r} \frac{1}{b_{j_k} - a_{j_k}} \cdot \int_{a_{j_1}}^{b_{j_1}} \cdots \int_{a_{j_r}}^{b_{j_r}} F^{\mathcal{N}}(\boldsymbol{x}|_{\boldsymbol{x}_J = \boldsymbol{y}_J})\, dx_{j_r} \dots dx_{j_1}$$

where $\boldsymbol{x}|_{\boldsymbol{x}_J=\boldsymbol{y}_J}$ is a short notation for the variable obtained from $\boldsymbol{x}$ by replacing x_j with y_j for all $j \in J$. With lemma 5.3, this conditional expectation can be expressed in FO(SUM) and by replacing J with I or $I \cup \{i\}$ respectively, we can express the Shap score.

More examples. Our main result will be that, over networks of a given depth, all of $\text{FO}(\mathbf{R}_{\text{lin}}, F)$ can be expressed in FO(SUM). So the examples from Section 3 (which are linear if a Manhattan or max distance is used) apply here as well. Moreover, the techniques by which we show our main result readily adapt to queries not about the final function F represented by the network, but about the function F_z represented by a neuron z given as a

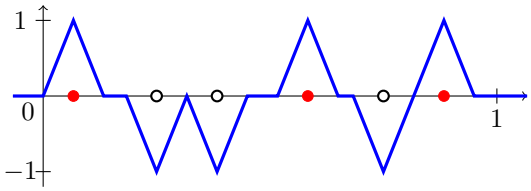

Figure 1 The function f_{S_1,S_2} of the proof of Theorem 6.1 for the set S_1 consisting of the three red points and the set S_2 consisting of the three white points.

parameter to the query, much as in Example 5.2. For example, in feature visualization [8] we want to find the input that maximizes the activation of some neuron z. Since this is expressible in $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}}, F)$, it is also expressible in FO(SUM).

6 Model-agnostic queries

We have already indicated that $\mathrm{FO}(\mathbf{R}, F)$ is "black box" while FO(SUM) is "white box". Black-box queries are commonly called *model agnostic* [8]. Some FO(SUM) queries may, and others may not, be model agnostic.

Formally, for some $\ell \geq 1$, let us call a closed FO(SUM) formula φ, possibly using weight constants $c_1, \ldots, c_k$, *depth-ℓ model agnostic* if for all $m \geq 1$ all neural networks $\mathcal{N}, \mathcal{N}' \in \bigcup_{i=1}^{\ell} \mathbf{F}(m, i)$ such that $F^{\mathcal{N}} = F^{\mathcal{N}'}$, and all $a_1, \ldots, a_k \in \mathbb{R}$ we have $\mathcal{N}, a_1, \ldots, a_k \models \varphi \Leftrightarrow \mathcal{N}', a_1, \ldots, a_k \models \varphi$. A similar definition applies to closed FO(SUM) weight terms.

For example, the term of Example 5.1 evaluating the function of a neural network of depth at most ℓ is depth-ℓ model agnostic. By comparison the formula stating that a network has useless neurons (cf. Example 5.2) is not model agnostic. The term t from Lemma 5.3, computing the integral, is depth-ℓ model agnostic.

▶ **Theorem 6.1.** *The query $\int_0^1 f = 0$ for functions $f \in \mathcal{PL}(1)$ is expressible by a depth-2 agnostic* FO(SUM) *formula, but not in* $\mathrm{FO}(\mathbf{R}, F)$.

Proof. We have already seen the expressibility in FO(SUM). We prove nonexpressibility in $\mathrm{FO}(\mathbf{R}, F)$.

Consider the equal-cardinality query Q about disjoint pairs (S_1, S_2) of finite sets of reals, asking whether $|S_1| = |S_2|$. Over *abstract* ordered finite structures, equal cardinality is well-known not to be expressible in order-invariant first-order logic [24]. Hence, by the generic collapse theorem for constraint query languages over the reals [21, 24], query Q is not expressible in $\mathrm{FO}(\mathbf{R}, S_1, S_2)$.

Now for any given S_1 and S_2, we construct a continuous piecewise linear function f_{S_1,S_2} as follows. We first apply a suitable affine transformation so that $S_1 \cup S_2$ falls within the open interval $(0, 1)$. Now f_{S_1,S_2} is a sawtooth-like function, with positive teeth at elements from S_1, negative teeth (of the same height, say 1) at elements from S_2, and zero everywhere else. To avoid teeth that overlap the zero boundary at the left or that overlap each other, we make them of width $\min\{m, M\}/2$, where m is the minimum of $S_1 \cup S_2$ and M is the minimum distance between any two distinct elements in $S_1 \cup S_2$.

Expressing the above construction uniformly in $\mathrm{FO}(\mathbf{R}, S_1, S_2)$ poses no difficulties; let $\psi(x, y)$ be a formula defining f_{S_1,S_2}. Now assume, for the sake of contradiction, that $\int_0^1 F = 0$ would be expressible by a closed $\mathrm{FO}(\mathbf{R}, F)$ formula φ. Then composing φ with ψ would express query Q in $\mathrm{FO}(\mathbf{R}, S_1, S_2)$. Indeed, clearly, $\int_0^1 f_{S_1,S_2} = 0$ if and only if $|S_1| = |S_2|$. So, φ cannot exist. ◀

It seems awkward that in the definition of model agnosticity we need to bound the depth. Let us call an FO(SUM) term or formula *fully model agnostic* if is depth-ℓ model agnostic for every ℓ. It turns out that there are no nontrivial fully model agnostic FO(SUM) formulas.

▶ **Theorem 6.2.** *Let φ be a fully model agnostic closed* FO(SUM) *formula over $\Upsilon^{\text{net}}(m,1)$. Then either $\mathcal{N} \models \varphi$ for all $\mathcal{N} \in \bigcup_{\ell\geq 1} \mathbf{F}(m,\ell)$ or $\mathcal{N} \not\models \varphi$ for all $\mathcal{N} \in \bigcup_{\ell\geq 1} \mathbf{F}(m,\ell)$.*

We omit the proof. The idea is that FO(SUM) is Hanf-local [23, 24]. No formula φ can distinguish a long enough structures consisting of two chains where the middle nodes are marked by two distinct constants c_1 and c_2, from its sibling structure where the markings are swapped. We can turn the two structures into neural networks by replacing the markings by two gadget networks N_1 and N_2, representing different functions, that φ is supposed to distinguish. However, the construction is done so that the function represented by the structure is the same as that represented by the gadget in the left chain. Still, FO(SUM) cannot distinguish these two structures. So, φ is either not fully model-agnostic, or N_1 and N_2 cannot exist and φ is trivial.

▶ **Corollary 6.3.** *The* FO$(\mathbf{R}, F)$ *query $F(0) = 0$ is not expressible in* FO(SUM).

7 From FO(R_{lin}) to FO(SUM)

In practice, the number of layers in the employed neural network architecture is often fixed and known. Our main result then is that FO(SUM) can express all FO($\mathbf{R}_{\text{lin}}$) queries.

▶ **Theorem 7.1.** *Let m and ℓ be natural numbers. For every closed* FO$(\mathbf{R}_{\text{lin}}, F)$ *formula ψ there exists a closed* FO(SUM) *formula φ such that for every network $\mathcal{N}$ in $\mathbf{F}(m,\ell)$, we have $\mathbf{R}, F^{\mathcal{N}} \models \psi$ iff $\mathcal{N} \models \varphi$.*

The challenge in proving this result is to simulate, using quantification and summation over neurons, the unrestricted access to real numbers that is available in FO($\mathbf{R}_{\text{lin}}$). Thereto, we will divide the relevant real space in a finite number of cells which we can represent by finite tuples of neurons.

The proof involves several steps that transform weighted structures. Before presenting the proof, we formalize such transformations in the notion of FO(SUM) translation, which generalize the classical notion of first-order interpretation [16] to weighted structures.

7.1 FO(SUM) translations

Let Υ and Γ be vocabularies for weighted structures, and let n be a natural number. An *n-ary* FO(SUM) *translation φ from Υ to Γ* consists of a number of formulas and weight terms over Υ, described next. There are formulas $\varphi_{\text{dom}}(\boldsymbol{x})$ and $\varphi_{=}(\boldsymbol{x}_1, \boldsymbol{x}_2)$; formulas $\varphi_R(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ for every k-ary relation symbol R of Γ; and formulas $\varphi_f(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ for every k-ary standard function symbol f of Γ. Furthermore, there are weight terms $\varphi_w(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ for every k-ary weight function w of Γ.

In the above description, bold $\boldsymbol{x}$ denote n-tuples of distinct variables. Thus, the formulas and weight terms of φ define relations or weight functions of arities that are a multiple of n.

We say that φ maps a weighted structure $\mathcal{A}$ over Υ to a weighted structure $\mathcal{B}$ over Γ if there exists a surjective function h from $\varphi_{\text{dom}}(\mathcal{A}) \subseteq A^n$ to B such that:

- $h(\boldsymbol{a}_1) = h(\boldsymbol{a}_2) \Leftrightarrow \mathcal{A} \models \varphi_{=}(\boldsymbol{a}_1, \boldsymbol{a}_2)$;
- $(h(\boldsymbol{a}_1), \ldots, h(\boldsymbol{a}_k)) \in R^{\mathcal{B}} \Leftrightarrow \mathcal{A} \models \varphi_R(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$;
- $(h(\boldsymbol{a}_0) = f^{\mathcal{B}}(h(\boldsymbol{a}_1), \ldots, h(\boldsymbol{a}_k)) \Leftrightarrow \mathcal{A} \models \varphi_f(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$;
- $w^{\mathcal{B}}(h(\boldsymbol{a}_1), \ldots, h(\boldsymbol{a}_m)) = \varphi_w^{\mathcal{A}}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$.

In the above, the bold $\boldsymbol{a}$ denote n-tuples in $\varphi_{\mathrm{dom}}(\mathcal{A})$.

For any given $\mathcal{A}$, if φ maps $\mathcal{A}$ to $\mathcal{B}$, then $\mathcal{B}$ is unique up to isomorphism. Indeed, the elements of B can be understood as representing the equivalence classes of the equivalence relation $\varphi_{=}(\mathcal{A})$ on $\varphi_{\mathrm{dom}}(\mathcal{A})$. In particular, for $\mathcal{B}$ to exist, φ must be *admissible* on $\mathcal{A}$, which means that $\varphi_{=}(\mathcal{A})$ is indeed an equivalence relation on $\varphi_{\mathrm{dom}}(\mathcal{A})$, and all relations and all functions $\varphi_R(\mathcal{A})$, $\varphi_f(\mathcal{A})$ and $\varphi_w(\mathcal{A})$ are invariant under this equivalence relation.

If $\mathbf{K}$ is a class of structures over Υ, and T is a transformation of structures in $\mathbf{K}$ to structures over Γ, we say that φ *expresses* T if φ is admissible on every $\mathcal{A}$ in $\mathbf{K}$, and maps $\mathcal{A}$ to $T(\mathcal{A})$.

The relevant reduction theorem for translations is the following:

▶ **Theorem 7.2.** *Let φ be an n-ary* FO(SUM) *translation from Υ to Γ, and let $\psi(y_1, \ldots, y_k)$ be a formula over Γ. Then there exists a formula $\varphi_\psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ over Υ such that whenever φ maps $\mathcal{A}$ to $\mathcal{B}$ through h, we have $\mathcal{B} \models \psi(h(\boldsymbol{a}_1), \ldots, h(\boldsymbol{a}_k))$ iff $\mathcal{A} \models \varphi_\psi(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$. Furthermore, for any weight term t over Γ, there exists a weight term φ_t over Υ such that $t^{\mathcal{B}}(h(\boldsymbol{a}_1), \ldots, h(\boldsymbol{a}_k)) = \varphi_t^{\mathcal{A}}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k)$.*

Proof (sketch). As this result is well known and straightforward to prove for classical first-order interpretations, we only deal here with summation terms, which are the main new aspect. Let t be of the form $\sum_{y:\gamma} t'$. Then for φ_t we take $\sum_{\boldsymbol{x}:\varphi_\gamma} \varphi_{t'}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{x})/(\sum_{\boldsymbol{x}':\varphi_=(\boldsymbol{x},\boldsymbol{x}')} 1)$. ◀

7.2 Proof of Theorem 7.1

We sketch the proof of Theorem 7.1. For clarity of exposition, we present it first for single inputs, i.e., the case $m = 1$. We present three Lemmas which can be chained together to obtain the theorem.

Piecewise linear functions. We can naturally model piecewise linear (PWL) functions from $\mathbb{R}$ to $\mathbb{R}$ as weighted structures, where the elements are simply the pieces. Each piece p is defined by a line $y = ax + b$ and left and right endpoints. Accordingly, we use a vocabulary $\Upsilon_1^{\mathrm{pwl}}$ with four unary weight functions indicating a, b, and the x-coordinates of the endpoints. (The left- and rightmost pieces have no endpoint; we set their x-coordinate to $\perp$.)

For $m = 1$ and $\ell = 2$, the proof of the following Lemma is based on the same ideas as in the proof sketch we gave for Lemma 5.3. For $m > 1$, PWL functions from $\mathbb{R}^m$ to $\mathbb{R}$ are more complex; the vocabulary $\Upsilon_m^{\mathrm{pwl}}$ and a general proof of the lemma will be described in Section 7.3.

▶ **Lemma 7.3.** *Let m and ℓ be natural numbers. There is an* FO(SUM) *translation from $\Upsilon^{\mathrm{net}}(m, 1)$ to $\Upsilon_m^{\mathrm{pwl}}$ that transforms every network $\mathcal{N}$ in $\mathbf{F}(m, \ell)$ into a proper weighted structure representing $F^{\mathcal{N}}$.*

Hyperplane arrangements. An *affine function* on $\mathbb{R}^d$ is a function of the form $a_0 + a_1 x_1 + \cdots + a_d x_d$. An affine *hyperplane* is the set of zeros of some non-constant affine function (i.e. where at least one of the a_i with $i > 0$ is non-zero). A *hyperplane arrangement* is a collection of affine hyperplanes.

We naturally model a hyperplane arrangement as a weighted structure, where the elements are the hyperplanes. The vocabulary $\Upsilon_d^{\mathrm{arr}}$ simply consists of unary weight functions a_0, a_1, …, a_d indicating the coefficients of the affine function defining each hyperplane.

▶ Remark 7.4. An Υ_d^{arr}-structure may have duplicates, i.e., different elements representing the same hyperplane. This happens when they have the same coefficients up to a constant factor. In our development, we will allow structures with duplicates as representations of hyperplane arrangements.

Cylindrical decomposition. We will make use of a linear version of the notion of cylindrical decomposition (CD) [5], which we call *affine CD*. An affine CD of $\mathbb{R}^d$ is a sequence $\mathcal{D} = \mathcal{D}_0, \ldots, \mathcal{D}_d$, where each $\mathcal{D}_i$ is a partition of $\mathbb{R}^i$. The blocks of partition $\mathcal{D}_i$ are referred to as i-cells or simply cells. The precise definition is by induction on d. For the base case, there is only one possibility $\mathcal{D}_0 = \{\mathbb{R}^0\}$. Now let $d > 0$. Then $\mathcal{D}_0, \ldots, \mathcal{D}_{d-1}$ should already be an affine CD of $\mathbb{R}^{d-1}$. Furthermore, for every cell C of $\mathcal{D}_{d-1}$, there must exist finitely many affine functions $\xi_1, \ldots, \xi_r$ from $\mathbb{R}^{d-1}$ to $\mathbb{R}$, where r may depend on C. These are called the *section mappings* above C, and must satisfy $\xi_1 < \cdots < \xi_r$ on C. In this way, the section mappings induce a partition of the cylinder $C \times \mathbb{R}$ in *sections* and *sectors*. Each section is the graph of a section mapping, restricted to C. Each sector is the volume above C between two consecutive sections. Now $\mathcal{D}_d$ must equal $\{C \times S \mid C \in \mathcal{D}_{d-1}$ and S is a section or sector above $C\}$.

The ordered sequence of cells $C \times S$ formed by the sectors and sections of C is called the *stack above* C, and C is called the *base cell* for these cells.

An affine CD of $\mathbb{R}^d$ is *compatible* with a hyperplane arrangement $\mathcal{A}$ in $\mathbb{R}^d$ if every every d-cell C lies entirely on, or above, or below every hyperplane $h = 0$. (Formally, the affine function h is everywhere zero, or everywhere positive, or everywhere negative, on C.)

We can represent a CD compatible with a hyperplane arrangement as a weighted structure with elements of two sorts: cells and hyperplanes. There is a constant o for the "origin cell" $\mathbb{R}^0$. Binary relations link every $i+1$-cell to its base i-cell, and to its delineating section mappings. (Sections are viewed as degenerate sectors where the two delineating section mappings are identical.) Ternary relations give the order of two hyperplanes in $\mathbb{R}_{i+1}$ above an i-cell, and whether they are equal. The vocabulary for CDs of $\mathbb{R}^d$ is denoted by Υ_d^{cell}.

▶ **Lemma 7.5.** *Let d be a natural number. There is an* FO(SUM) *translation from Υ_d^{arr} to Υ_d^{cell} that maps any hyperplane arrangement $\mathcal{A}$ to a CD that is compatible with $\mathcal{A}$.*

Proof (sketch). We follow the method of vertical decomposition [15]. There is a projection phase, followed by a buildup phase. For the projection phase, let $\mathcal{A}_d := \mathcal{A}$. For $i = d, \ldots, 1$, take all intersections between hyperplanes in $\mathcal{A}_i$, and project one dimension down, i.e., project on the first $i-1$ components. The result is a hyperplane arrangement $\mathcal{A}_{i-1}$ in $\mathbb{R}^{i-1}$. For the buildup phase, let $\mathcal{D}_0 := \{\mathbb{R}^0\}$. For $i = 0, \ldots, d-1$, build a stack above every cell C in $\mathcal{D}_i$ formed by intersecting $C \times \mathbb{R}$ with all hyperplanes in $\mathcal{A}_{i+1}$. The result is a partition $\mathcal{D}_{i+1}$ such that $\mathcal{D}_0, \ldots, \mathcal{D}_{i+1}$ is a CD of $\mathbb{R}^{i+1}$ compatible with $\mathcal{A}_{i+1}$. This algorithm is implementable in FO(SUM). ◀

Ordered formulas and cell selection. Let ψ be the FO($\mathbf{R}_{\text{lin}}, F$) formula under consideration. Let $x_1, \ldots, x_d$ be an enumeration of the set of variables in ψ, free or bound. We may assume that ψ is in prenex normal form $Q_1 x_1 \ldots Q_d x_d\, \chi$, where each Q_i is $\exists$ or $\forall$, and χ is quantifier-free.

We will furthermore assume that ψ is *ordered*, meaning that every atomic subformula is of the form $F(x_{i_1}, \ldots, x_{i_m}) = x_j$ with $i_1 < \cdots < i_m < j$, or is a *linear constraint* of the form $a_0 + a_1 x_1 + \cdots + a_d x_d > 0$. By using extra variables, every FO($\mathbf{R}_{\text{lin}}, F$) formula can be brought in ordered normal form.

Consider a PWL function $f : \mathbb{R} \to \mathbb{R}$. Every piece is a segment of a line $ax + b = y$ in $\mathbb{R}^2$. We define the *hyperplane arrangement corresponding to f in d dimensions* to consist of all hyperplanes $ax_i + b = x_j$, for all lines $ax + b = y$ of f, where $i < j$ (in line with the ordered assumption on the formula ψ). We denote this arrangement by $\mathcal{A}_f$.

Also the query ψ gives rise to a hyperplane arrangement, denoted by $\mathcal{A}_\psi$, which simply consisting of all hyperplanes corresponding to the linear constraints in ψ.

For the following statement, we use the disjoint union $\uplus$ of two weighted structures. Such a disjoint union can itself be represented as a weighted structure over the disjoint union of the two vocabularies, with two extra unary relations to distinguish the two domains.

▶ **Lemma 7.6.** *Let $\psi \equiv Q_1 x_1 \dots Q_d x_d\, \chi$ be an ordered closed $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}}, F)$ formula with function symbol F of arity m. Let $k \in \{0, \dots, d\}$, and let ψ_k be $Q_{k+1} x_{k+1} \dots Q_d x_d\, \chi$. There exists a unary $\mathrm{FO}(\mathrm{SUM})$ query over $\Upsilon_m^{\mathrm{pwl}} \uplus \Upsilon_d^{\mathrm{cell}}$ that returns, on any piecewise linear function $f : \mathbb{R}^m \to \mathbb{R}$ and any CD $\mathcal{D}$ of $\mathbb{R}^d$ compatible with $\mathcal{A}_f \cup \mathcal{A}_\psi$, a set of cells in $\mathbb{R}^k$ whose union equals $\{(v_1, \dots, v_k) \mid \mathbf{R}, f \models \psi(v_1, \dots, v_k)\}$.*

Proof (sketch). As already mentioned we focus first on $m = 1$. The proof is by downward induction on k. The base case $k = d$ deals with the quantifier-free part of ψ. We focus on the atomic subformulas. Subformulas of the form $F(x_i) = x_j$ are dealt with as follows. For every piece p of f, with line $y = ax + b$, select all i-cells where x_i lies between p's endpoints. For each such cell, repeatedly take all cells in the stacks above it until we reach $j - 1$-cells. Now for each of these cells, take the section in its stack given by the section mapping $x_j = ax_i + b$. For each of these sections, again repeatedly take all cells in the stacks above it until we reach d-cells. Denote the obtained set of d-cells by S_p; the desired set of cells is $\bigcup_p S_p$.

Subformulas that are linear constraints, where i is the largest index such that a_1 is nonzero, can be dealt with by taking, above every $i - 1$-cell all sections that lie above the hyperplane corresponding to the constraint, if $a_i > 0$, or, if $a_i < 0$, all sections that lie below it. The described algorithm for the quantifier-free part can be implemented in FO(SUM).

For the inductive case, if Q_{k+1} is $\exists$, we must show that we can project a set of cells down one dimension, which is easy given the cylindrical nature of the decomposition; we just move to the underlying base cells. If Q_{k+1} is $\forall$, we treat it as $\neg\exists\neg$, so we complement the current set of cells, project down, and complement again. ◀

To conclude, let us summarise the structure of the whole proof. We are given a neural network $\mathcal{N}$ in $\mathbf{F}(m, \ell)$, and we want to evaluate a closed $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}}, F)$ formula ψ. We assume the query to be in prenex normal form and ordered. We start with an interpretation that transforms $\mathcal{N}$ to a structure representing the piecewise linear function $F^{\mathcal{N}}$ (Lemma 7.3). Then, using another interpretation, we expand the structure by the hyperplane arrangement obtained from the linear pieces of $F^{\mathcal{N}}$ as well as the query. Using Lemma 7.5, we expand the current structure by a cell decomposition compatible with the hyperplane arrangement. Finally, using Lemma 7.6 we inductively process the query on this cell decomposition, at each step selecting the cells representing all tuples satisfying the current formula. Since the formula ψ is closed, we eventually either get the single 0-dimensional cell, in which case ψ holds, or the empty set, in which case ψ does not hold.

7.3 Extension to multiple inputs

For $m > 1$, the notion of PWL function $f : \mathbb{R}^m \to \mathbb{R}$ is more complex. We can conceptualize our representation of f as a decomposition of $\mathbb{R}^m$ into polytopes where, additionally, every polytope p is accompanied by an affine function f_p such that $f = \bigcup_p f_p|_p$. We call f_p the

component function of f on p. Where for $m = 1$ each pies of piece f was delineated by just two breakpoints, now our polytope in $\mathbb{R}^m$ may be delineated by many hyperplanes, called breakplanes. Thus, the vocabulary $\Upsilon_m^{\mathrm{pwl}}$ includes the position of a polytope relative to the breakplanes, indicating whether the polytope is on the breakplane, or on the positive or negative side of it. We next sketch how to prove Lemma 7.3 in its generality. The proof of Lemma 7.6 poses no additional problems.

We will define a PWL function f_u for every neuron u in the network; the final result is then f_{out}. To represent these functions for every neuron, we simply add one extra relation symbol, indicating to which function each element of a $\Upsilon_m^{\mathrm{pwl}}$-structure belongs. The construction is by induction on the layer number. At the base of the induction are the input neurons. The i-th input neuron defines the PWL functions where there is only one polytope ($\mathbb{R}^m$ itself), whose section mapping is the function $\boldsymbol{x} \mapsto x_i$.

Scaling. For any hidden neuron u and incoming edge $v \to u$ with weight w, we define an auxiliary function $f_{v,u}$ which simply scales f_v by w.

To represent the function defined by u, we need to sum the $f_{v,u}$'s and add u's bias; and apply ReLU. We describe these two steps below, which can be implemented in FO(SUM). For $u = \mathrm{out}$, the ReLU step is omitted.

Summing. For each $v \to u$, let $\mathcal{D}_{v,u}$ be the CD for $f_{v,u}$, and let $\mathcal{A}_{v,u}$ be the set of hyperplanes in $\mathbb{R}^m$ that led to $\mathcal{D}_{v,u}$. We define the arrangements $\mathcal{A}_u = \bigcup_v \mathcal{A}_{v,u}$ and $\mathcal{A} = \bigcup_u \mathcal{A}_u$. We apply Lemma 7.5 to $\mathcal{A}$ to obtain a CD $\mathcal{D}$ of $\mathcal{A}$, which is also compatible with each $\mathcal{A}_u$. Every m-cell C in $\mathcal{D}$ is contained in a unique polytope $p_{v,u}^C \in f_{v,u}$ for every $v \to u$. We can define $p_{v,u}^C$ as the polytope that is positioned the same with respect to the hyperplanes in $\mathcal{A}_{v,u}$ as C is. Two m-cells C and C' are called u-equivalent if $p_{v,u}^C = p_{v,u}^{C'}$ for every $v \to u$. We can partition $\mathbb{R}^m$ in polytopes formed by merging each u-equivalence class $[C]$. Over this partition we define a PWL function g_u. On each equivalence class $[C]$, we define g_u as $\sum_{v \to u} f_{p_{v,u}^C}$, plus u's bias. The constructed function g_u equals $b(u) + \sum_v f_{u,v}$.

ReLU. To represent $\mathrm{ReLU}(g_u)$, we construct the new arrangements $\mathcal{B}_u$ formed by the union of $\mathcal{A}_u$ with all hyperplanes given by component functions of g_u, and $\mathcal{B} = \bigcup_u \mathcal{B}_u$. Again apply Lemma 7.5 to $\mathcal{B}$ to obtain a CD $\mathcal{E}$ of $\mathcal{B}$, which is compatible with each B_u. Again every m-cell C in $\mathcal{E}$ is contained in a unique polytope p_u^C of g_u with respect to $\mathcal{A}_u$. Now two m-cells C and C' are called strongly u-equivalent if they are positioned the same with respect to the hyperplanes in $\mathcal{B}_u$. This implies $p_u^C = p_u^{C'}$ but is stronger. We can partition $\mathbb{R}^m$ in polytopes formed by merging each u-equivalence class $[C]$. Over this partition we define a PWL function $f_u{}'$. Let ξ_u^C be the component function of g_u on p_u^C. On each equivalence class $[C]$, we define $f_u{}'$ as ξ_u^C if it is positive on C; otherwise it is set to be zero. The constructed function $f_u{}'$ equals f_u as desired.

8 Conclusion

The immediate motivation for this research is explainability and the verification of machine learning models. In this sense, our paper can be read as an application to machine learning of classical query languages known from database theory. The novelty compared to earlier proposals [3, 26] is our focus on real-valued weights and input and output features. More speculatively, we may envision machine learning models as genuine data sources, maybe in combination with more standard databases, and we want to provide a uniform interface.

For example, practical applications of large language models will conceivably also need to store a lot of hard facts. However, just being able to query them through natural-language prompts may be suboptimal for integrating them into larger systems. Thus query languages for machine learning models may become a highly relevant research direction.

FO(SUM) queries will be likely very complex, so our result opens up challenges for query processing of complex, analytical SQL queries. Such queries are at the focus of much current database systems research, and supported by recent systems such as DuckDB [30] and Umbra [20]. It remains to be investigated to what extent white-box querying can be made useful in practice. The construction of a cell decomposition of variable space turned out crucial in the proof of our main result. Such cell decompositions might be preproduced by a query processor as a novel kind of index data structure.

While the language FO($\mathbf{R}$) should mainly be seen as an expressiveness benchmark, techniques from SMT solving and linear programming are being adapted in the context of verifying neural networks [2]. Given the challenge, it is conceivable that for specific classes of applications, FO($\mathbf{R}$) querying can be made practical.

Many follow-up questions remain open. Does the simulation of FO($\mathbf{R}_{\text{lin}}$) by FO(SUM) extend to FO($\mathbf{R}$)? Importantly, how about other activations than ReLU [34]? If we extend FO(SUM) with quantifiers over weights, i.e., real numbers, what is the expressiveness gain? Expressing FO($\mathbf{R}$) on bounded-depth neural networks now becomes immediate, but do we get strictly more expressivity? Also, to overcome the problem of being unable to even evaluate neural networks of unbounded depth, it seems natural to add recursion to FO(SUM). Fixed-point languages with real arithmetic can be difficult to handle [6, 10].

The language FO(SUM) can work with weighted relational structures of arbitrary shapes, so it is certainly not restricted to the FNN architecture. Thus, looking at other NN architectures is another direction for further research. Finally, we mention the question of designing flexible model query languages where the number of inputs, or outputs, need not be known in advance [3, 4].

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

2 A. Albarghouthi. Introduction to neural network verification. *Foundations and Trends in Programming Languages*, 7(1–2):1–157, 2021. `doi:10.1561/2500000051`.

3 Marcelo Arenas, Daniel Báez, Pablo Barceló, Jorge Pérez, and Bernardo Subercaseaux. Foundations of symbolic languages for model interpretability. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 11690–11701, 2021. URL: `https://proceedings.neurips.cc/paper/2021/hash/60cb558c40e4f18479664069d9642d5a-Abstract.html`.

4 Marcelo Arenas, Pablo Barceló, Diego Bustamante, Jose Caraball, and Bernardo Subercaseaux. A uniform language to explain decision trees. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, pages 60–70, 2024. `doi:10.24963/kr.2024/6`.

5 S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer, second edition, 2008.

6 Michael Benedikt, Martin Grohe, Leonid Libkin, and Luc Segoufin. Reachability and connectivity queries in constraint databases. *J. Comput. Syst. Sci.*, 66(1):169–206, 2003. doi:10.1016/S0022-0000(02)00034-X.

7 F. Bodria, F. Giannotti, R. Guidotti, F. Naretto, D. Pedreschi, and S. Rinzivillo. Benchmarking and survey of explanation methods for black box models. *Data Mining and Knowledge Discovery*, 37:1719–1778, 2023. doi:10.1007/s10618-023-00933-9.

8 Molnar Christoph. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Leanpub, second edition, 2022. URL: https://christophm.github.io/interpretable-ml-book.

9 George Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.*, 2(4):303–314, 1989. doi:10.1007/BF02551274.

10 Floris Geerts and Bart Kuijpers. On the decidability of termination of query evaluation in transitive-closure logics for polynomial constraint databases. *Theor. Comput. Sci.*, 336(1):125–151, 2005. doi:10.1016/j.tcs.2004.10.034.

11 Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. URL: http://www.deeplearningbook.org/.

12 E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998. doi:10.1006/inco.1997.2675.

13 Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. doi:10.1007/3-540-68804-8.

14 Martin Grohe, Christoph Standke, Juno Steegmans, and Jan Van den Bussche. Query languages for neural networks. *CoRR*, abs/2408.10362, 2024. doi:10.48550/arXiv.2408.10362.

15 Dan Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, chapter 24, pages 529–562. Chapman and Hall/CRC, second edition, 2004. doi:10.1201/9781420035315.ch24.

16 Wilfrid Hodges. *Model Theory*, volume 42 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1993.

17 Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. doi:10.1016/0893-6080(91)90009-T.

18 P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995. doi:10.1006/jcss.1995.1051.

19 A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, 1982. doi:10.1145/322326.322332.

20 A. Kohn, V. Leis, and Th. Neumann. Tidy tuples and flying start: fast compilation and fast execution of relational queries in Umbra. *VLDB Journal*, 30(5):883–905, 2021. doi:10.1007/s00778-020-00643-4.

21 Gabriel M. Kuper, Leonid Libkin, and Jan Paredaens, editors. *Constraint Databases*. Springer, 2000. doi:10.1007/978-3-662-04031-7.

22 Marta Kwiatkowska and Xiyue Zhang. When to trust AI: advances and challenges for certification of neural networks. In Maria Ganzha, Leszek A. Maciaszek, Marcin Paprzycki, and Dominik Slezak, editors, *Proceedings of the 18th Conference on Computer Science and Intelligence Systems, FedCSIS 2023, Warsaw, Poland, September 17-20, 2023*, volume 35 of *Annals of Computer Science and Information Systems*, pages 25–37. Polish Information Processing Society, 2023. doi:10.15439/2023F2324.

23 L. Libkin. Expressive power of SQL. *Theoretical Computer Science*, 296:379–404, 2003. doi:10.1016/S0304-3975(02)00736-3.

24 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

25 Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021. doi:10.1561/2400000035.

26 X. Liu and E. Lorini. A unified logical framework for explanations in classifier systems. *Journal of Logic and Computation*, 33(2):485–515, 2023. doi:10.1093/logcom/exac102.

27 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, pages 4765–4774, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

28 Abo Khamis M., H.Q. Ngo, and A. Rudra. Juggling functions inside a database. *SIGMOD Record*, 46(1):6–13, 2017. doi:10.1145/3093754.3093757.

29 João Marques-Silva. Logic-based explainability in machine learning. In Leopoldo E. Bertossi and Guohui Xiao, editors, *Reasoning Web. Causality, Explanations and Declarative Knowledge - 18th International Summer School 2022, Berlin, Germany, September 27-30, 2022, Tutorial Lectures*, volume 13759 of *Lecture Notes in Computer Science*, pages 24–104. Springer, 2022. doi:10.1007/978-3-031-31414-8_2.

30 Mark Raasveldt and Hannes Mühleisen. Duckdb: an embeddable analytical database. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1981–1984. ACM, 2019. doi:10.1145/3299869.3320212.

31 C. Rudin. Stop explaining black box maching learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1:206–215, 2019. doi:10.1038/s42256-019-0048-x.

32 Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. doi:10.48550/arXiv.1312.6199.

33 A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, 1951.

34 Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HyGIdiRqtm.

35 Szymon Torunczyk. Aggregate queries on sparse databases. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 427–443. ACM, 2020. doi:10.1145/3375395.3387660.

36 Steffen van Bergerem and Nicole Schweikardt. Learning concepts described by weight aggregation logic. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CSL.2021.10.

37 Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanation without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law & Technology*, 31(2):841–887, 2018. doi:10.2139/ssrn.3063289.