Towards a Unifying Logical Framework for Neural Networks

Xiyue Zhang[1], Xiaohong Chen[2(✉)], and Meng Sun[1(✉)]

[1] Peking University, Beijing, China
{zhangxiyue,sunm}@pku.edu.cn
[2] University of Illinois Urbana-Champaign, Champaign, USA
xc3@illinois.edu

Abstract. Neural networks are increasingly used in safety-critical applications such as medical diagnosis and autonomous driving, which calls for the need for formal specification of their behaviors. In this paper, we use matching logic—a unifying logic to specify and reason about programs and computing systems—to axiomatically define dynamic propagation and temporal operations in neural networks and to formally specify common properties about neural networks. As instances, we use matching logic to formalize a variety of neural networks, including generic feed-forward neural networks with different activation functions and recurrent neural networks. We define their formal semantics and several common properties in matching logic. This way, we obtain a unifying logical framework for specifying neural networks and their properties.

Keywords: Matching logic · Neural networks · Formal specifications

1 Introduction

Neural networks have been used in an increasing number of cutting-edge applications, especially safety-critical ones, such as autonomous vehicle control [2,10], healthcare [1], and cyber security [11,35]. Due to the impressive performance of neural networks in prediction accuracy and efficiency, a wide range of systems have incorporated them as decision-making components.

Feed-forward neural networks (FNNs) [33], which generally include *convolutional neural networks* (CNNs) [24], are the quintessential neural network models. As the name suggests, information in FNNs flows across the network, from the input layer, through the intermediate layers (also called hidden layers), to the output layer. Another popular type *recurrent neural networks* (RNNs) [16] are designed for processing sequential data [42], which have achieved tremendous success and powered many important commercial applications [37].

Along with the widespread application of neural networks to mission-critical domains, concerns with regard to their safety and reliability arise. This highlights the importance of providing a formal and rigorous specification framework for neural networks, wherein the neural network specification could further lend itself to proof certification and model checking for safety guarantees about

H. Seidl et al. (Eds.): ICTAC 2022, LNCS 13572, pp. 442–461, 2022.
https://doi.org/10.1007/978-3-031-17715-6_28

the neural network behaviors. An ideal specification framework should support different activation functions, network architectures, and various properties dedicated to neural networks. In addition, it should admit the flexibility of incorporating new activation function designs, operations, and properties that arise in the rapidly developing field of machine learning.

Although the last few years witnessed a growing interest in formal verification of neural networks [3,4,13–15,17,22,32,38–40,44,45,47,49,51], there are few attempts in the development of specification frameworks to characterize neural network behaviors. [25] proposes a logical formalism, ReLU temporal logic (ReTL), for feed-forward networks using ReLU activation function [27], which extends linear temporal logic [30,31] with terms capturing data processing in ReLU networks. Albeit the popularity and piece-wise linear characteristic of ReLU networks, ReTL confined to a specific type of network falls short in providing a *unifying* logical framework for the specification of generic neural networks with different activation functions and architectures. [34] explores the specification of some properties dedicated to neural networks without considering the network itself, and organizes the properties along two dimensions: semantic classification and trace-theoretic classification. [36] introduces a framework DNNV focusing on standardizing the network and property format of verifiers by performing network simplification, property reduction, and translation. Therefore, the problem of finding *a unifying logic* that can serve as a specification framework for all types of neural networks, is yet to be addressed. In comparison with existing verification techniques, a unifying logical framework aims to provide rigorous definitions of formal semantics for different types of neural networks. On this basis, the behavior correctness of neural networks such as the robustness property can be specified as a logical formula and proved by generating machine-checkable proof certificates.

In this paper, we initiate an attempt to present a unifying logical framework for the formalization of neural networks using matching logic [7]. This framework builds on the patterns and pattern matching semantics of matching logic and leverages its key insight of capturing a new specification (also called a theory) based on a simple and minimal core. Specifically, we define a general logical framework to characterize the linear operations, dynamic propagation, and temporal behaviors of neural networks. It turns out the proposed logical framework not only subsumes ReTL (Sect. 4), but also offers good extensibility to neural network variants with different activation functions, as well as realistic neural network architectures such as RNNs (Sect. 5) and CNNs (see [52, Appendix C]). For the logical formalism ReTL, we prove equivalence theorems to show that our definitions are syntactically and semantically faithful. Based on this logical framework, the correctness of neural network behaviors can be further guaranteed by the existence of formal proofs for safety properties of interest, which are encoded as machine-checkable proof objects and generated leveraging existing verification practices.

To sum up, the primary contributions of this work are:

1. We present a unifying logical framework for specifying and reasoning about neural network behaviors based on matching logic (Sect. 3).

2. We define ReTL in our logical framework and prove the correctness of the definition by the equivalence theorems (Theorems 1 and 2). We establish formal semantics in the form of standard models for generic FNNs with a variety of activation functions (Sect. 4).
3. We define formal semantics for RNNs with specialized operations and architecture design to process sequential data (Sect. 5).
4. We show the formal specifications of common neural network properties, including robustness, interval, monotonicity, and fairness, based on our logical framework (Sect. 6).

2 Preliminaries

2.1 Neural Networks Preliminaries

Neural networks consist of layers of computation units, interconnected in a feed-forward manner or integrated with loops/cycles in the network. The former type is known as feed-forward neural networks and the representative of the latter type is recurrent neural networks. Convolutional neural networks are a specialized version of feed-forward neural networks with their operations and architectures designed for computer vision tasks.

A general feed-forward neural network containing $L+1$ layers of interconnected neurons (with the first layer as the input layer) can be described by a set of matrices $\{M_i\}_{i=1}^{L}$ and bias vectors $\{b_i\}_{i=1}^{L}$ for the computation of affine transformations, followed by a pointwise activation function for nonlinear transformations. Activation functions allow effective backpropagation to learn the mappings between the network inputs and outputs. For classification tasks, *softmax* function is usually used as the activation function for the output layer, giving the probability of the input being classified in each label.

One of the most popular neural networks used in practice is the feed-forward neural network with the ReLU activation function (ReLU network in short). We present the mathematical description of layer-by-layer forward computation in ReLU networks in the following:

- The affine transformation is performed first to obtain an intermediate vector for each layer i from the previous layer $i-1$: $v_i = M_i \cdot v_{i-1} + b_i$ for $1 \leq i \leq L$ where v_0 is the vector on the input layer.
- The pointwise application of ReLU is then performed on layer i except the output layer, which presents the final vector representation: $v'_i = relu(v_i)$ for $1 \leq i \leq L-1$.

In the case of the output layer (i.e., $i = L$), $v'_L = softmax(v_L)$. The final predicted label is $y = \arg\max_{1 \leq j \leq n} v'_L[j]$. We often leave out the activation function *softmax* on the output layer in the prediction phase. It would not affect the predicted label as the value comparison results between the vector components stay the same without the application of *softmax*.

2.2 Matching Logic Preliminaries

In this section, we review the syntax and semantics of matching logic following [6], and then introduce the definitions of several important constructs.

Matching Logic Syntax. Matching logic is a logical formalism that specifies and reasons about structure by means of patterns and pattern matching. Matching logic formulas, called *patterns*, are built using variables, symbols, a binary construct called *application*, the standard FOL constructs $\bot$, $\rightarrow$, $\exists$, and the least fixpoint construct μ. Variables in matching logic include a set EV of *element variables* denoted $x, y, \ldots$, and a set SV of *set variables* denoted $Y, Z, \ldots$.

Definition 1. *A matching logic* signature Σ *is a set of* (constant) symbols *denoted* $\sigma, \sigma_1, \sigma_2, \ldots$. *The set of* ($\Sigma$-)patterns *is inductively defined as follows:*

$$\varphi ::= x \mid Y \mid \sigma \mid \varphi_1\ \varphi_2 \mid \bot \mid \varphi_1 \rightarrow \varphi_2 \mid \exists x\,.\,\varphi \mid \mu Y\,.\,\varphi$$

where in $\mu Y\,.\,\varphi$, *we require that* φ *is positive in* Y, *i.e.,* Y *is not nested in an odd number of times on the left-hand side of an implication* $\varphi_1 \rightarrow \varphi_2$.

Pattern $\varphi_1\,\varphi_2$ is called an application. For example, $\mathsf{succ}\ \mathsf{zero}$ is a pattern matched by the successor of 0, i.e., 1, where $\mathsf{succ} \in \Sigma$ represents the *successor* function. Other connectives such as $\neg$ and $\vee$ are defined in the usual way.

Matching Logic Semantics. Matching logic has *pattern matching* semantics and each pattern is interpreted in a model as the set of elements that match it. We first define matching logic models.

Definition 2. *A* matching logic Σ-model *consists of*

1. *a nonempty* carrier set M;
2. *the interpretation of application: an* application function $\_\bullet\_ : M \times M \rightarrow \mathcal{P}(M)$, *where* $\mathcal{P}(M)$ *is the powerset of* M;
3. *the interpretation of symbols:* $M_\sigma \subseteq M$ *as a subset of* M *for* $\sigma \in \Sigma$.

Note that application and symbols are interpreted to return a set of elements in matching logic models. Next, we define variable and pattern valuations.

Definition 3. *Given a model* M, *a* variable valuation *is a function* ρ: $(EV \cup SV) \rightarrow (M \cup \mathcal{P}(M))$ *such that* $\rho(x) \in M$ *for* $x \in EV$ *and* $\rho(Y) \subseteq M$ *for* $Y \in SV$. *We define* pattern valuation $|\varphi|_{M,\rho}$ *inductively as follows:*

1. $|x|_{M,\rho} = \{\rho(x)\}$ *for* $x \in EV$
2. $|Y|_{M,\rho} = \rho(Y)$ *for* $Y \in SV$
3. $|\sigma|_{M,\rho} = M_\sigma$ *for* $\sigma \in \Sigma$
4. $|\varphi_1\,\varphi_2|_{M,\rho} = \bigcup_{a_i \in |\varphi_i|_{M,\rho}, i \in \{1,2\}} a_1 \bullet a_2$
5. $|\bot|_{M,\rho} = \emptyset$
6. $|\varphi_1 \rightarrow \varphi_2|_{M,\rho} = M \setminus (|\varphi_1|_{M,\rho} \setminus |\varphi_2|_{M,\rho})$

7. $|\exists x \,.\, \varphi|_{M,\rho} = \bigcup_{a \in M} |\varphi|_{M,\rho[a/x]}$
8. $|\mu Y \,.\, \varphi|_{M,\rho} = \mathbf{lfp}(A \mapsto |\varphi|_{M,\rho[A/Y]})$

Here, $\mathbf{lfp}(A \mapsto |\varphi|_{M,\rho[A/Y]})$ *denotes the smallest set* A *such that* $A = |\varphi|_{M,\rho[A/Y]}$*, where the existence of the unique least fixpoint is guaranteed by the Knaster-Tarski theorem* [43] *(see also* [7, Section IV.B]*).*

We now define the matching logic validity.

Definition 4. *Given* M *and* φ*,* $M \vDash \varphi$ *iff* $|\varphi|_{M,\rho} = M$ *for all* ρ*. Let* Γ *be a theory; that is a set of patterns which we call* axioms*.* $M \vDash \Gamma$ *iff* $M \vDash \varphi$ *for all* $\varphi \in \Gamma$*.* $\Gamma \vDash \varphi$ *iff* $M \vDash \Gamma$ *implies* $M \vDash \varphi$ *for all* M*.*

Important Constructs. Several mathematical constructs that are of practical importance, such as equality, membership, set inclusion, sorts, many-sorted functions and partial functions, can be defined in matching logic. Here we present the definitions of *many-sorted function* and *partial function*. Detailed definitions and notations of other mathematical constructs can be found in [52, Appendix A.2].

A many-sorted function $f : s_1 \times \cdots \times s_n \to s$ can be defined as a symbol, and axiomatized by the following axiom:

$$\text{(Function)} \quad \forall x_1 : s_1 \cdots \forall x_n : s_n \,.\, \exists y : s \,.\, f(x_1, \cdots, x_n) = y$$

Note that the (Function) axiom guarantees that there is exactly one element y because if there exists y' such that $f(x_1, \cdots, x_n) = y'$, then $y = y'$.

Similarly, a many-sorted partial function $f : s_1 \times \cdots \times s_n \rightharpoonup s$ can be defined as a symbol, and axiomatized by the following axiom:

$$\text{(Partial Function)} \quad \forall x_1 : s_1 \cdots \forall x_n : s_n \,.\, \exists y : s \,.\, f(x_1, \cdots, x_n) \subseteq y$$

3 Unifying Logical Framework for Neural Networks

In this section, we present our logical framework for unifying specification of neural network behaviors based on matching logic. We provide a complete formal definition of its main constructs for linear operations, dynamic propagation, and temporal behaviors in neural networks.

3.1 Defining Linear Operations

In this subsection, we show how the vector space in neural networks can be characterized and defined as patterns of matching logic. The most common computation operations in neural networks are linear operations on the vector space, which are defined as follows.

Definition 5. *Let* Nat, Vector *and* Matrix *be the sorts of natural numbers, vectors and matrices, respectively. We define the following symbols:*

$$\_[\_] : \mathsf{Vector} \times \mathsf{Nat} \rightharpoonup \mathsf{Vector}$$
$$\_+\_ : \mathsf{Vector} \times \mathsf{Vector} \rightharpoonup \mathsf{Vector}$$
$$\_\cdot\_ : \mathsf{Matrix} \times \mathsf{Vector} \rightharpoonup \mathsf{Vector}$$

The *projection* symbol $\_[\_]$ takes a vector v and a natural number k, and returns the k-th component of v (as a vector of length 1) when $k \leq \mathsf{len}(v)$. The *addition* symbol $\_+\_$ takes two vectors v, w, and returns $v + w$ when v and w are of the same length. The *matrix multiplication* symbol $\_\cdot\_$ takes a matrix M and a vector v, and returns $M \cdot x$ when the number of columns of M, denoted $col(M)$, equals the length of v. Note that the above symbols are partial functions as stated by the (Partial Function) axioms. They are undefined in the cases of vector dimension exceedance, vector dimension inequality, matrix dimension mismatch, etc.

The comparison operators for the Vector sort are specified in the following way. For any two vectors $v = (v_i)_n$ and $w = (w_i)_n$ of length n, we denote $v = w$ iff $v_i = w_i$ for every $1 \leq i \leq n$. We denote $v \geq w$ iff $v_i \geq w_i$ for every $1 \leq i \leq n$. Other comparison operators are defined in a similar way.

Definition 6. *Let* Layer *be the sort of* network layers *and* Term *be the sort of* functions *that map a layer to a vector. We define the partial function*

$$\mathsf{eval} : \mathsf{Term} \times \mathsf{Layer} \rightharpoonup \mathsf{Vector}$$

to evaluate a term at a given layer.

The key insight of introducing the Term and Layer sorts, as well as the eval symbol is to fully capture the dynamic propagation in neural networks. Recall that in feed-forward neural networks, the transformation parameters in terms of matrices, bias vectors, and activation functions are configured differently at different layers. By introducing Term, the layer information is captured implicitly in the function term. The transformation can be configured accordingly given the element in the Layer sort. For example, when we compute one-step forward linear transformations in a network, matrix M_j and bias vector b_j can be configured according to the layer position j given by the (argument) sort Layer of symbol eval.

We declare a subsort relation $\mathsf{Vector} \subseteq_{\mathsf{subsort}} \mathsf{Term}$, since each vector v in Vector sort can be regarded as a constant term, which is axiomatized as:

$$(\text{Constant Term}) \quad \forall v : \mathsf{Vector}\ \forall l : \mathsf{Layer}\,.\,\mathsf{eval}(v, l) = v$$

This allows us to characterize patterns of Vector sort as patterns of Term sort in a consistent manner, where the (Constant Term) axiom is automatically assumed for elements in Vector sort. The symbols for Vector sort are propagated and overloaded with respect to Term sort, which also leads us to the corresponding axiom for each symbol.

Definition 7. *Let* Layer *and* Term *be the sorts of layers and terms, and* eval *be the evaluation function in Definition 6. We define the following symbols:*

$$\begin{aligned} \_[\_] &: \mathsf{Term} \times \mathsf{Nat} \rightharpoonup \mathsf{Term} \\ \_+\_ &: \mathsf{Term} \times \mathsf{Term} \rightharpoonup \mathsf{Term} \\ \_\cdot\_ &: \mathsf{Matrix} \times \mathsf{Term} \rightharpoonup \mathsf{Term} \end{aligned}$$

as well as the following propagation axioms:

$$\begin{aligned} (\textsc{Projection}) \quad & \mathsf{eval}(t[k], l) = \mathsf{eval}(t, l)[k] \\ (\textsc{Addition}) \quad & \mathsf{eval}(t_1 + t_2, l) = \mathsf{eval}(t_1, l) + \mathsf{eval}(t_2, l) \\ (\textsc{Matmult}) \quad & \mathsf{eval}(M \cdot t, l) = M \cdot \mathsf{eval}(t, l) \end{aligned}$$

where the universal quantification with respect to sorts Term, Layer, Nat, *and* Matrix *are defined in the expected way.*

The above axioms state that a term t on the layer set can be lifted pointwise (i.e., layer-wise) to incorporate symbols for projection, addition, and matrix multiplication. Intuitively, (Projection) states that pointwise projection $t[k]$ maps any layer l to the k-th vector component to which term t maps. (Addition) axiomatizes the pointwise addition $t_1 + t_2$, which maps any layer l to the sum of the vectors to which the (argument) terms t_1 and t_2 map. (Matmult) axiomatizes the pointwise multiplication $M \cdot t$, which maps any layer l to the multiplication result of the constant matrix M and the vector to which term t maps.

3.2 Defining Dynamic Propagation

Neural networks achieve prediction by performing dynamic (forward) propagation from the input layer to the output layer. On each layer, a neural network computes a new vector through the layer-to-layer transformations. We show the mathematical description of the dynamic transformations for a feed-forward ReLU network in Sect. 2.1, where the vector of the successor layer is attained by first computing the affine (linear) transformations of the given vector and then applying the nonlinear activation function. This computation process continues until a vector representation reaches the output layer.

To capture such dynamic computation flows through the network layers, we define the following function:

$$\mathsf{next} : \mathsf{Term} \to \mathsf{Term}$$

Recall that the elements in Term sort are functions that map layers to vectors which are computed with properly configured transformations (dependent on the layer position). We introduce the next symbol to allow the function term to update; that is, the mapping from layers to the feature vectors, in order to specify the forward propagation across the network. We illustrate the mechanism of the next symbol in the following example. For a neural network with $L + 1$ layers,

we denote the transformation functions between adjacent layers as $f_1, f_2, \cdots, f_L$ where f_i $(1 \leq i \leq L)$ is the transformation function from layer $i-1$ to layer i. The next symbol satisfies that for all terms t, for layers $1 \leq l \leq L$:

$$\mathsf{eval}(\mathsf{next}(t), l) = f_l(\mathsf{eval}(t, l-1))$$

Intuitively, the new term $\mathsf{next}(t)$ represents the updated function term that maps layers to the vector representation after one-layer forward computation with regard to the layer transformations. Specifically, for the feature vector v_l of layer l characterized by the updated term as $\mathsf{eval}(\mathsf{next}(t), l)$, it is the computation result of applying the transformation function f_l to the feature vector v_{l-1} of the previous layer specified as $\mathsf{eval}(t, l-1)$. And the layer position l settles the configuration of the corresponding transformation function.

We have seen how the next function characterizes the forward flow of the vector representation across the layers. Other symbols that characterize the dynamic behaviors of neural networks can be defined in a similar way. For example, we define the function shift to allow the information to flow backward.

$$\mathsf{shift} : \mathsf{Term} \rightarrow \mathsf{Term}$$
$$(\textsc{Shift}) \quad \forall t : \mathsf{Term}\ \forall l : \mathsf{Layer}\,.\,\mathsf{eval}(\mathsf{shift}(t), l) = \mathsf{eval}(t, l+1)$$

Intuitively, the symbol shift takes a term t and returns a new term that maps a layer to the vector representation originally configured on the successor layer after one-step backward shift. Note that shift makes no update to the feature vectors, but only changes the mapping from layer positions to feature vectors.

3.3 Defining Atomic Formulas and Temporal Operations

To reason about neural network behaviors, the framework needs to support constraint specifications on intermediate feature vectors and output vectors with respect to an oracle, which often comes in the form of a geometric region in the vector space. Therefore, we design the following constructs as atomic formulas to capture the vector constraints with regard to different layer positions.

$$t_1 = t_2 \equiv \exists l : \mathsf{Layer}\,.\,l \wedge \mathsf{eval}(t_1, l) = \mathsf{eval}(t_2, l)$$
$$t_1 \geq t_2 \equiv \exists l : \mathsf{Layer}\,.\,l \wedge \mathsf{eval}(t_1, l) \geq \mathsf{eval}(t_2, l)$$

The pattern $t_1 = t_2$ is matched by the layers such that the vectors to which t_1 corresponds are equal to the ones t_2 corresponds to. Note that we use existential quantification over layers to obtain the union set of layers of which the corresponding feature vectors satisfy the required constraints. We overload the equality notation to define patterns with respect to terms, which are interpreted as matching elements in the Layer sort. Similar interpretation applies to the pattern $t_1 \geq t_2$. The intuition behind the above definitions is to specify the constraint with regard to the vectors (on the intermediate and output layers) as membership inquiries on layers, that is, to determine whether a layer is a

member of the set of layers that match the required constraint. Other atomic formulas in the form of comparison operators like $>, <, \leq, \neq$ can be derived by the propositional connectives, e.g., $t_1 < t_2 \equiv \neg(t_1 \geq t_2)$.

Neural networks can be regarded as transition systems whose states are the layers and the vector values on them. To specify layer transitions, we introduce the symbol $\mathsf{X} : \mathsf{Layer} \to \mathsf{Layer}$, which takes a layer and returns its predecessor layer in a neural network. The pattern $\mathsf{X}\varphi$ is matched by layers whose direct successor layers match φ. This construct allows us to specify and reason about behaviors of arbitrary layers. For example, we can reason about the i-th step unrolled recurrent layer we are interested in by transiting to this specific state through the X symbol. Other temporal operations can be derived from the symbol X and the μ-binder. $\varphi_1 \,\mathsf{U}\, \varphi_2$ is defined as $\mu Y \,.\, \varphi_2 \vee (\varphi_1 \wedge \mathsf{X} Y)$ where Y is the set variable of Layer sort.

4 Instance: Feed-Forward Neural Networks

In this section, we instantiate the generic logical framework for neural networks in Sect. 3 with *feed-forward neural networks* (FNNs) and obtain a formal semantics of FNNs in matching logic. Among different types of FNNs, due to the computational simplicity and piece-wise linear property of the *ReLU* function, ReLU networks have rapidly become the most popular networks in practical applications. A recent logic design ReTL [25] is introduced to specify and reason about the behaviors of ReLU networks. We will first introduce the logical formalism ReTL. We then demonstrate how ReTL can be defined and subsumed in the proposed logical framework. We finally show the generalization of our framework to feed-forward network variants with different activation functions.

4.1 ReTL: A Logic for ReLU Networks

We give an overview of ReTL designed for specifying and reasoning about ReLU networks. The *syntax* of ReTL defines terms and formulas. Formally, ReTL terms are inductively defined by the following grammar:

$$\underline{\textit{ReTL terms}} \quad t ::= v \mid x \mid t[i] \mid t_1 + t_2 \mid Mt \mid \bigcirc_k t$$

where $t[i]$ is the projection of t on the i-th component; $t_1 + t_2$ is the addition of t_1 and t_2; Mt is the multiplication between a (concrete) matrix M and a term t; and $\bigcirc_k t$ denotes the transformed (future) value of t in k-th next layer.

ReTL formulas are then built from formulas such as $t_1 = t_2$ and $t_1 \geq t_2$ where t_1 and t_2 are terms of the same length, propositional connectives such as $\neg$ and $\wedge$, FOL-style quantification, and temporal operators $\mathsf{X}\varphi$ ("next" φ) and $\varphi_1 \,\mathsf{U}\, \varphi_2$ (φ_1 "until" φ_2). Formally,

$$\underline{\textit{ReTL formulas}} \quad \varphi ::= t_1 = t_2 \mid t_1 \geq t_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists x \,.\, \varphi \mid \mathsf{X}\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

The *semantics* of ReTL is defined with regard to a ReLU network N (with $L + 1$ layers), a layer position $0 \leq l \leq L$, and a valuation ρ that assigns each

vector variable to a vector of proper length. The semantics of ReTL terms are inductively defined as follows:

- $|v|^{\mathsf{ReTL}}_{N,\rho,l} = v$ for any vector constant v.
- $|x|^{\mathsf{ReTL}}_{N,\rho,l} = \rho(x)$ for any vector variable x.
- $|t[i]|^{\mathsf{ReTL}}_{N,\rho,l}$ is the i-th element of $|t|^{\mathsf{ReTL}}_{N,\rho,l}$.
- $|t_1 + t_2|^{\mathsf{ReTL}}_{N,\rho,l} = |t_1|^{\mathsf{ReTL}}_{N,\rho,l} + |t_2|^{\mathsf{ReTL}}_{N,\rho,l}$.
- $|Mt|^{\mathsf{ReTL}}_{N,\rho,l} = M|t|^{\mathsf{ReTL}}_{N,\rho,l}$.
- The semantics of $|\bigcirc_k t|^{\mathsf{ReTL}}_{N,\rho,l}$ are defined in several cases:
  (1) if $k = 0$, then $|\bigcirc_k t|^{\mathsf{ReTL}}_{N,\rho,l} = |t|^{\mathsf{ReTL}}_{N,\rho,l}$,
  (2) if $k > 0$ and $l + k < L$, then $|\bigcirc_k t|^{\mathsf{ReTL}}_{N,\rho,l} = relu(M_{l+k}\ |\bigcirc_{k-1} t|^{\mathsf{ReTL}}_{N,\rho,l} + b_{l+k})$,
  (3) if $k > 0$ and $l + k = L$, then $|\bigcirc_k t|^{\mathsf{ReTL}}_{N,\rho,l} = M_L|\bigcirc_{k-1} t|^{\mathsf{ReTL}}_{N,\rho,l} + b_L$,
  (4) if $l + k > L$, the term is undefined.

ReTL formulas $t_1 = t_2$ and $t_1 \geq t_2$ are interpreted as $|t_1|^{\mathsf{ReTL}}_{N,\rho,l} = |t_2|^{\mathsf{ReTL}}_{N,\rho,l}$ and $|t_1|^{\mathsf{ReTL}}_{N,\rho,l} \geq |t_2|^{\mathsf{ReTL}}_{N,\rho,l}$, respectively. The satisfaction relation $\models_{\mathsf{ReTL}}$ of ReTL formulas is defined as follows:

- $N, \rho, l \models_{\mathsf{ReTL}} t_1 = t_2$ *iff* $|t_1|^{\mathsf{ReTL}}_{N,\rho,l} = |t_2|^{\mathsf{ReTL}}_{N,\rho,l}$.
- $N, \rho, l \models_{\mathsf{ReTL}} t_1 \geq t_2$ *iff* $|t_1|^{\mathsf{ReTL}}_{N,\rho,l} \geq |t_2|^{\mathsf{ReTL}}_{N,\rho,l}$.
- $N, \rho, l \models_{\mathsf{ReTL}} \neg\varphi$ *iff* $\mathsf{N}, \rho, l \not\models_{\mathsf{ReTL}} \varphi$.
- $N, \rho, l \models_{\mathsf{ReTL}} \varphi_1 \wedge \varphi_2$ *iff* $N, \rho, l \models_{\mathsf{ReTL}} \varphi_1$ and $N, \rho, l \models_{\mathsf{ReTL}} \varphi_2$.
- $N, \rho, l \models_{\mathsf{ReTL}} \exists x \in \mathbb{R}^n . \varphi$ *iff* there exists $v \in \mathbb{R}^n$ such that $N, \rho[v/x], l \models_{\mathsf{ReTL}} \varphi$, where $\rho[v/x]$ denotes the valuation with $\rho[v/x](x) = v$ and $\rho[v/x](y) = \rho(y)$ for all $y \neq x$.
- $N, \rho, l \models_{\mathsf{ReTL}} \mathsf{X}\varphi$ *iff* $l < L$ and $N, \rho, l + 1 \models_{\mathsf{ReTL}} \varphi$.
- $N, \rho, l \models_{\mathsf{ReTL}} \varphi_1\ \mathsf{U}\ \varphi_2$ *iff* there exists some $k \geq l$ and $k \leq L$ such that $N, \rho, k \models_{\mathsf{ReTL}} \varphi_2$ and $N, \rho, j \models_{\mathsf{ReTL}} \varphi_1$ for $l \leq j < k$.

4.2 Defining ReTL in Matching Logic

The syntax and semantics of *ReTL terms* can be subsumed by the Term sort (including Vector as the subsort), the symbols for linear operations and dynamic propagation in our logical framework. However, the definition of $\bigcirc_k t$ in ReTL for k-step forward transformations is not trivial. In the following, we show how to capture the syntax and semantics of the operation $\bigcirc_k t$ for k-step dynamic computation using the next and shift symbols. We use next^k and shift^k as the shortcut for k continuous application of next and shift in the following context.

With the next symbol defined for the forward propagation of feature vectors and the shift symbol for the backward layer shift, the $\bigcirc_k t$ operation can be defined as:

$$\bigcirc_k t \equiv \mathsf{shift}^k(\mathsf{next}^k(t))$$

Thus, for a given layer l, we have

$$\mathsf{eval}(\bigcirc_k t, l) = \mathsf{eval}(\mathsf{next}^k(t), l + k)$$

Recall that the satisfaction relation in ReTL is evaluated with regard to a certain layer l. Through defining $\bigcirc_k t$ with k-step forward propagation and k-step backward shift, the interpretation of $\bigcirc_k t$ is consistent with the semantics in ReTL, where the forward propagation of the vector representation from l to $l+k$ is followed by backward shift from layer $l+k$ to l. This way, the satisfaction relation can be evaluated with regard to layer l.

The forward propagation for arbitrary steps can be captured by repeating the application of next accordingly without introducing a variable indicating how many forward steps to take. As the next symbol involves no layer shift operations, the updated vector representation is taken on layer $l+k$, which also indicates the correct configuration of next step forward transformations (from layer $l+k$).

Note that in ReTL, $\bigcirc_k t$ is *not* equivalent to $\underbrace{\bigcirc_1 \cdots \bigcirc_1}_{k} t$. It can be seen more evidently in matching logic, where $\bigcirc_k t \equiv \mathsf{shift}^k(\mathsf{next}^k(t))$, but

$$\underbrace{\bigcirc_1 \cdots \bigcirc_1}_{k} t = \underbrace{\mathsf{shift}(\mathsf{next}(\cdots \mathsf{shift}(\mathsf{next}}_{k}(t)) \cdots))$$

Since next and shift are not commutable when shift changes the configured layer transformation to the predecessor layer, the equivalence does not hold when the linear mappings of network layers are different from each other, which is common in practice.

To sum up, we showed that the syntax and semantics of ReTL terms and formulas are *precisely captured* by matching logic patterns, using the built-in constructs for propositional connectives, quantification, and temporal operators in matching logic, without needing to introduce additional logical symbols.

4.3 Defining Standard Models of Feed-Forward ReLU Networks

In this subsection, we establish formal semantics of feed-forward ReLU networks, which are defined in the form of standard models.

Definition 8. *Let N be a ReLU network with $L+1$ layers. A* standard model *M^N for N consists of:*

1. *Carrier set, which is the disjoint union of the following:*
   - *The carrier sets of the* Nat, Vector, *and* Matrix *sorts are defined in the usual way.*
   - *The carrier set of sort* Layer *is $M^N_{\mathsf{Layer}} = \{0, 1, \cdots, L\}$, i.e., the set of the network layers.*
   - *The carrier set of sort* Term *is the set of functions M^N_{Term} that map network layers to vectors.*
2. *Symbol interpretations:*
   - *The projection, addition, and matrix multiplication symbols are interpreted in the usual way.*

- *The* eval *and* shift *symbols are interpreted as functions* M^N_{eval} *and* M^N_{shift} *straightforward from their definitions.* M^N_{eval} *is the partial function mapping a term and a layer (arguments) to a vector;* M^N_{shift} *is the function that maps a term to a new term satisfying Axiom* (SHIFT).
- *The* next *symbol is interpreted as* M^N_{next}*, the unique function that satisfies the following property*

$$M^N_{\mathsf{eval}}(M^N_{\mathsf{next}}(t), l) = \begin{cases} M^N_{\mathsf{eval}}(t, 0) & \text{if } = 0 \\ relu(M_l \cdot M^N_{\mathsf{eval}}(t, l-1) + b_l) & \text{if } \leq l < L \\ M_L \cdot M^N_{\mathsf{eval}}(t, L-1) + b_L & \text{if } = L \end{cases}$$

- *The* X *symbol is interpreted as the function that satisfies*

$$M^N_{\mathsf{X}}(l) = \begin{cases} \{l-1\} & \text{if } 1 \leq l \leq L \\ \emptyset & \text{if } l = 0 \end{cases}$$

Recall that any ReTL term t and formula φ can be defined as a matching logic pattern. Theorems 1 and 2 show that the definitions of ReTL terms and formulas in our logical framework are syntactically and semantically faithful.

Theorem 1. *Let* t *be an ReTL term. Then* $|\mathsf{eval}(t, l)|_{M^N,\rho} = \{|t|^{\mathsf{ReTL}}_{N,\rho,l}\}$.

Theorem 2. *Let* φ *be an ReTL formula. Then* $l \in |\varphi|_{M^N,\rho}$ *iff* $N, \rho, l \models_{\mathsf{ReTL}} \varphi$.

Note that $|\mathsf{eval}(t, l)|_{M^N,\rho}$ evaluates to a singleton set, while $|t|^{\mathsf{ReTL}}_{N,\rho,l}$ evaluates to a single element. Please refer to [52, Appendix B] for proof details.

4.4 Instances of Neural Networks Using Other Activation Functions

We have established that ReTL can be subsumed in the proposed logical framework. In this subsection, we show that this framework allows us to generalize to neural networks using other nonlinear activation functions, without the need to define a new logical formalism specialized for neural networks with new activation function design as nonlinear mapping [9,23]. In the following, we present an example of FNNs that use *tanh* as the nonlinear activation function [20].

Definition 9. *Let* N *be a feed-forward neural network with tanh as the activation function and* $L+1$ *layers. A* standard model M^N *for* N *consists of:*

1. *The* next *symbol is interpreted as* M^N_{next}*, the unique function satisfying that*

$$M^N_{\mathsf{eval}}(M^N_{\mathsf{next}}(t), l) = \begin{cases} M^N_{\mathsf{eval}}(t, 0) & \text{if } = 0 \\ tanh(M_l \cdot M^N_{\mathsf{eval}}(t, l-1) + b_l) & \text{if } \leq l < L \\ M_L \cdot M^N_{\mathsf{eval}}(t, L-1) + b_L & \text{if } l = L \end{cases}$$

2. *Everything else is the same as Definition 8.*

Formal semantics of all the variants of FNNs [9,23,26,28], including CNNs [52, Appendix C], can be similarly defined in our framework.

5 Instance: Recurrent Neural Networks

In this section, we define *recurrent neural networks* (RNNs) [16] in our logical framework. RNNs are often used to process sequential data or time series data such as natural language [42]. A typical RNN consists of an input layer, a recurrent layer, and a fully-connected layer, in which the recurrent layer enables the RNN to process sequential inputs of any length with shared parameters. The main challenge to characterize the dynamic computation of RNNs is to specify the propagation on the recurrent layer.

Propagation on the Recurrent Layer. The computation flow on the recurrent layer, in the form of hidden state vectors, can be regarded the same as feature vectors in FNNs by unrolling the recurrent layer step by step. The recurrent layer processes the information from the sequential input by incorporating it into the hidden state which is passed forward through time steps.

We can formulate the unrolled recurrence after l steps as $h_l = f(h_{l-1}, x_l)$ where h_l and h_{l-1} represent the hidden state vectors at step l and the previous step $l-1$, x_l indicates the input vector at step l, and f represents the transformation function on the recurrent layer. The number of unrolled hidden layers then depends on the length of the input sequence. However, unlike FNNs, the RNN transformations across the unrolled hidden layers are shared with (1) fixed weight matrices: hidden-to-hidden matrix M_{hh} and input-to-hidden matrix M_{xh}, (2) fixed bias vector b_h, and (3) fixed activation function σ_h.

The challenge to characterize the propagation on the recurrent layer is that for each unrolled hidden layer, different input vectors corresponding to one specific token (or word) of the input sequence are involved. The Term sort is useful to address this challenge, since it is designed to capture the vector representation at different layer positions. For an input sequence $t^{\mathsf{in}} = (t_1^{\mathsf{in}}, t_2^{\mathsf{in}}, \cdots, t_L^{\mathsf{in}})$, we use $\mathsf{eval}(t^{\mathsf{in}}, l)$ to specify the input vector t_l^{in} fed to the recurrent layer at time step l (i.e., unrolled hidden layer l).

The forward propagation of hidden state vectors can then be specified by adding the transformation of a proper input vector at each layer position. Generally, for an input sequence t^{in}, the forward propagation to hidden layer l for $1 \leq l \leq \mathsf{len}(t^{\mathsf{in}})$ is defined as:

$$h_l = \sigma_h(M_{hh} \cdot h_{l-1} + M_{xh} \cdot t_l^{\mathsf{in}} + b_h)$$

Propagation on the Fully-Connected Layer. The dynamic propagation of the hidden state vector on the recurrent layer is generally followed by a fully-connected output layer for many-to-one prediction tasks. The forward propagation from the last hidden layer position $\mathsf{len}(t^{\mathsf{in}})$ (decided by the length of the sequential input) to the output layer demonstrates the same behaviors as FNNs.

Specifically, the forward propagation to the output layer in RNNs is characterized by a hidden-to-output weight matrix M_{hy} and a bias vector b_y. Generally,

the dynamic transformation to the output layer can be defined as:

$$y_l = M_{hy} \cdot h_{l-1} + b_y$$

The same as FNNs, only linear transformations are involved in the computation on the output layer. The projection, addition and matrix multiplication symbols naturally accommodate hidden state vectors of the unrolled recurrent layer.

Standard Models. Here we define the standard models of RNNs.

Definition 10. *Let N be a recurrent neural network with an input layer, a recurrent layer, and a fully-connected layer. A* standard model *M^N for N with respect to a sequential data t^{in} consists of:*

1. *$M^N_{\mathsf{Layer}} = \{0, 1, \cdots, \mathsf{len}(t^{\mathsf{in}})+1\}$ is the carrier set of the* Layer *sort where layer* 0 *indicates the input layer, layers $1, \cdots, \mathsf{len}(t^{\mathsf{in}})$ are the unrolled hidden layers, and layer $\mathsf{len}(t^{\mathsf{in}}) + 1$ indicates the output layer.*
2. *The* next *symbol is interpreted as M^N_{next}, the unique function satisfying that*

$$M^N_{\mathsf{eval}}(M^N_{\mathsf{next}}(t), l) = \begin{cases} h_0 & \text{if } l = 0 \\ \sigma_h(M_{hh} \cdot M^N_{\mathsf{eval}}(t, l-1) + M_{xh} \cdot M^N_{\mathsf{eval}}(t^{\mathsf{in}}, l) + b_h) & \\ & \text{if } 1 \leq l \leq \mathsf{len}(t^{\mathsf{in}}) \\ M_{hy} \cdot M^N_{\mathsf{eval}}(t, \mathsf{len}(t^{\mathsf{in}})) + b_y & \text{if } l = \mathsf{len}(t^{\mathsf{in}}) + 1 \end{cases}$$

3. *Everything else is the same as Definition 8.*

Now we have shown that RNNs integrated with loops/cycles can be defined in the proposed logical framework in a consistent manner with FNNs, with slight change to the carrier set of Layer sort and the interpretation of the next symbol.

6 Specifying Neural Network Properties

In this section, we present some common properties of neural networks and how to formally specify them as patterns, in our unifying framework.

6.1 Robustness

The robustness of neural networks against adversarial perturbations has been extensively investigated [5,12,18,21,29,48]. In general, the robustness property states that, given an input x, a distance function L_p, and a distance bound ϵ, the prediction of the neural network on the ϵ-neighborhood $\eta(x, L_p, \epsilon)$ of x where $\eta(x, L_p, \epsilon) = \{x' \,|\, \|x' - x\|_p \leq \epsilon\}$ is the same as the prediction of input x. This is often referred to as local robustness. Note that in the following we formalize the local robustness property in terms of vector constraints instead of using the $argmax$ function.

Example 1. Consider a neural network N with $L+1$ layers, input dimension s_0, and output dimension s_L. Given an input-label pair (x_0, y_0), the local robustness of N on the input $x_0 \in \mathbb{R}^{s_0}$ with respect to an $\epsilon_0 \in \mathbb{R}^{s_0}$ neighborhood in terms of L_∞ norm can be specified as:

$$\forall x \in \mathbb{R}^{s_0} \,.\, (x \leq x_0 + \epsilon_0) \wedge (x \geq x_0 - \epsilon_0) \rightarrow$$
$$\forall 1 \leq j \leq s_L \,.\, \mathsf{eval}(\mathsf{next}^L(x)[y_0], L) \geq \mathsf{eval}(\mathsf{next}^L(x)[j], L)$$

6.2 Interval Property

The interval property [21,46] aims to analyze whether the outputs of a neural network are restricted in a geometric region. A simple interval property is to determine whether a real number *ub* or *lb* is a valid upper or lower bound for a specific dimension of the output vectors on an input region.

Example 2. Consider a neural network N with $L+1$ layers, input dimension s_0, and output dimension s_L. Given an input region $[0,1]^{s_0}$, the interval property is to check whether lb is a valid lower bound for the k_0-th component of the network outputs on the input region. This interval property can be specified as:

$$\forall x \in \mathbb{R}^{s_0} \,.\, x \leq \mathbf{1} \wedge x \geq \mathbf{0} \rightarrow \mathsf{eval}(\mathsf{next}^L(x)[k_0], L) \geq lb$$

where $\mathbf{1}$ and $\mathbf{0}$ denote the vectors in $\mathbb{R}^{s_0}$ where all the components are 1 and 0.

Neural networks can be regarded as programs that compute the function results of the inputs. From this perspective, the aforementioned robustness and interval properties can be viewed as special cases of functional correctness which specifies the input-output constraints of the programs (neural networks). The specifications of such properties can be formulated in a unified manner with respect to a pair of pre- and post-conditions in the form of $\forall x \,.\, P(x) \rightarrow Q(\mathsf{next}^L(x))$.

6.3 Monotonicity

The monotonicity property [50] focuses on the output monotonicity of a network with respect to a user-specified subset of the inputs. Assuming that a neural network is used to predict whether to give an applicant the loan, the network is expected to guarantee that the prediction will be monotonically increasing when the value of the applicant's income increases and the other values are the same.

Example 3. Consider a neural network N with $L+1$ layers, input dimension s_0, and output dimension s_L. Given that the domain-specific feature dimension is i_0, and the monotonic label dimension is k_0, the monotonicity property is to check whether the k_0-th component of the network outputs is monotonic with respect to the value of input feature i_0. This monotonicity property can be specified as:

$$\forall x_1, x_2 \in \mathbb{R}^{s_0} \,.\, (x_1[i_0] \leq x_2[i_0] \wedge (\forall 1 \leq i \leq s_0 \,.\, i \neq i_0 \rightarrow x_1[i] = x_2[i])) \rightarrow$$
$$\mathsf{eval}(\mathsf{next}^L(x_1)[k_0], L) \leq \mathsf{eval}(\mathsf{next}^L(x_2)[k_0], L)$$

6.4 Fairness

The fairness property [19,41] generally constrains that the neural network's prediction should not be influenced by protected features such as gender, age and race. There are many different formulations of neural network fairness. Here we focus on the independence-based fairness, which states that the neural network's prediction is independent of the values of protected features. Note that we specify the values of protected features explicitly in the following to formalize the independence-based fairness in a more straightforward manner.

Example 4. Consider a neural network N with $L+1$ layers, whose input and output dimensions are s_0 and s_L, respectively. Suppose i_0 is the protected feature dimension, Q is the set of protected feature values, and k_0 is the specified label. *Fairness* states that the k_0-th component of the network outputs is independent of the value of the protected feature i_0, which can be specified as:

$$\forall x_1, x_2 \in \mathbb{R}^{s_0} . ((\forall q_1, q_2 \in Q . x_1[i_0] = q_1 \wedge x_2[i_0] = q_2) \wedge (\forall 1 \leq i \leq s_0 . \\ i \neq i_0 \rightarrow x_1[i] = x_2[i])) \rightarrow \mathsf{eval}(\mathsf{next}^L(x_1)[k_0], L) = \mathsf{eval}(\mathsf{next}^L(x_2)[k_0], L)$$

The exact equivalence on the prediction can be relaxed by introducing a positive tolerance ϵ to the output difference, which is specified as:

$$\forall x_1, x_2 \in \mathbb{R}^{s_0} . ((\forall q_1, q_2 \in Q . x_1[i_0] = q_1 \wedge x_2[i_0] = q_2) \wedge \\ (\forall 1 \leq i \leq s_0 . i \neq i_0 \rightarrow x_1[i] = x_2[i])) \rightarrow \\ (\mathsf{eval}(\mathsf{next}^L(x_1)[k_0], L) \leq \mathsf{eval}(\mathsf{next}^L(x_2)[k_0], L) + \epsilon \wedge \\ \mathsf{eval}(\mathsf{next}^L(x_1)[k_0], L) \geq \mathsf{eval}(\mathsf{next}^L(x_2)[k_0], L) - \epsilon)$$

Neural networks can be viewed as data-driven systems, of which the execution on an input forms a trace, i.e., a sequence of vectors on the layers. We have been familiar with ordinary properties that reason over a single execution of a system. In contrast, the hyperproperty [8] reasons over a set of executions of a system, instead of over a single one. The monotonicity and fairness properties both involve examining pairs of executions, which renders them both as hyperproperties. Compared with ordinary properties, hyperproperties are capable of constraining complex relations among multiple execution traces of a system.

7 Conclusion

We present a unifying logical framework for the specification of neural network behaviors by defining the linear operations, dynamic propagation, and temporal operations as a matching logic theory. The key insight is to provide a general framework to define formal semantics of neural networks with different activation functions and network architectures, so as to offer good extensibility and flexibility in the rapidly developing field of machine learning. We prove that existing logic design ReTL can be faithfully defined in our framework. We also show that the logical framework can serve as the specification formalism of important network properties, such as robustness, interval, monotonicity, and fairness.

Acknowledgements. This research was sponsored by the National Natural Science Foundation of China under Grant No. 62172019, 61772038, and CCF-Huawei Formal Verification Innovation Research Plan. The work presented in this paper was supported in part by NSF CNS 16-19275. This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092.

References

1. Babak, A., Delong, A., Weirauch, M.T., Frey, B.J.: Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat. Biotechnol. **33**(8), 831–838 (2015). https://doi.org/10.1038/nbt.3300
2. Bojarski, M., et al.: End to end learning for self-driving cars. CoRR abs/1604.07316 (2016). http://arxiv.org/abs/1604.07316
3. Boopathy, A., Weng, T., Chen, P., Liu, S., Daniel, L.: CNN-Cert: an efficient framework for certifying robustness of convolutional neural networks. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, Honolulu, Hawaii, USA, pp. 3240–3247. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33013240
4. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Proceedings of the 32nd Annual Conference on Neural Information Processing Systems, Montréal, Canada, pp. 4795–4804. Curran Associates Inc. (2018)
5. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: Proceedings of the 38th IEEE Symposium on Security and Privacy, San Jose, California, USA, pp. 39–57. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.49
6. Chen, X., Lucanu, D., Roşu, G.: Matching logic explained. J. Logical Algebraic Methods Program. **120**, 1–36 (2021). https://doi.org/10.1016/j.jlamp.2021.100638
7. Chen, X., Roşu, G.: Matching μ-logic. In: Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science, Vancouver, Canada, pp. 1–13. IEEE (2019). https://doi.org/10.1109/LICS.2019.8785675
8. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010). https://doi.org/10.3233/JCS-2009-0393
9. Clevert, D., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUs). In: Proceedings of the 4th International Conference on Learning Representations, San Juan, Puerto Rico. OpenReview.net (2016)
10. Codevilla, F., Müller, M., López, A.M., Koltun, V., Dosovitskiy, A.: End-to-end driving via conditional imitation learning. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation, Brisbane, Australia, pp. 1–9. IEEE (2018). https://doi.org/10.1109/ICRA.2018.8460487
11. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, British Columbia, Canada, pp. 3422–3426. IEEE (2013). https://doi.org/10.1109/ICASSP.2013.6638293
12. Dreossi, T., Jha, S., Seshia, S.A.: Semantic adversarial deep learning. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 3–26. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_1

13. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Dutle, A., Muñoz, C., Narkawicz, A. (eds.) NFM 2018. LNCS, vol. 10811, pp. 121–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77935-5_9
14. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence, Monterey, California, USA, pp. 550–559. AUAI Press (2018)
15. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
16. Elman, J.L.: Finding structure in time. Cogn. Sci. **14**(2), 179–211 (1990). https://doi.org/10.1207/s15516709cog1402_1
17. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: Proceedings of the 39th IEEE Symposium on Security and Privacy, San Francisco, California, USA, pp. 3–18. IEEE Computer Society (2018). https://doi.org/10.1109/SP.2018.00058
18. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Proceedings of the 3rd International Conference on Learning Representations, San Diego, California, USA. OpenReview.net (2015)
19. Hardt, M., Price, E., Srebro, N.: Equality of opportunity in supervised learning. In: Proceedings of the 30th Annual Conference on Neural Information Processing Systems, Barcelona, Spain, pp. 3315–3323. Curran Associates Inc. (2016)
20. Hayou, S., Doucet, A., Rousseau, J.: On the selection of initialization and activation function for deep neural networks. CoRR abs/1805.08266 (2018). http://arxiv.org/abs/1805.08266
21. Huang, X., et al.: A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev. **37**, 100270 (2020). https://doi.org/10.1016/j.cosrev.2020.100270
22. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
23. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. In: Proceedings of the 31st Annual Conference on Neural Information Processing Systems, Long Beach, California, USA, pp. 971–980. Curran Associates Inc. (2017)
24. LeCun, Y., Bengio, Y.: Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. MIT Press, Cambridge (1998)
25. Liu, W.-W., Song, F., Zhang, T.-H.-R., Wang, J.: Verifying ReLU neural networks from a model checking perspective. J. Comput. Sci. Technol. **35**(6), 1365–1381 (2020). https://doi.org/10.1007/s11390-020-0546-7
26. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, vol. 30. PMLR (2013)
27. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning, Haifa, Israel, pp. 807–814. Omnipress (2010)
28. Narayan, S.: The generalized sigmoid activation function: competitive supervised learning. Inf. Sci. **99**(1), 69–82 (1997). https://doi.org/10.1016/S0020-0255(96)00200-9

29. Papernot, N., McDaniel, P.D., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: Proceedings of the 1st IEEE European Symposium on Security and Privacy, Saarbrücken, Germany, pp. 372–387. IEEE (2016). https://doi.org/10.1109/EuroSP.2016.36
30. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
31. Roşu, G.: Finite-trace linear temporal logic: coinductive completeness. Formal Methods Syst. Des. **53**(1), 138–163 (2018). https://doi.org/10.1007/s10703-018-0321-3
32. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, Stockholm, Sweden, pp. 2651–2659. ijcai.org (2018). https://doi.org/10.24963/ijcai.2018/368
33. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015). https://doi.org/10.1016/j.neunet.2014.09.003
34. Seshia, S.A., et al.: Formal specification for deep neural networks. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 20–34. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_2
35. Shin, E.C.R., Song, D., Moazzezi, R.: Recognizing functions in binaries with neural networks. In: Proceedings of the 24th USENIX Security Symposium, Washington, D.C., USA, pp. 611–626. USENIX Association (2015)
36. Shriver, D., Elbaum, S., Dwyer, M.B.: DNNV: a framework for deep neural network verification. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 137–150. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_6
37. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: Proceedings of the 7th International Conference on Document Analysis and Recognition, Edinburgh, Scotland, UK, pp. 958–962. IEEE Computer Society (2003). https://doi.org/10.1109/ICDAR.2003.1227801
38. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: Proceedings of the 33rd Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, pp. 15072–15083. Curran Associates Inc. (2019)
39. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 41:1–41:30 (2019). https://doi.org/10.1145/3290354
40. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: Proceedings of the 7th International Conference on Learning Representations, New Orleans, LA, USA. OpenReview.net (2019)
41. Sun, B., Sun, J., Dai, T., Zhang, L.: Probabilistic verification of neural networks against group fairness. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) FM 2021. LNCS, vol. 13047, pp. 83–102. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_5
42. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Proceedings of the 28th Annual Conference on Neural Information Processing Systems, Montreal, Quebec, Canada, pp. 3104–3112. Curran Associates Inc. (2014)
43. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pac. J. Math. **5**(2), 285–309 (1955). pjm/1103044538

44. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: Proceedings of the 7th International Conference on Learning Representations, New Orleans, LA, USA. OpenReview.net (2019)
45. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Proceedings of the 32nd Annual Conference on Neural Information Processing Systems, Montréal, Canada, pp. 6369–6379. Curran Associates Inc. (2018)
46. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proceedings of the 27th USENIX Security Symposium, Baltimore, Maryland, USA, pp. 1599–1614. USENIX Association (2018)
47. Weng, T., et al.: Towards fast computation of certified robustness for ReLU networks. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, Stockholmsmässan, Stockholm, Sweden, vol. 80, pp. 5273–5282. PMLR (2018)
48. Weng, T., et al.: Evaluating the robustness of neural networks: an extreme value theory approach. In: Proceedings of the 6th International Conference on Learning Representations, Vancouver, British Columbia, Canada. OpenReview.net (2018)
49. Xiang, W., Tran, H., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE Trans. Neural Netw. Learn. Syst. **29**(11), 5777–5783 (2018). https://doi.org/10.1109/TNNLS.2018.2808470
50. You, S., Ding, D., Canini, K.R., Pfeifer, J., Gupta, M.R.: Deep lattice networks and partial monotonic functions. In: Proceedings of the 31st Annual Conference on Neural Information Processing Systems, Long Beach, California, USA, pp. 2981–2989. Curran Associates Inc. (2017)
51. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Proceedings of the 32nd Annual Conference on Neural Information Processing Systems, Montréal, Canada, pp. 4944–4953. Curran Associates Inc. (2018)
52. Zhang, X., Chen, X., Sun, M.: Towards a unifying logical framework for neural networks. Technical report, Peking University and University of Illinois Urbana-Champaign (2022). https://hdl.handle.net/2142/114412