Logic Tensor Networks

Samy Badreddine [a,b,*], Artur d'Avila Garcez [c], Luciano Serafini [d],
Michael Spranger [a,b]

[a] *Sony Computer Science Laboratories Inc, 3-14-13 Higashigotanda, 141-0022, Tokyo, Japan*
[b] *Sony AI Inc, 1-7-1 Konan, 108-0075, Tokyo, Japan*
[c] *City, University of London, Northampton Square, EC1V 0HB, London, United Kingdom*
[d] *Fondazione Bruno Kessler, Via Sommarive 18, 38123, Trento, Italy*

ARTICLE INFO

ABSTRACT

Attempts at combining logic and neural networks into neurosymbolic approaches have been on the increase in recent years. In a neurosymbolic system, symbolic knowledge assists deep learning, which typically uses a sub-symbolic distributed representation, to learn and reason at a higher level of abstraction. We present Logic Tensor Networks (LTN), a neurosymbolic framework that supports querying, learning and reasoning with both rich data and abstract knowledge about the world. LTN introduces a fully differentiable logical language, called Real Logic, whereby the elements of a first-order logic signature are grounded onto data using neural computational graphs and first-order fuzzy logic semantics. We show that LTN provides a uniform language to represent and compute efficiently many of the most important AI tasks such as multi-label classification, relational learning, data clustering, semi-supervised learning, regression, embedding learning and query answering. We implement and illustrate each of the above tasks with several simple explanatory examples using TensorFlow 2. The results indicate that LTN can be a general and powerful framework for neurosymbolic AI.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

Artificial Intelligence (AI) agents are required to learn from their surroundings and reason about what has been learned to make decisions, act in the world, or react to various stimuli. The latest Machine Learning (ML) has adopted mostly a pure sub-symbolic learning approach. Using distributed representations of entities, the latest ML performs quick decision-making without building a comprehensible model of the world. While achieving impressive results in computer vision, natural language, game playing, and multimodal learning, such approaches are known to be data inefficient and to struggle at out-of-distribution generalization. Although the use of appropriate inductive biases can alleviate such shortcomings, in general, sub-symbolic models lack comprehensibility. By contrast, symbolic AI is based on rich, high-level representations of the world that use human-readable symbols. By *rich knowledge*, we refer to logical representations which are more

expressive than propositional logic or propositional probabilistic approaches, and which can express knowledge using full first-order logic, including universal and existential quantification ($\forall x$ and $\exists y$), arbitrary n-ary relations over variables, e.g. $R(x, y, z, \dots)$, and function symbols, e.g. fatherOf(x), $x + y$, etc. Symbolic AI has achieved success at theorem proving, logical inference, and verification. However, it also has shortcomings when dealing with incomplete knowledge. It can be inefficient with large amounts of inaccurate data and lack robustness to outliers. Purely symbolic decision algorithms usually have high computational complexity making them impractical for the real world. It is now clear that the predominant approach to ML, where learning is based on recognizing the latent structures hidden in the data, is insufficient and may benefit from symbolic AI [17]. In this context, neurosymbolic AI, which stems from *neural* networks and *symbolic* AI, attempts to combine the strength of both paradigms (see [16,39,53] for recent surveys). That is to say, combine reasoning with complex representations of knowledge (knowledge-bases, semantic networks, ontologies, trees, and graphs) with learning from complex data (images, time series, sensorimotor data, natural language). Consequently, a main challenge for neurosymbolic AI is the grounding of symbols, including constants, functional and relational symbols, into real data, which is akin to the longstanding *symbol grounding* problem [30].

Logic Tensor Networks (LTN) are a neurosymbolic framework and computational model that supports learning and reasoning about data with rich knowledge. In LTN, one can represent and effectively compute the most important tasks of deep learning with a fully differentiable first-order logic language, called Real Logic, which adopts infinitely many truth-values in the interval [0, 1] [22,25]. In particular, LTN supports the specification and computation of the following AI tasks uniformly using the same language: data clustering, classification, relational learning, query answering, semi-supervised learning, regression, and embedding learning.

LTN and Real Logic were first introduced in [60]. Since then, LTN has been applied to different AI tasks involving perception, learning, and reasoning about relational knowledge. In [18,19], LTN was applied to semantic image interpretation whereby relational knowledge about objects was injected into deep networks for object relationship detection. In [6], LTN was evaluated on its capacity to perform reasoning about ontological knowledge. Furthermore, [7] shows how LTN can be used to learn an embedding of concepts into a latent real space by taking into consideration ontological knowledge about such concepts. In [3], LTN is used to annotate a reinforcement learning environment with prior knowledge and incorporate latent information into an agent. In [41], authors embed LTN in a state-of-the-art convolutional object detector. Extensions and generalizations of LTN have also been proposed in the past years, such as LYRICS [46] and *Differentiable Fuzzy Logic* (DFL) [66,67]. LYRICS provides an input language allowing one to define background knowledge using a first-order logic where predicate and function symbols are grounded onto any computational graph. DFL analyzes how a large collection of fuzzy logic operators behave in a differentiable learning setting. DFL also introduces new semantics for fuzzy logic implications called sigmoidal implications, and it shows that such semantics outperform other semantics in several semi-supervised machine learning tasks.

This paper provides a thorough description of the full formalism and several extensions of LTN. We show using an extensive set of explanatory examples, how LTN can be applied to solve many ML tasks with the help of logical knowledge. In particular, the earlier versions of LTN have been extended with: (1) *Explicit domain declaration:* constants, variables, functions and predicates are now domain typed (e.g. the constants *John* and *Paris* can be from the domain of *person* and *city*, respectively). The definition of structured domains is also possible (e.g. the domain *couple* can be defined as the Cartesian product of two domains of persons); (2) *Guarded quantifiers:* guarded universal and existential quantifiers now allow the user to limit the quantification to the elements that satisfy some Boolean condition, e.g. $\forall x : \text{age}(x) < 10 \, (\text{playsPiano}(x) \rightarrow \text{enfantProdige}(x))$ restricts the quantification to the cases where *age* is lower than 10; (3) *Diagonal quantification:* Diagonal quantification allows the user to write statements about specific tuples extracted in order from n variables. For example, if the variables capital and country both have k instances such that the i-th instance of capital corresponds to the i-th instance of country, one can write $\forall \text{Diag}(\text{capital}, \text{country}) \, \text{capitalOf}(\text{capital}, \text{country})$.

Inspired by the work of [67], this paper also extends the product t-norm configuration of LTN with the generalized mean aggregator, and it introduces solutions to the vanishing or exploding gradient problems. Finally, the paper formally defines a semantic approach to *refutation-based reasoning* in Real Logic to verify if a statement is a logical consequence of a knowledge base. Example 4.8 proves that this new approach can better capture logical consequences compared to simply querying unknown formulas after learning (as done in [6]).

The new version of LTN has been implemented in TensorFlow 2 [1]. Both the LTN library and the code for the examples used in this paper are available at https://github.com/logictensornetworks/logictensornetworks.

The remainder of the paper is organized as follows: In Section 2, we define and illustrate Real Logic as a fully-differentiable first-order logic. In Section 3, we specify learning and reasoning in Real Logic and its modeling into deep networks with Logic Tensor Networks (LTN). In Section 4, we illustrate the reach of LTN by investigating a range of learning problems from clustering to embedding learning. In Section 5, we place LTN in the context of the latest related work in neurosymbolic AI. In Section 6 we conclude and discuss directions for future work. The Appendix contains information about the implementation of LTN in TensorFlow 2, experimental set-ups, the different options for the differentiable logic operators, and a study of their relationship with gradient computations.

2. Real Logic

2.1. Syntax

Real Logic forms the basis of Logic Tensor Networks. Real Logic is defined on a first-order language $\mathcal{L}$ with a signature that contains a set $\mathcal{C}$ of constant symbols (objects), a set $\mathcal{F}$ of functional symbols, a set $\mathcal{P}$ of relational symbols (predicates), and a set $\mathcal{X}$ of variable symbols. $\mathcal{L}$-formulas allow us to specify relational knowledge with variables, e.g. the atomic formula is_friend(v_1, v_2) may state that the person v_1 is a friend of the person v_2, the formula $\forall x \forall y$(is_friend$(x, y) \rightarrow$ is_friend(y, x)) states that the relation is_friend is symmetric, and the formula $\forall x(\exists y$(Italian$(x) \land$ is_friend(x, y)))) states that every person has a friend that is Italian. Since we are interested in learning and reasoning in real-world scenarios where degrees of truth are often fuzzy and exceptions are present, formulas can be partially true, and therefore we adopt fuzzy semantics.

Objects can be of different types. Similarly, functions and predicates are typed. Therefore, we assume there exists a non-empty set of symbols $\mathcal{D}$ called *domain symbols*. To assign types to the elements of $\mathcal{L}$ we introduce the functions $\mathbf{D}$, $\mathbf{D_{in}}$ and $\mathbf{D_{out}}$ such that:

- $\mathbf{D} : \mathcal{X} \cup \mathcal{C} \rightarrow \mathcal{D}$. Intuitively, $\mathbf{D}(x)$ and $\mathbf{D}(c)$ returns the domain of a variable x or a constant c.
- $\mathbf{D_{in}} : \mathcal{F} \cup \mathcal{P} \rightarrow \mathcal{D}^*$, where $\mathcal{D}^*$ is the Kleene star of $\mathcal{D}$, that is the set of all finite sequences of symbols in $\mathcal{D}$. Intuitively, $\mathbf{D_{in}}(f)$ and $\mathbf{D_{in}}(p)$ returns the domains of the arguments of a function f or a predicate p. If f takes two arguments (for example, $f(x, y)$), $\mathbf{D_{in}}(f)$ returns two domains, one per argument.
- $\mathbf{D_{out}} : \mathcal{F} \rightarrow \mathcal{D}$. Intuitively, $\mathbf{D_{out}}(f)$ returns the range of a function symbol.

Real Logic may also contain propositional variables, as follows: if P is a 0-ary predicate with $\mathbf{D_{in}}(P) = \langle \rangle$ (the empty sequence of domains) then P is a propositional variable (an atom with truth-value in the interval $[0, 1]$).

A term is constructed recursively in the usual way from constant symbols, variables, and function symbols. An expression formed by applying a predicate symbol to an appropriate number of terms with appropriate domains is called an atomic formula, which evaluates to *true* or *false* in classical logic and a number in $[0, 1]$ in the case of Real Logic. We define the set of terms of the language as follows:

- each element t of $\mathcal{X} \cup \mathcal{C}$ is a term of the domain $\mathbf{D}(t)$;
- if t_i is a term of domain $\mathbf{D}(t_i)$ for $1 \leqslant i \leqslant n$ then $t_1 t_2 \ldots t_n$ (the sequence composed of t_1 followed by t_2 and so on, up to t_n) is a term of the domain $\mathbf{D}(t_1)\mathbf{D}(t_2) \ldots \mathbf{D}(t_n)$;
- if t is a term of the domain $\mathbf{D_{in}}(f)$ then $f(t)$ is a term of the domain $\mathbf{D_{out}}(f)$.

We allow the following set of formula in $\mathcal{L}$:

- $t_1 = t_2$ is an atomic formula for any terms t_1 and t_2 with $\mathbf{D}(t_1) = \mathbf{D}(t_2)$;
- $p(t)$ is an atomic formula if $\mathbf{D}(t) = \mathbf{D_{in}}(p)$;
- If ϕ and ψ are formula and $x_1, \ldots, x_n$ are n distinct variable symbols then $\diamond\phi$, $\phi \circ \psi$ and $Q x_1 \ldots x_n \phi$ are formula, where $\diamond$ is a unary connective, $\circ$ is a binary connective and Q is a quantifier.

We use $\diamond \in \{\neg\}$ (negation), $\circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}$ (conjunction, disjunction, implication and bi-conditional, respectively) and $Q \in \{\forall, \exists\}$ (universal and existential, respectively).

Example 1. Let Town denote the domain of towns in the world and People denote the domain of living people. Suppose that $\mathcal{L}$ contains the constant symbols Alice, Bob and Charlie of domain People, and Rome and Seoul of domain Town. Let x be a variable of domain People and u be a variable of domain Town. The term x, u (i.e. the sequence x followed by u) has domain People, Town which denotes the Cartesian product between People and Town (People $\times$ Town). Alice, Rome is interpreted as an element of the domain People, Town. Let lives_in be a predicate with input domain $\mathbf{D_{in}}$(lives_in) = People, Town. lives_in(Alice, Rome) is a well-formed expression, whereas lives_in(Bob, Charlie) is not.

2.2. Semantics of Real Logic

The semantics of Real Logic departs from the standard abstract semantics of First-order Logic (FOL). In Real Logic, domains are interpreted concretely by tensors in the real field.[1] Every object denoted by constants, variables, and terms, is interpreted as a tensor of real values. Functions are interpreted as real functions or tensor operations. Predicates are interpreted as functions or tensor operations projecting onto a value in the interval $[0, 1]$.

---

[1] In the rest of the paper, we commonly use "tensor" to designate "tensor in the real field".

To emphasize the fact that in Real Logic symbols are grounded onto real-valued features, we use the term *grounding*, denoted by $\mathcal{G}$, in place of *interpretation*.[2] Notice that this is different from the common use of the term *grounding* in logic, which indicates the operation of replacing the variables of a term or formula with constants or terms containing no variables. To avoid confusion, we use the synonym *instantiation* for this purpose. $\mathcal{G}$ associates a tensor of real numbers to any term of $\mathcal{L}$, and a real number in the interval $[0, 1]$ to any formula ϕ of $\mathcal{L}$. Intuitively, $\mathcal{G}(t)$ are the numeric features of the objects denoted by t, and $\mathcal{G}(\phi)$ represents the system's degree of confidence in the truth of ϕ; the higher the value, the higher the confidence.

2.2.1. Grounding domains and the signature

A grounding for a logical language $\mathcal{L}$ on the set of domains $\mathcal{D}$ provides the interpretation of both the domain symbols in $\mathcal{D}$ and the non-logical symbols in $\mathcal{L}$.

Definition 1. A grounding $\mathcal{G}$ associates to each domain $D \in \mathcal{D}$ a set $\mathcal{G}(D) \subseteq \bigcup_{n_1 \dots n_d \in \mathbb{N}^*} \mathbb{R}^{n_1 \times \cdots \times n_d}$. For every $D_1 \dots D_n \in \mathcal{D}^*$,

$$\mathcal{G}(D_1 \dots D_n) = \bigtimes_{i=1}^{n} \mathcal{G}(D_i), \text{ that is } \mathcal{G}(D_1) \times \mathcal{G}(D_2) \times \dots \times \mathcal{G}(D_n).$$

Notice that the elements in $\mathcal{G}(D)$ may be tensors of any rank d and any dimensions $n_1 \times \cdots \times n_d$, as $\mathbb{N}^*$ denotes the Kleene star of $\mathbb{N}$.[3]

Example 2. Let digit_images denote a domain of images of handwritten digits. If we use images of 256×256 RGB pixels, then $\mathcal{G}(\text{digit\_images}) \subseteq \mathbb{R}^{256 \times 256 \times 3}$. Let us consider the predicate is_digit(, 8). The terms , 8 have domains digit_images, digits. Any input to the predicate is a tuple in $\mathcal{G}(\text{digit\_images}, \text{digits}) = \mathcal{G}(\text{digit\_images}) \times \mathcal{G}(\text{digits})$.

A grounding assigns to each constant symbol c, a tensor $\mathcal{G}(c)$ in the domain $\mathcal{G}(\mathbf{D}(c))$; It assigns to a variable x a finite sequence of tensors $d_1 \dots d_k$, each in $\mathcal{G}(\mathbf{D}(x))$. These tensors represent the instances of x. Differently from in FOL where a variable is assigned to a single value of the domain of interpretations at a time, in Real Logic a variable is assigned to a sequence of values in its domain, the k examples of x. A grounding assigns to a function symbol f a function taking tensors from $\mathcal{G}(\mathbf{D_{in}}(f))$ as input, and producing a tensor in $\mathcal{G}(\mathbf{D_{out}}(f))$ as output. Finally, a grounding assigns to a predicate symbol p a function taking tensors from $\mathcal{G}(\mathbf{D_{in}}(p))$ as input, and producing a truth-value in the interval $[0, 1]$ as output.

Definition 2. A *grounding* $\mathcal{G}$ of $\mathcal{L}$ is a function defined on the signature of $\mathcal{L}$ that satisfies the following conditions:

1. $\mathcal{G}(x) = \langle d_1 \dots d_k \rangle \in \bigtimes_{i=1}^{k} \mathcal{G}(\mathbf{D}(x))$ for every variable symbol $x \in \mathcal{X}$, with $k \in \mathbb{N}_0^+$. Notice that $\mathcal{G}(x)$ is a sequence and not a set, meaning that the same value of $\mathcal{G}(\mathbf{D}(x))$ can occur multiple times in $\mathcal{G}(x)$, as is usual in a Machine Learning data set with "attributes" and "values";
2. $\mathcal{G}(f) \in \mathcal{G}(\mathbf{D_{in}}(f)) \to \mathcal{G}(\mathbf{D_{out}}(f))$ for every function symbol $f \in \mathcal{F}$;
3. $\mathcal{G}(p) \in \mathcal{G}(\mathbf{D_{in}}(p)) \to [0, 1]$ for every predicate symbol $p \in \mathcal{P}$.

If a grounding depends on a set of parameters θ, we denote it as $\mathcal{G}_\theta(\cdot)$ or $\mathcal{G}(\cdot \mid \theta)$ interchangeably. Section 4 describes how such parameters can be learned using the concept of satisfiability.

2.2.2. Grounding terms and atomic formulas

We now extend the definition of grounding to all first-order terms and atomic formulas. Before formally defining these groundings, we describe on a high level what happens when grounding terms that contain free variables.[4]

Let x be a variable that denotes people. As explained in Definition 2, x is grounded as an explicit sequence of k instances $(k = |\mathcal{G}(x)|)$. Consequently, a term height(x) is also grounded in k height values, each corresponding to one instance. We can generalize to expressions with multiple free variables, as shown in Example 3.

In the formal definition below, instead of considering a single term at a time, it is convenient to consider sequences of terms $\mathbf{t} = t_1 t_2 \dots t_k$ and define the grounding on $\mathbf{t}$ (with the definition of the grounding of a single term being derived as a special case). The fact that the sequence of terms $\mathbf{t}$ contains n distinct variables $x_1, \dots, x_n$ is denoted by $\mathbf{t}(x_1, \dots, x_n)$. The grounding of $\mathbf{t}(x_1, \dots, x_n)$, denoted by $\mathcal{G}(\mathbf{t}(x_1, \dots, x_n))$, is a tensor with n corresponding axes, one for each free variable, defined as follows:

Definition 3. Let $\mathbf{t}(x_1, \dots, x_n)$ be a sequence $t_1 \dots t_m$ of m terms containing n distinct variables $x_1, \dots, x_n$. Let each term t_i in $\mathbf{t}$ contain n_i variables $x_{j_{i1}}, \dots, x_{j_{in_i}}$.

---

[2] An interpretation is an assignment of truth-values *true* or *false*, or in the case of Real Logic a value in $[0, 1]$, to a formula. A model is an interpretation that maps a formula to *true*.

[3] A tensor of rank 0 corresponds to a scalar, a tensor of rank 1 to a vector, a tensor of rank 2 to a matrix and so forth, in the usual way.

[4] We assume the usual syntactic definition of free and bound variables in FOL. A variable is free if it is not bound by a quantifier ($\forall, \exists$).
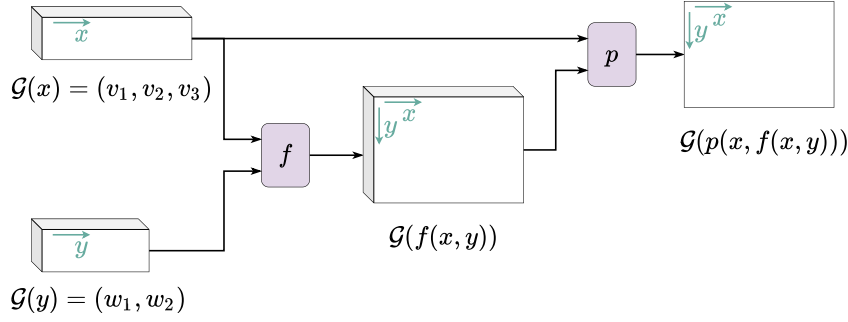
Fig. 1. Illustration of Example 3: $\overrightarrow{x}$ and $\overrightarrow{y}$ indicate dimensions associated with the free variables x and y. A tensor representing a term that includes a free variable x will have an axis $\overrightarrow{x}$. One can index $\overrightarrow{x}$ to obtain results calculated using each of the v_1, v_2 or v_3 values of x. In our graphical convention, the depth of the boxes indicates that the tensor can have *feature dimensions* (refer to the end of Example 3).

- $\mathcal{G}(\mathbf{t})$ is a tensor with dimensions $(|\mathcal{G}(x_1)|, \ldots, |\mathcal{G}(x_n)|)$ such that the element of this tensor indexed by $k_1, \ldots, k_n$, written as $\mathcal{G}(\mathbf{t})_{k_1 \ldots k_n}$, is equal to the concatenation of $\mathcal{G}(t_i)_{k_{j_{i1}} \ldots k_{j_{in_i}}}$ for $1 \leqslant i \leqslant m$;
- $\mathcal{G}(f(\mathbf{t}))_{i_1 \ldots i_n} = \mathcal{G}(f)(\mathcal{G}(\mathbf{t})_{i_1 \ldots i_n})$, i.e. the element-wise application of $\mathcal{G}(f)$ to $\mathcal{G}(\mathbf{t})$;
- $\mathcal{G}(p(\mathbf{t}))_{i_1 \ldots i_n} = \mathcal{G}(p)(\mathcal{G}(\mathbf{t})_{i_1 \ldots i_n})$, i.e. the element-wise application of $\mathcal{G}(p)$ to $\mathcal{G}(\mathbf{t})$.

If term t_i contains n_i variables $x_{j_1}, \ldots, x_{j_{n_i}}$ selected from $x_1, \ldots, x_n$ then $\mathcal{G}(t_i)_{k_{j_1} \ldots k_{j_{n_i}}}$ can be obtained from $\mathcal{G}(\mathbf{t})_{i_1 \ldots i_n}$ with an appropriate mapping of indices i to k.

Example 3. Suppose that $\mathcal{L}$ contains the variables x and y, the function f, the predicate p and the set of domains $\mathcal{D} = \{V, W\}$. Let $\mathbf{D}(x) = V$, $\mathbf{D}(y) = W$, $\mathbf{D_{in}}(f) = VW$, $\mathbf{D_{out}}(f) = W$ and $\mathbf{D}(p) = VW$. In what follows, an example of the grounding of $\mathcal{L}$ and $\mathcal{D}$ is shown on the left, and the grounding of some examples of possible terms and atomic formulas is shown on the right.

$$\mathcal{G}(V) = \mathbb{R}^+$$
$$\mathcal{G}(W) = \mathbb{R}^-$$
$$\mathcal{G}(x) = \langle v_1, v_2, v_3 \rangle$$
$$\mathcal{G}(y) = \langle w_1, w_2 \rangle$$
$$\mathcal{G}(p) : x, y \mapsto \sigma(x + y)$$
$$\mathcal{G}(f) : x, y \mapsto x \cdot y$$

$$\mathcal{G}(f(x, y)) = \begin{pmatrix} v_1 \cdot w_1 & v_1 \cdot w_2 \\ v_2 \cdot w_1 & v_2 \cdot w_2 \\ v_3 \cdot w_1 & v_3 \cdot w_2 \end{pmatrix}$$

$$\mathcal{G}(p(x, f(x, y))) = \begin{pmatrix} \sigma(v_1 + v_1 \cdot w_1) & \sigma(v_1 + v_1 \cdot w_2) \\ \sigma(v_2 + v_2 \cdot w_1) & \sigma(v_2 + v_2 \cdot w_2) \\ \sigma(v_3 + v_3 \cdot w_1) & \sigma(v_3 + v_3 \cdot w_2) \end{pmatrix}$$

Notice the dimensions of the results. $\mathcal{G}(f(x, y))$ and $\mathcal{G}(p(x, f(x, y)))$ return $|\mathcal{G}(x)| \times |\mathcal{G}(y)| = 3 \times 2$ values, one for each combination of individuals that occur in the variables. For functions, we can have additional dimensions associated to the output domain. Let us suppose a different grounding such that $\mathcal{G}(\mathbf{D_{out}}(f)) = \mathbb{R}^m$. Then the dimensions of $\mathcal{G}(f(x, y))$ would have been $|\mathcal{G}(x)| \times |\mathcal{G}(y)| \times m$, where $|\mathcal{G}(x)| \times |\mathcal{G}(y)|$ are the dimensions for indexing the free variables and m are dimensions associated to the output domain of f. Let us call the latter *feature dimensions*, as captioned in Fig. 1. Notice that $\mathcal{G}(p(x, f(x, y)))$ will always return a tensor with the exact dimensions $|\mathcal{G}(x)| \times |\mathcal{G}(y)| \times 1$ because, under any grounding, a predicate always returns a value in $[0, 1]$. Therefore, as the "feature dimensions" of predicates is always 1, we choose to "squeeze it" and not to represent it in our graphical convention (see Fig. 1, the box output by the predicate has no depth).

2.2.3. Connectives and quantifiers

The semantics of the connectives is defined according to the semantics of first-order fuzzy logic [28]. Conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$) and negation ($\neg$) are associated, respectively, with a t-norm (T), a t-conorm (S), a fuzzy implication (I) and a fuzzy negation (N) operation $\text{FuzzyOp} \in \{T, S, I, N\}$. Definitions of some common fuzzy operators are presented in Appendix B. Let ϕ and ψ be two formulas with free variables $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$, respectively. Let us assume that the first k variables are common to ϕ and ψ. Recall that $\diamond$ and $\circ$ denote the set of unary and binary connectives, respectively. Formally:

$$\mathcal{G}(\diamond\phi)_{i_1, \ldots, i_m} = \text{FuzzyOp}(\diamond)(\mathcal{G}(\phi)_{i_1, \ldots, i_m}) \tag{1}$$

$$\mathcal{G}(\phi \circ \psi)_{i_1, \ldots, i_{m+n-k}} = \text{FuzzyOp}(\circ)(\mathcal{G}(\phi)_{i_1, \ldots, i_k, i_{k+1}, \ldots, i_m} \mathcal{G}(\psi)_{i_1, \ldots, i_k, i_{m+1}, \ldots, i_{m+n-k}}) \tag{2}$$

In (2), $(i_1, \ldots, i_k)$ denote the indices of the k common variables, $(i_{k+1}, \ldots, i_m)$ denote the indices of the $m - k$ variables appearing only in ϕ, and $(i_{m+1}, \ldots, i_{m+n-k})$ denote the indices of the $n - k$ variables appearing only in ψ. Intuitively,
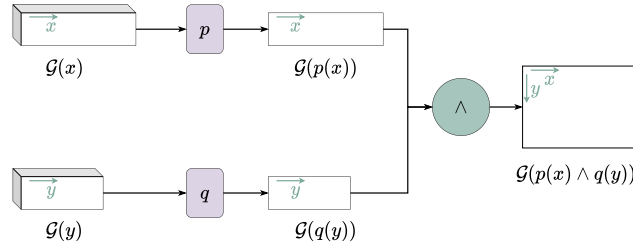
Fig. 2. Illustration of an element-wise operator implementing conjunction ($p(x) \wedge q(y)$). We assume that x and y are two different variables. The result has one number in the interval $[0, 1]$ to every combination of individuals from $\mathcal{G}(x)$ and $\mathcal{G}(y)$.
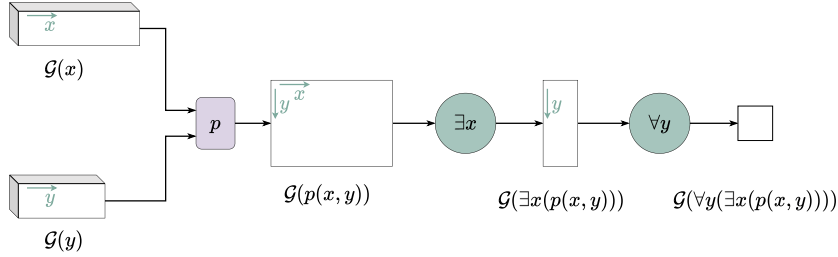

Fig. 3. Illustration of an aggregation operation implementing quantification ($\forall y \exists x$) over variables x and y. We assume that x and y have different domains. The result is a single number in the interval $[0, 1]$.

$\mathcal{G}(\phi \circ \psi)$ is a tensor whose elements are obtained by applying FuzzyOp($\circ$) element-wise to every combination of individuals from $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ (see Fig. 2).

The semantics of the quantifiers ($\{\forall, \exists\}$) is defined with the use of aggregation. Let Agg be a symmetric and continuous aggregation operator, Agg : $\bigcup_{n \in \mathbb{N}} [0, 1]^n \to [0, 1]$. An analysis of suitable aggregation operators is presented in Appendix B. For every formula ϕ containing $x_1, \ldots, x_n$ free variables, suppose, without loss of generality, that quantification applies to the first h variables. We shall therefore apply Agg to the first h axes of $\mathcal{G}(\phi)$, as follows:

$$\mathcal{G}(Q x_1, \ldots, x_h(\phi))_{i_{h+1}, \ldots, i_n} = \underset{\substack{i_1 = 1, \ldots, |\mathcal{G}(x_1)| \\ \vdots \\ i_h = 1, \ldots, |\mathcal{G}(x_h)|}}{\mathrm{Agg}(Q)} \mathcal{G}(\phi)_{i_1, \ldots, i_h, i_{h+1}, \ldots, i_n} \tag{3}$$

where Agg(Q) is the aggregation operator associated with the quantifier Q. Intuitively, we obtain $\mathcal{G}(Q x_1, \ldots, x_h(\phi))$ by reducing the dimensions associated with $x_1, \ldots, x_h$ using the operator Agg(Q) (see Fig. 3).

Notice that the above *grounded* semantics can assign different meanings to the three formulas:

$$\forall xy\big(\phi(x, y)\big) \quad \forall x\big(\forall y\big(\phi(x, y)\big)\big) \quad \forall y\big(\forall x\big(\phi(x, y)\big)\big)$$

The semantics of the three formulas will coincide if the aggregation operator is bi-symmetric.

LTN also allows the following form of quantification, here called diagonal quantification (Diag):

$$\mathcal{G}(Q \, \mathrm{Diag}(x_1, \ldots, x_h)(\phi))_{i_{h+1}, \ldots, i_n} = \underset{i = 1, \ldots, \min_{1 \leqslant j \leqslant h} |\mathcal{G}(x_j)|}{\mathrm{Agg}(Q)} \mathcal{G}(\phi)_{i, \ldots, i, i_{h+1}, \ldots, i_n} \tag{4}$$

Diag($x_1, \ldots, x_h$) quantifies over specific tuples such that the i-th tuple contains the i-th instance of each of the variables in the argument of Diag, under the assumption that all variables in the argument are grounded onto sequences with the same number of instances. Diag($x_1, \ldots, x_h$) is called diagonal quantification because it quantifies over the diagonal of $\mathcal{G}(\phi)$ along the axes associated with $x_1 \ldots x_h$, although in practice only the diagonal is built and not the entire $\mathcal{G}(\phi)$, as shown in Fig. 4. For example, given a data set with samples x and target labels y, if looking to write a statement $p(x, y)$ that holds true for each pair of sample and label, one can write $\forall \mathrm{Diag}(x, y) \, p(x, y)$ given that $|\mathcal{G}(x)| = |\mathcal{G}(y)|$. As another example, given two variables x and y whose groundings contain 10 instances of x and y each, the expression $\forall \mathrm{Diag}(x, y) \, p(x, y)$ produces 10 results such that the i-th result corresponds to the i-th instances of each grounding. Without
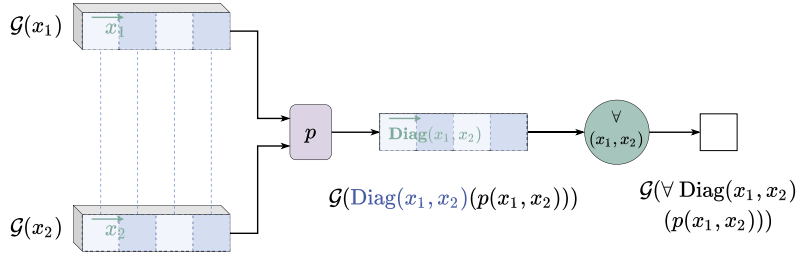
Fig. 4. Diagonal Quantification: Diag(x_1, x_2) quantifies over specific tuples only, such that the i-th tuple contains the i-th instances of the variables x_1 and x_2 in the groundings $\mathcal{G}(x_1)$ and $\mathcal{G}(x_2)$, respectively. Diag(x_1, x_2) assumes, therefore, that x_1 and x_2 have the same number of instances as in the case of samples x_1 and their labels x_2 in a typical supervised learning tasks.

Diag, the expression would be evaluated for all 10×10 combinations of the elements in $\mathcal{G}(x)$ and $\mathcal{G}(y)$.[5] Diag will find much application in the examples and experiments to follow.

2.3. Guarded quantifiers

In many situations, one may wish to quantify over a set of elements of a domain whose grounding satisfies some condition. In particular, one may wish to express such condition using formulas of the language of the form:

$$\forall y \, (\exists x : \text{age}(x) > \text{age}(y) \, (\text{parent}(x, y))) \tag{5}$$

The grounding of such a formula is obtained by aggregating the values of parent(x, y) only for the instances of x that satisfy the condition age$(x) >$ age(y), that is: (see Fig. 5)

$$\underset{j=1,\dots,|\mathcal{G}(y)|}{\text{Agg}(\forall)} \quad \underset{\substack{i=1,\dots,|\mathcal{G}(x)| \ s.t. \\ \mathcal{G}(\text{age}(x))_i > \mathcal{G}(\text{age}(y))_j}}{\text{Agg}(\exists)} \quad \mathcal{G}(\text{parent}(x, y))_{i,j}$$

The evaluation of which tuple is safe is purely symbolic and non-differentiable. Guarded quantifiers operate over only a subset of the variables, when this symbolic knowledge is crisp and available. More generally, in what follows, m is a symbol representing the condition, which we shall call a *mask*, and $\mathcal{G}(m)$ associates a function[6] returning a Boolean to m.

$$\mathcal{G}(Q \ x_1, \dots, x_h : m(x_1, \dots, x_n)(\phi))_{i_{h+1},\dots,i_n} \overset{\text{def}}{=} \underset{\substack{i_1=1,\dots,|\mathcal{G}(x_1)| \\ \vdots \\ i_h=1,\dots,|\mathcal{G}(x_h)| \ s.t. \\ \mathcal{G}(m)(\mathcal{G}(x_1)_{i_1},\dots,\mathcal{G}(x_n)_{i_n})}}{\text{Agg}(Q)} \quad \mathcal{G}(\phi)_{i_1,\dots,i_h,i_{h+1},\dots,i_n} \tag{6}$$

Notice that the semantics of a guarded sentence $\forall x : m(x)(\phi(x))$ is different than the semantics of $\forall x(m(x) \rightarrow \phi(x))$. In crisp and traditional FOL, the two statements would be equivalent. In Real Logic, they can give different results. Let $\mathcal{G}(x)$ be a sequence of 3 values, $\mathcal{G}(m(x)) = (0, 1, 1)$ and $\mathcal{G}(\phi(x)) = (0.2, 0.7, 0.8)$. Only the second and third instances of x are safe, that is, are in the masked subset. Let $\rightarrow$ be defined using the Reichenbach operator $I_R(a, b) = 1 - a + ab$ and $\forall$ be defined using the mean operator. We have $\mathcal{G}(\forall x(m(x) \rightarrow \phi(x))) = \frac{1+0.7+0.8}{3} = 0.833\dots$ whereas $\mathcal{G}(\forall x : m(x)(\phi(x))) = \frac{0.7+0.8}{2} = 0.75$. Also, in the computational graph of the guarded sentence, there are no gradients attached to the instances that do not verify the mask. Similarly, the semantics of $\exists x : m(x)(\phi(x))$ is not equivalent to that of $\exists x(m(x) \wedge \phi(x))$.

2.4. Stable Product Real Logic

It has been shown in [67] that not all first-order fuzzy logic semantics are equally suited for gradient-descent optimization. Many fuzzy logic operators can lead to vanishing or exploding gradients. Some operators are also *single-passing*, in that they propagate gradients to only one input at a time.

In general, the best performing symmetric configuration[7] for the connectives uses the product t-norm T_P for conjunction, its dual t-conorm S_P for disjunction, standard negation N_S, and the Reichenbach implication I_R (the corresponding S-

---

[5] Notice how Diag is not simply "syntactic sugar" for creating a new variable pairs_xy by stacking pairs of examples from $\mathcal{G}(x)$ and $\mathcal{G}(y)$. If the groundings of x and y have incompatible ranks (for instance, if x denotes images and y denotes their labels), stacking them in a tensor $\mathcal{G}(\text{pairs\_xy})$ is non-trivial, requiring several reshaping operations.

[6] In some edge cases, a masking may produce an empty sequence, e.g. if for some value of $\mathcal{G}(y)$, there is no value in $\mathcal{G}(x)$ that satisfies age$(x) >$ age(y), we resort to the concept of an *empty semantics*: $\forall$ returns 1 and $\exists$ returns 0.

[7] We define a symmetric configuration as a set of fuzzy operators such that conjunction and disjunction are defined by a t-norm and its dual t-conorm, respectively, and the implication operator is derived from such conjunction or disjunction operators and standard negation (cf. Appendix B for details). In [67], van Krieken et al. also analyze non-symmetric configurations and even operators that do not strictly verify fuzzy logic semantics.
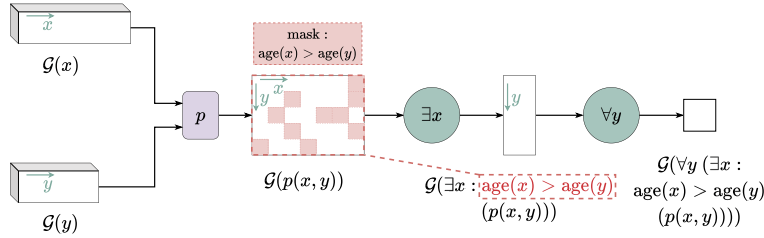
Fig. 5. Example of Guarded Quantification: one can filter out elements of the various domains that do not satisfy some condition before the aggregation operators for ∀ and ∃ are applied.

Implication to the above operators). This subset of Real Logic where the grounding of the connectives is restricted to the product configuration is called *Product Real Logic* in [67]. Given a and b two truth-values in $[0, 1]$:

$$\neg : N_S(a) = 1 - a \tag{7}$$

$$\wedge : T_P(a, b) = ab \tag{8}$$

$$\vee : S_P(a, b) = a + b - ab \tag{9}$$

$$\rightarrow : I_R(a, b) = 1 - a + ab \tag{10}$$

Appropriate aggregators for ∃ and ∀ are the generalized mean A_{pM} with $p \geqslant 1$ to approximate the existential quantification, and the generalized mean w.r.t. the error A_{pME} with $p \geqslant 1$ to approximate the universal quantification. They can be understood as a smooth maximum and a smooth minimum, respectively. Given n truth-values $a_1, \ldots, a_n$ all in $[0, 1]$:

$$\exists : A_{pM}(a_1, \ldots, a_n) = \left(\frac{1}{n} \sum_{i=1}^{n} a_i^p \right)^{\frac{1}{p}} \qquad p \geqslant 1 \tag{11}$$

$$\forall : A_{pME}(a_1, \ldots, a_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - a_i)^p \right)^{\frac{1}{p}} \qquad p \geqslant 1 \tag{12}$$

A_{pME} measures the power of the deviation of each value from the ground truth 1. With $p = 2$, it is equivalent to $1 - \mathrm{RMSE}(\boldsymbol{a}, \mathbf{1})$, where RMSE is the root-mean-square error, $\boldsymbol{a}$ is the vector of truth-values and $\mathbf{1}$ is a vector of 1's.

The intuition behind the choice of p is that the higher that p is, the more weight that A_{pM} (resp. A_{pME}) will give to *true* (resp. *false*) truth-values, converging to the max (resp. min) operator. Therefore, the value of p can be seen as a hyper-parameter as it offers flexibility to account for outliers in the data depending on the application.

Nevertheless, *Product Real Logic* still has the following gradient problems: $T_P(a, b)$ has vanishing gradients on the edge case $a = b = 0$; $S_P(a, b)$ has vanishing gradients on the edge case $a = b = 1$; $I_R(a, b)$ has vanishing gradients on the edge case $a = 0, b = 1$; $A_{pM}(a_1, \ldots, a_n)$ has exploding gradients when $\sum_i (a_i)^p$ tends to 0; $A_{pME}(a_1, \ldots, a_n)$ has exploding gradients when $\sum_i (1 - a_i)^p$ tends to 0 (see Appendix C for details).

To address these problems, we define the projections π_0 and π_1 below with ϵ an arbitrarily small positive real number:

$$\pi_0 : [0, 1] \rightarrow]0, 1] : a \rightarrow (1 - \epsilon)a + \epsilon \tag{13}$$

$$\pi_1 : [0, 1] \rightarrow [0, 1[: a \rightarrow (1 - \epsilon)a \tag{14}$$

We then derive the following stable operators to produce what we call the *Stable Product Real Logic* configuration:

$$N_S'(a) = N_S(a) \tag{15}$$

$$T_P'(a, b) = T_P(\pi_0(a), \pi_0(b)) \tag{16}$$

$$S_P'(a, b) = S_P(\pi_1(a), \pi_1(b)) \tag{17}$$

$$I_R'(a, b) = I_R(\pi_0(a), \pi_1(b)) \tag{18}$$

$$A_{pM}'(a_1, \ldots, a_n) = A_{pM}(\pi_0(a_1), \ldots, \pi_0(a_n)) \qquad p \geqslant 1 \tag{19}$$

$$A_{pME}'(a_1, \ldots, a_n) = A_{pME}(\pi_1(a_1), \ldots, \pi_1(a_n)) \qquad p \geqslant 1 \tag{20}$$

It is important noting that the conjunction operator in stable product semantics is not a T-norm.[8] $T_P'(a, b)$ does not satisfy identity in $[0, 1[$ since for any $0 \leqslant a < 1$, $T_P'(a, 1) = (1 - \epsilon)a + \epsilon \neq a$, although ϵ can be chosen arbitrarily small.

---

[8] Recall that a T-norm is a function $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$ satisfying commutativity, monotonicity, associativity and identity, that is, $T(a, 1) = a$.

In the experimental evaluations reported in Section 4, we find that the adoption of the stable product semantics is an important practical step to improve the numerical stability of the learning system.

3. Learning, reasoning, and querying in Real Logic

In Real Logic, one can define the tasks of *learning*, *reasoning* and *query-answering*. Given a Real Logic theory that represents the knowledge of an agent at a given time, *learning* is the task of making generalizations from specific observations obtained from data. This is often called inductive inference. *Reasoning* is the task of deriving what knowledge follows from the facts which are currently known. *Query answering* is the task of evaluating the truth value of a certain logical expression (called a query), or finding the set of objects in the data that evaluate a certain expression to *true*. In what follows, we define and exemplify each of these tasks. To do so, we first need to specify which types of knowledge can be represented in Real Logic.

3.1. Representing knowledge with Real Logic

In logic-based knowledge representation systems, knowledge is represented by logical formulas whose intended meanings are propositions about a domain of interest. The connection between the symbols occurring in the formulas and what holds in the domain is not represented in the knowledge base and is left implicit since it does not have any effect on the logic computations. In Real Logic, by contrast, the connection between the symbols and the domain is represented explicitly in the language by the grounding $\mathcal{G}$, which plays an important role in both learning and reasoning. $\mathcal{G}$ is an integral part of the knowledge represented by Real Logic. A Real Logic knowledge base is therefore defined by the formulas of the logical language and knowledge about the domain in the form of groundings obtained from data. The following types of knowledge can be represented in Real Logic.

3.1.1. Knowledge through symbol groundings

Boundaries for domain grounding. These are constraints specifying that the value of a certain logical expression must be within a certain range. For instance, one may specify that the domain D must be interpreted in the $[0, 1]$ hyper-cube or in the standard n-simplex, i.e. the set $d_1, \ldots, d_n \in (\mathbb{R}^+)^n$ such that $\sum_i d_i = 1$. Other intuitive examples of range constraints include the elements of the domain "color" grounded onto points in $[0, 1]^3$ such that every element is associated with the triplet of values (R, G, B) with $R, G, B \in [0, 1]$, or the range of a function $age(x)$ as an integer between 0 and 100.

Explicit definition of grounding for symbols. Knowledge can be more strictly incorporated by fixing the grounding of some symbols. If a constant c denotes an object with known features $\mathbf{v}_c \in \mathbb{R}^n$, we can fix its grounding $\mathcal{G}(c) = \mathbf{v}_c$. Training data that consists in a set of n data items such as n images (or tuples known as training examples) can be specified in Real Logic by n constants, e.g. $img_1, img_2, \ldots, img_n$, and by their groundings, e.g. $\mathcal{G}(img_1) = $, $\mathcal{G}(img_2) = $, $\ldots$, $\mathcal{G}(img_n) = $. These can be gathered in a variable *imgs*. A binary predicate sim that measures the similarity of two objects can be grounded as, e.g., a cosine similarity function of two vectors $\mathbf{v}$ and $\mathbf{w}$, $(\mathbf{v}, \mathbf{w}) \mapsto \frac{\mathbf{v} \cdot \mathbf{w}}{||\mathbf{v}|| \, ||\mathbf{w}||}$. The output layer of the neural network associated with a multi-class single-label predicate $P(x, class)$ can be a `softmax` function normalizing the output such that it guarantees exclusive classification, i.e. $\sum_i P(x, i) = 1$.[9] Grounding of constants and functions allows the computation of the grounding of their results. If, for example, $\mathcal{G}(transp)$ is the function that transposes a matrix then $\mathcal{G}(transp(img_1)) = $.

Parametric definition of grounding for symbols. Here, the exact grounding of a symbol σ is not known, but it is known that it can be obtained by finding a set of real-valued parameters, that is, via learning. To emphasize this fact, we adopt the notation $\mathcal{G}(\sigma) = \mathcal{G}(\sigma \mid \boldsymbol{\theta}_\sigma)$ where $\boldsymbol{\theta}_\sigma$ is the set of parameter values that determines the value of $\mathcal{G}(\sigma)$. The typical example of parametric grounding for constants is the learning of an embedding. Let $emb(word \mid \boldsymbol{\theta}_{emb})$ be a word embedding with parameters $\boldsymbol{\theta}_{emb}$ which takes as input a word and returns its embedding in $\mathbb{R}^n$. If the words of a vocabulary $W = \{w_1, \ldots, w_{|W|}\}$ are constant symbols, their groundings $\mathcal{G}(w_i \mid \boldsymbol{\theta}_{emb})$ are defined parametrically w.r.t. $\boldsymbol{\theta}_{emb}$ as $emb(w_i \mid \boldsymbol{\theta}_{emb})$. An example of parametric grounding for a function symbol f is to assume that $\mathcal{G}(f)$ is a linear function such that $\mathcal{G}(f): \mathbb{R}^m \to \mathbb{R}^n$ maps each $\mathbf{v} \in \mathbb{R}^m$ into $\boldsymbol{A}_f \mathbf{v} + \boldsymbol{b}_f$, with $\boldsymbol{A}_f$ a matrix of real numbers and $\boldsymbol{b}$ a vector of real numbers. In this case, $\mathcal{G}(f) = \mathcal{G}(f \mid \boldsymbol{\theta}_f)$, where $\boldsymbol{\theta}_f = \{\boldsymbol{A}_f, \boldsymbol{b}_f\}$. Finally, the grounding of a predicate symbol can be given, for example, by a neural network N with parameters $\boldsymbol{\theta}_N$. As an example, consider a neural network N trained for image classification into n classes: $cat, dog, horse$, etc. N takes as input a vector $\mathbf{v}$ of pixel values and produces as output a vector $\mathbf{y} = (y_{cat}, y_{dog}, y_{horse}, \ldots)$ in $[0, 1]^n$ such that $\mathbf{y} = N(\mathbf{v} \mid \boldsymbol{\theta}_N)$, where y_c is the probability that input image $\mathbf{v}$ is of class c. In case classes are, alternatively, chosen to be represented by unary predicate symbols such as cat($\mathbf{v}$), dog($\mathbf{v}$), horse($\mathbf{v}$),…then $\mathcal{G}(\text{cat}(\mathbf{v})) = N(\mathbf{v} \mid \theta_N)_{cat}$, $\mathcal{G}(\text{dog}(\mathbf{v})) = N(\mathbf{v} \mid \theta_N)_{dog}$, $\mathcal{G}(\text{horse}(\mathbf{v})) = N(\mathbf{v} \mid \theta_N)_{horse}$, etc.

---

[9] Notice that `softmax` is often used as the last layer in neural networks to turn logits into a probability distribution. However, we do not use the `softmax` function as such here. Instead, we use it here to enforce an exclusivity constraint on satisfiability scores.

3.1.2. Knowledge through formulas

Factual propositions. Knowledge about the properties of specific objects in the domain is represented, as usual, by logical propositions, as exemplified below: Suppose that it is known that img_1 is a number eight, img_2 is a number nine, and img_n is a number two. This can be represented by adding the following facts to the knowledge-base: nine(img_1), eight$(img_2), \ldots,$ two(img_n). Supervised learning, that is, learning with the use of training examples which include target values (labeled data), is specified in Real Logic by combining grounding definitions and factual propositions. For example, the fact that an image is a positive example for the class nine and a negative example for the class eight is specified by defining $\mathcal{G}(img_1) = $ alongside the propositions nine(img_1) and $\neg$eight(img_1). Notice how semi-supervision can be specified naturally in Real Logic by adding propositions containing disjunctions, e.g. eight$(img_1) \vee$ nine(img_1), which state that img_1 is either an eight or a nine (or both). Finally, relational learning can be achieved by relating logically multiple objects (defined as constants or variables or even as more complex sequences of terms) such as e.g.: nine$(img_1) \rightarrow \neg$nine(img_2) (if img_1 is a nine then img_2 is not a nine) or nine$(img) \rightarrow \neg$eight(img) (if an image is a nine then it is not an eight). The use of more complex knowledge including the use of variables such as img above is the topic of *generalized propositions*, discussed next.

Generalized propositions. General knowledge about all or some of the objects of some domains can be specified in Real Logic by using first-order logic formulas with quantified variables. This general type of knowledge allows one to spec-ify arbitrary constraints on the groundings independently from the specific data available. It allows one to specify, in a concise way, knowledge that holds true for all the objects of a domain. This is especially useful in Machine Learning in the semi-supervised and unsupervised settings, where there is no specific knowledge about a single individual. For example, as part of a task of multi-label classification with constraints on the labels [12], a positive label constraint may express that if an example is labeled with $l_1, \ldots, l_k$ then it should also be labeled with l_{k+1}. This can be specified in Real Logic with a universally quantified formula: $\forall x \ (l_1(x) \wedge \cdots \wedge l_k(x) \rightarrow l_{k+1}(x))$.[10] Another example of soft con-straints used in Statistical Relational Learning associates the labels of related examples. For instance, in Markov Logic Networks [54], as part of the well-known *Smokers and Friends* example, people who are smokers are associated by the friendship relation. In Real Logic, the formula $\forall xy \ ((\text{smokes}(x) \wedge \text{friend}(x, y)) \rightarrow \text{smokes}(y))$ would be used to encode the soft constraint that friends of smokers are normally smokers.

3.1.3. Knowledge through fuzzy semantics

Definition for operators. The grounding of a formula ϕ depends on the operators approximating the connectives and quan-tifiers that appear in ϕ. Different operators give different interpretations of the satisfaction associated with the formula. For instance, the operator $A_{pME}(a_1, \ldots, a_n)$ that approximates universal quantification can be understood as a smooth minimum. It depends on a hyper-parameter p (the exponent used in the generalized mean). If $p = 1$ then $A_{pME}(a_1, \ldots, a_n)$ corresponds to the arithmetic mean. As p increases, given the same input, the value of the uni-versally quantified formula will decrease as A_{pME} converges to the min operator. To define how strictly the universal quantification should be interpreted in each proposition, one can use different values of p for different propositions of the knowledge base. For instance, a formula $\forall x \ P(x)$ where A_{pME} is used with a low value for p will in fact denote that P holds for *some x*, whereas a formula $\forall x \ Q(x)$ with a higher p may denote that Q holds for *most x*.

3.1.4. Satisfiability

In summary, a Real Logic knowledge-base has three components: the first describes knowledge about the grounding of symbols (domains, constants, variables, functions, and predicate symbols); the second is a set of closed logical formulas describing factual propositions and general knowledge; the third lies in the operators and the hyperparameters used to evaluate each formula. The definition that follows formalizes this notion.

Definition 4 *(Theory/Knowledge-base).* A *theory* of Real Logic is a triple $\mathcal{T} = \langle \mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta} \rangle$, where $\mathcal{K}$ is a set of closed first-order logic formulas defined on the set of symbols $S = D \cup X \cup C \cup F \cup P$ denoting, respectively, domains, variables, constants, function and predicate symbols; $\mathcal{G}(\cdot \mid \boldsymbol{\theta})$ is a parametric grounding for all the symbols $s \in S$ and all the logical operators; and $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}_s\}_{s \in S}$ is the hypothesis space for each set of parameters $\boldsymbol{\theta}_s$ associated with symbol s.

Learning and reasoning in a Real Logic theory are both associated with searching and applying the set of values of parameters $\boldsymbol{\theta}$ from the hypothesis space $\boldsymbol{\Theta}$ that maximize the satisfaction of the formulas in $\mathcal{K}$. We use the term *grounded theory*, denoted by $\langle \mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}} \rangle$, to refer to a Real Logic theory with a specific set of learned parameter values. This idea shares some similarity with the weighted MAX-SAT problem [42], where the weights for formulas in $\mathcal{K}$ are given by their fuzzy truth-values obtained by choosing the parameter values of the grounding. To define this optimization problem, we aggregate the truth-values of all the formulas in $\mathcal{K}$ by selecting a *formula aggregating* operator SatAgg : $[0, 1]^* \rightarrow [0, 1]$.

Definition 5. The *satisfiability* of a theory $\mathcal{T} = \langle \mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}} \rangle$ with respect to the *aggregating operator* SatAgg is defined as $\text{SatAgg}_{\phi \in \mathcal{K}} \ \mathcal{G}_{\boldsymbol{\theta}}(\phi)$.

---

[10] This can also be specified using a guarded quantifier $\forall x : ((l_1(x) \wedge \cdots \wedge l_k(x)) > th) \ l_{k+1}(x)$ where th is a threshold value in $[0, 1]$.

3.2. Learning

Given a Real Logic theory $\mathcal{T} = (\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta})$, *learning* is the process of searching for the set of parameter values $\boldsymbol{\theta}^*$ that maximize the satisfiability of $\mathcal{T}$ w.r.t. a given aggregator:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmax}} \; \underset{\phi \in \mathcal{K}}{\operatorname{SatAgg}} \mathcal{G}_{\boldsymbol{\theta}}(\phi)$$

Notice that with this general formulation, one can learn the grounding of constants, functions, and predicates. The learning of the grounding of constants corresponds to the learning of *embeddings*. The learning of the grounding of functions corresponds to the learning of *generative* models or a *regression* task. Finally, the learning of the grounding of predicates corresponds to a *classification* task in Machine Learning.

In some cases, it is useful to impose some regularization (as done customarily in ML) on the set of parameters $\boldsymbol{\theta}$, thus encoding a preference on the hypothesis space $\boldsymbol{\Theta}$, such as a preference for smaller parameter values. In this case, learning is defined as follows:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmax}} \left(\underset{\phi \in \mathcal{K}}{\operatorname{SatAgg}} \mathcal{G}_{\boldsymbol{\theta}}(\phi) - \lambda \mathrm{R}(\boldsymbol{\theta}) \right)$$

where $\lambda \in \mathbb{R}^+$ is the regularization parameter and R is a regularization function, e.g. L_1 or L_2 regularization, that is, $L_1(\boldsymbol{\theta}) = \sum_{\theta \in \boldsymbol{\theta}} |\theta|$ and $L_2(\boldsymbol{\theta}) = \sum_{\theta \in \boldsymbol{\theta}} \theta^2$.

LTN can generalize and extrapolate when querying formulas grounded with unseen data (for example, new individuals from a domain), using knowledge learned with previous groundings (for example, re-using a trained predicate). This is explained in Section 3.3.

3.3. Querying

Given a grounded theory $\mathcal{T} = (\mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}})$, *query answering* allows one to check if a certain fact is true (or, more precisely, by how much it is true since in Real Logic truth-values are real numbers in the interval $[0, 1]$). There are various types of queries that can be asked of a grounded theory.

A first type of query is called *truth queries*. Any formula in the language of $\mathcal{T}$ can be a truth query. The answer to a truth query ϕ_q is the truth value of ϕ_q obtained by computing its grounding, i.e. $\mathcal{G}_{\boldsymbol{\theta}}(\phi_q)$. Notice that, if ϕ_q is a closed formula, the answer is a scalar in $[0, 1]$ denoting the truth-value of ϕ_q according to $\mathcal{G}_{\boldsymbol{\theta}}$, if ϕ_q contains n free variables $x_1, \dots, x_n$, the answer to the query is a tensor of order n such that the component indexed by $i_1 \dots i_n$ is the truth-value of ϕ_q evaluated in $\mathcal{G}_{\boldsymbol{\theta}}(x_1)_{i_1}, \dots, \mathcal{G}_{\boldsymbol{\theta}}(x_n)_{i_n}$.

The second type of query is called *value queries*. Any term in the language of $\mathcal{T}$ can be a value query. The answer to a value query t_q is a tensor of real numbers obtained by computing the grounding of the term, i.e. $\mathcal{G}_{\boldsymbol{\theta}}(t_q)$. Analogously to truth queries, the answer to a value query is a "tensor of tensors" if t_q contains variables. Using value queries, one can inspect how a constant or a term, more generally, is embedded in the manifold.

The third type of query is called *generalization truth queries*. With generalization truth queries, we are interested in knowing the truth-values of formulas when these are applied to a new (unseen) set of objects of a domain, such as a validation or a test set of examples typically used in the evaluation of machine learning systems. A generalization truth query is a pair $(\phi_q(x), \boldsymbol{U})$, where ϕ_q is a formula with a free variable x and $\boldsymbol{U} = (\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(k)})$ is a set of unseen examples whose dimensions are compatible with those of the domain of x. The answer to the query $(\phi_q(x), \boldsymbol{U})$ is $\mathcal{G}_{\boldsymbol{\theta}}(\phi_q(x))$ for x taking each value $\mathbf{u}^{(i)}$, $1 \leqslant i \leqslant k$, in $\boldsymbol{U}$. The result of this query is therefore a vector of $|\boldsymbol{U}|$ truth-values corresponding to the evaluation of ϕ_q on new data $\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(k)}$.

The fourth and final type of query is *generalization value queries*. These are analogous to generalization truth queries with the difference that they evaluate a term $t_q(x)$, and not a formula, on new data $\boldsymbol{U}$. The result, therefore, is a vector of $|\boldsymbol{U}|$ values corresponding to the evaluation of the trained model on a regression task using test data $\boldsymbol{U}$.

3.4. Reasoning

3.4.1. Logical consequence in Real Logic

From a pure logic perspective, reasoning is the task of verifying if a formula is a logical consequence of a set of formulas. This can be achieved semantically using model theory ($\models$) or syntactically via a proof theory ($\vdash$). To characterize reasoning in Real Logic, we adapt the notion of logical consequence for fuzzy logic provided in [9]: A formula ϕ is a fuzzy logical consequence of a finite set of formulas Γ, in symbols $\Gamma \models \phi$ if for every fuzzy interpretation f, if all the formulas in Γ are true (i.e. evaluate to 1) in f then ϕ is true in f. In other words, every model of Γ is a model of ϕ. A direct application of this definition to Real Logic is not practical since in most practical cases the level of satisfiability of a grounded theory $(\mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}})$ will not be equal to 1. We therefore define an interval $[q, 1]$ with $\frac{1}{2} < q < 1$ and assume that a formula is true if its truth-value is in the interval $[q, 1]$. This leads to the following definition:

Definition 6. A closed formula ϕ is a logical consequence of a knowledge-base $(\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta})$, in symbols $(\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta}) \models_q \phi$, if, for every grounded theory $\langle \mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}} \rangle$, if $\text{SatAgg}(\mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}}) \geqslant q$ then $\mathcal{G}_{\boldsymbol{\theta}}(\phi) \geqslant q$.

3.4.2. Reasoning by optimization

Logical consequence by direct application of Definition 6 requires querying the truth value of ϕ for a potentially infinite set of groundings. Therefore, we consider in practice the following directions:

Reasoning Option 1 (querying after learning) This is approximate logical inference by considering only the grounded theories that maximally satisfy $(\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta})$. We therefore define that ϕ is a *brave logical consequence* of a Real Logic knowledge-base $(\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta})$ if $\mathcal{G}_{\boldsymbol{\theta}^*}(\phi) \geqslant q$ for all the $\boldsymbol{\theta}^*$ such that:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmax}}\, \text{SatAgg}(\mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}}) \qquad \text{and} \qquad \text{SatAgg}(\mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}^*}) \geqslant q$$

The objective is to find all $\boldsymbol{\theta}^*$ that optimally satisfy the knowledge base and to measure if they also satisfy ϕ. One can search for such $\boldsymbol{\theta}^*$ by running multiple optimizations with the objective function of Section 3.2.

This approach is somewhat naive. Even if we run the optimization multiple times with multiple parameter initializations (to, hopefully, reach different optima in the search space), the obtained groundings may not be representative of other optimal or close-to-optimal groundings. In Section 4.8, we give an example that shows the limitations of this approach and motivates the next one.

Reasoning Option 2 (proof by refutation) Here, we reason by refutation and search for a counter-example to the logical consequence by introducing an alternative search objective. Normally, according to Definition 6, one tries to verify that[11]:

$$\text{for all } \theta \in \Theta, \text{ if } \mathcal{G}_\theta(\mathcal{K}) \geqslant q \text{ then } \mathcal{G}_\theta(\phi) \geqslant q. \tag{21}$$

Instead, we solve the dual problem:

$$\text{there exists } \theta \in \Theta \text{ such that } \mathcal{G}_\theta(\mathcal{K}) \geqslant q \text{ and } \mathcal{G}_\theta(\phi) < q. \tag{22}$$

If Eq. (22) is true then a counterexample to Eq. (21) has been found and the logical consequence does not hold. If Eq. (22) is false then no counterexample to Eq. (21) has been found and the logical consequence is assumed to hold true. A search for such parameters θ (the counterexample) can be performed by minimizing $\mathcal{G}_\theta(\phi)$ while imposing a constraint that seeks to invalidate results where $\mathcal{G}_\theta(\mathcal{K}) < q$. We therefore define:

$$\text{penalty}(\mathcal{G}_\theta, q) = \begin{cases} c \text{ if } \mathcal{G}_\theta(\mathcal{K}) < q, \\ 0 \text{ otherwise,} \end{cases} \quad \text{where } c > 1.$$

Given $\mathcal{G}^*$ such that[12]:

$$\mathcal{G}^* = \underset{\mathcal{G}_\theta}{\text{argmin}}(\mathcal{G}_\theta(\phi) + \text{penalty}(\mathcal{G}_\theta, q)) \tag{23}$$

- If $\mathcal{G}^*(\mathcal{K}) < q$: Then for all $\mathcal{G}_\theta$, $\mathcal{G}_\theta(\mathcal{K}) < q$ and therefore $(\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta}) \models_q \phi$.
- If $\mathcal{G}^*(\mathcal{K}) \geqslant q$ and $\mathcal{G}^*(\phi) \geqslant q$: Then for all $\mathcal{G}_\theta$ with $\mathcal{G}_\theta(\mathcal{K}) \geqslant q$, we have that $\mathcal{G}_\theta(\phi) \geqslant \mathcal{G}^*(\phi) \geqslant q$ and therefore $(\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta}) \models_q \phi$.
- If $\mathcal{G}^*(\mathcal{K}) \geqslant q$ and $\mathcal{G}^*(\phi) < q$: Then $(\mathcal{K}, \mathcal{G}(\cdot \mid \boldsymbol{\theta}), \boldsymbol{\Theta}) \nvDash_q \phi$.

Clearly, Equation (23) cannot be used as an objective function for gradient-descent due to null derivatives. Therefore, we propose to approximate the penalty function with the soft constraint:

$$\texttt{elu}(\alpha, \beta(q - \mathcal{G}_\theta(\mathcal{K}))) = \begin{cases} \beta(q - \mathcal{G}_\theta(\mathcal{K})) & \text{if } \mathcal{G}_\theta(\mathcal{K}) \leqslant q, \\ \alpha(e^{q - \mathcal{G}_\theta(\mathcal{K})} - 1) & \text{otherwise,} \end{cases}$$

where $\alpha \geqslant 0$ and $\beta \geqslant 0$ are hyper-parameters (see Fig. 6). When $\mathcal{G}_\theta(\mathcal{K}) < q$, the penalty is linear in $q - \mathcal{G}_\theta(\mathcal{K})$ with a slope of β. Setting β high, the gradients for $\mathcal{G}_\theta(\mathcal{K})$ will be high in absolute value if the knowledge-base is not satisfied. When $\mathcal{G}_\theta(\mathcal{K}) > q$, the penalty is a negative exponential that converges to $-\alpha$. Setting α low but non-zero seeks to ensure that the

---

[11] For simplicity, we temporarily define the notation $\mathcal{G}(\mathcal{K}) := \text{SatAgg}_{\phi \in \mathcal{K}}(\mathcal{K}, \mathcal{G})$.

[12] In the objective function, $\mathcal{G}^*$ should satisfy $\mathcal{G}^*(\mathcal{K}) \geqslant q$ before reducing $\mathcal{G}^*(\phi)$ because the penalty c which is greater than 1 is higher than any potential reduction in $\mathcal{G}(\phi)$ which is smaller or equal to 1.
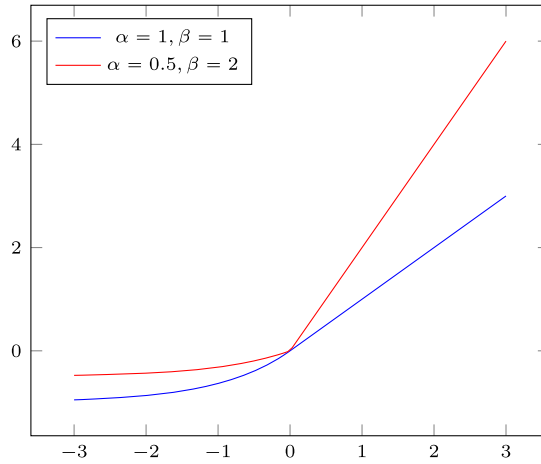
Fig. 6. $\mathtt{elu}(\alpha, \beta x)$ where $\alpha \geqslant 0$ and $\beta \geqslant 0$ are hyper-parameters. The function $\mathtt{elu}(\alpha, \beta(q - \mathcal{G}_\theta(\mathcal{K})))$ with α low and β high is a soft constraint for penalty$(\mathcal{G}_\theta, q)$ suitable for learning.

gradients do not vanish when the penalty should not apply (when the knowledge-base is satisfied). We obtain the following approximate objective function:

$$\mathcal{G}^* = \underset{\mathcal{G}_\theta}{\mathrm{argmin}}(\mathcal{G}_\theta(\phi) + \mathtt{elu}(\alpha, \beta(q - \mathcal{G}_\theta(\mathcal{K})))) \tag{24}$$

Section 4.8 will illustrate the use of reasoning by refutation with an example in comparison with reasoning as *querying after learning*. Of course, other forms of reasoning are possible, not least that adopted in [6], but a direct comparison is outside the scope of this paper and left as future work.

4. The reach of Logic Tensor Networks

The objective of this section is to show how the language of Real Logic can be used to specify a number of tasks that involve learning from data and reasoning. Examples of such tasks are classification, regression, clustering, and link prediction. The solution of a problem specified in Real Logic is obtained by interpreting such a specification in *Logic Tensor Networks*. The LTN library implements Real Logic in Tensorflow 2 [1] and is available from GitHub.[13] Every logical operator is grounded using Tensorflow primitives such that LTN implements directly a Tensorflow graph. Due to Tensorflow built-in optimization, LTN is relatively efficient while providing the expressive power of first-order logic. Details on the implementation of the examples described in this section are reported in Appendix A. The implementation of the examples presented here is also available from the LTN repository on GitHub. Except when stated otherwise, the results reported are the average result over 10 runs using a 95% confidence interval. Every example uses a stable real product configuration to approximate the Real Logic operators and the Adam optimizer [34] with a learning rate of 0.001. Table A.3 in the Appendix gives an overview of the network architectures used to obtain the results reported in this section.

4.1. Binary classification

The simplest machine learning task is binary classification. Suppose that one wants to learn a binary classifier A for a set of points in $[0, 1]^2$. Suppose that a set of positive and negative training examples is given. LTN uses the following language and grounding:

Domains:
  points (denoting the examples).
Variables:
  x_+ for the positive examples.
  x_- for the negative examples.
  x for all examples.
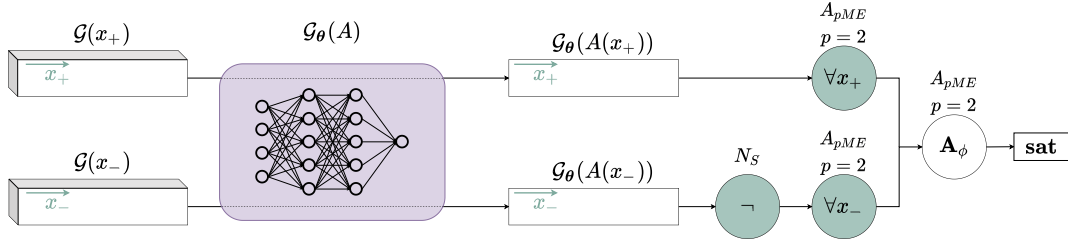 $\mathbf{D}(x) = \mathbf{D}(x_+) = \mathbf{D}(x_-) =$ points.

Fig. 7. Symbolic Tensor Computational Graph for the Binary Classification Example. In the figure, $\mathcal{G}x_+$ and $\mathcal{G}x_-$ are inputs to the network $\mathcal{G}_\theta(A)$ and the dotted lines indicate the propagation of activation from each input through the network, which produces two outputs.

Predicates:

$A(x)$ for the trainable classifier.
$\mathbf{D_{in}}(A) =$ points.

Axioms:

$$\forall x_+ \; A(x_+) \tag{25}$$

$$\forall x_- \; \neg A(x_-) \tag{26}$$

Grounding:

$\mathcal{G}(\text{points}) = [0,1]^2$.
$\mathcal{G}(x) \in [0,1]^{m\times 2}$ ($\mathcal{G}(x)$ is a sequence of m points, that is, m examples).
$\mathcal{G}(x_+) = \langle d \in \mathcal{G}(x) \mid \|d - (0.5, 0.5)\| < 0.09 \rangle$.[14]
$\mathcal{G}(x_-) = \langle d \in \mathcal{G}(x) \mid \|d - (0.5, 0.5)\| \geqslant 0.09 \rangle$.[15]
$\mathcal{G}(A \mid \theta) : x \mapsto \texttt{sigmoid}(\text{MLP}_\theta(x))$, where MLP is a Multilayer Perceptron with a single output neuron, whose parameters θ are to be learned.[16]

Learning:

Let us define $\boldsymbol{D}$ the data set of all examples. The objective function with $\mathcal{K} = \{\forall x_+ \; A(x_+), \forall x_- \; \neg A(x_-)\}$ is given by $\text{argmax}_{\theta \in \Theta} \; \text{SatAgg}_{\phi \in \mathcal{K}} \; \mathcal{G}_{\theta, x \leftarrow \boldsymbol{D}}(\phi)$.[17] In practice, the optimizer uses the following loss function:

$$\boldsymbol{L} = (1 - \underset{\phi \in \mathcal{K}}{\text{SatAgg}} \, \mathcal{G}_{\theta, x \leftarrow \boldsymbol{B}}(\phi))$$

where $\boldsymbol{B}$ is a mini-batch sampled from $\boldsymbol{D}$.[18] The objective and loss functions depend on the following hyper-parameters:

- the choice of fuzzy logic operator semantics used to approximate each connective and quantifier,
- the choice of hyper-parameters underlying the operators, such as the value of the exponent p in any generalized mean,
- the choice of formula aggregator function.

Using the stable product configuration to approximate connectives and quantifiers, and $p = 2$ for every occurrence of A_{pME}, and using for the formula aggregator also A_{pME} with $p = 2$, yields the following satisfaction equation:

$$
\begin{aligned}
\underset{\phi \in \mathcal{K}}{\text{SatAgg}}\, \mathcal{G}_\theta(\phi) = 1 - \frac{1}{2}\Big(1 - \Big(1 - \Big(\frac{1}{|\mathcal{G}(x_+)|}\sum_{v \in \mathcal{G}(x_+)} \big(1 - \texttt{sigmoid}(\text{MLP}_\theta(v))\big)^2\Big)^{\frac{1}{2}\cdot 2}\Big) \\
+ 1 - \Big(1 - \Big(\frac{1}{|\mathcal{G}(x_-)|}\sum_{v \in \mathcal{G}(x_-)} \big(\texttt{sigmoid}(\text{MLP}_\theta(v))\big)^2\Big)^{\frac{1}{2}\cdot 2}\Big)\Big)^{\frac{1}{2}}
\end{aligned}
$$

The computational graph of Fig. 7 shows $\text{SatAgg}_{\phi \in \mathcal{K}} \, \mathcal{G}_\theta(\phi))$ as used with the above loss function.

We are therefore interested in learning the parameters θ of the MLP used to model the binary classifier. We sample 100 data points uniformly from $[0,1]^2$ to populate the data set of positive and negative examples. The data set was split into 50 data points for training and 50 points for testing. The training was carried out for a fixed number of

---

[14] $\mathcal{G}(x_+)$ are, by definition in this example, the training examples with Euclidean distance to the center $(0.5, 0.5)$ smaller than the threshold of 0.09.

[15] $\mathcal{G}(x_-)$ are, by definition, the training examples with Euclidean distance to the center $(0.5, 0.5)$ larger or equal to the threshold of 0.09.

[16] $\texttt{sigmoid}(x) = \frac{1}{1+e^{-x}}$.

[17] The notation $\mathcal{G}_{x \leftarrow \boldsymbol{D}}(\phi(x))$ means that the variable x is grounded with the data $\boldsymbol{D}$ (that is, $\mathcal{G}(x) := \boldsymbol{D}$) when grounding $\phi(x)$.

[18] As usual in ML, while it is possible to compute the loss function and gradients over the entire data set, it is preferred to use mini-batches of the examples.
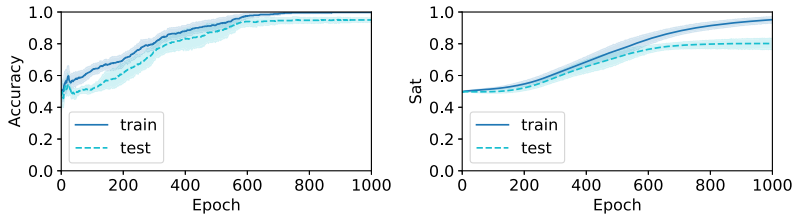
Fig. 8. Binary Classification task (training and test set performance): average accuracy (left) and satisfiability (right). Due to the random initializations, accuracy and satisfiability start on average at 0.5 with performance increasing rapidly after a few epochs.

1000 epochs using backpropagation with the Adam optimizer [34] with a batch size of 64 examples. Fig. 8 shows the classification accuracy and satisfaction level of the LTN on both training and test sets averaged over 10 runs using a 95% confidence interval. The accuracy shown is the ratio of examples correctly classified, with an example deemed as being positive if the classifier outputs a value higher than 0.5.

Notice that a model can reach an accuracy of 100% while satisfaction of the knowledge base is yet not maximized. For example, if the threshold for an example to be deemed as positive is 0.7, all examples may be classified correctly with a confidence score of 0.7. In that case, while the accuracy is already maximized, the satisfaction of $\forall x_+ A(x_+)$ would still be 0.7, and can still improve until the confidence for every sample reaches 1.0.

This first example, although straightforward, illustrates step-by-step the process of using LTN in a simple setting. Notice that, according to the nomenclature of Section 3.3, measuring accuracy amounts to querying the *truth query* (respectively, the *generalization truth query*) $A(x)$ for all the examples of the training set (respectively, test set) and comparing the results with the classification threshold. In Fig. 9, we show the results of such queries $A(x)$ after optimization. Next, we show how the LTN language can be used to solve progressively more complex problems by combining learning and reasoning.

4.2. Multi-class single-label classification

The natural extension of binary classification is a multi-class classification task. We first approach multi-class single-label classification, which assumes that each example is assigned to one and only one label.

For illustration purposes, we use the *Iris flower* data set [20], which consists of classification into three mutually exclusive classes; call these A, B, and C. While one could train three unary predicates $A(x)$, $B(x)$ and $C(x)$, it turns out to be more effective if this problem is modeled by a single binary predicate $P(x, l)$, where l is a variable denoting a multi-class label, in this case, classes A, B or C. This syntax allows one to write statements quantifying over the classes, e.g. $\forall x (\exists l (P(x, l)))$. Since the classes are mutually exclusive, the output layer of the MLP representing $P(x, l)$ will be a `softmax` layer, instead of a `sigmoid` function, to ensure the exclusivity constraint on satisfiability scores.[19] The problem can be specified as follows:

Domains:
> items, denoting the examples from the Iris flower data set.
> labels, denoting the class labels.

Variables:
> x_A, x_B, x_C for the positive examples of classes A, B, C.
> x for all examples.
> $\mathbf{D}(x_A) = \mathbf{D}(x_B) = \mathbf{D}(x_C) = \mathbf{D}(x) = $ items.

Constants:
> l_A, l_B, l_C, the labels of classes A (Iris setosa), B (Iris virginica), C (Iris versicolor), respectively.
> $\mathbf{D}(l_A) = \mathbf{D}(l_B) = \mathbf{D}(l_C) = $ labels.

Predicates:
> $P(x, l)$ denoting the fact that item x is classified as l.
> $\mathbf{D_{in}}(P) = $ items, labels.

Axioms:

$$\forall x_A \; P(x_A, l_A) \tag{27}$$

$$\forall x_B \; P(x_B, l_B) \tag{28}$$

$$\forall x_C \; P(x_C, l_C) \tag{29}$$

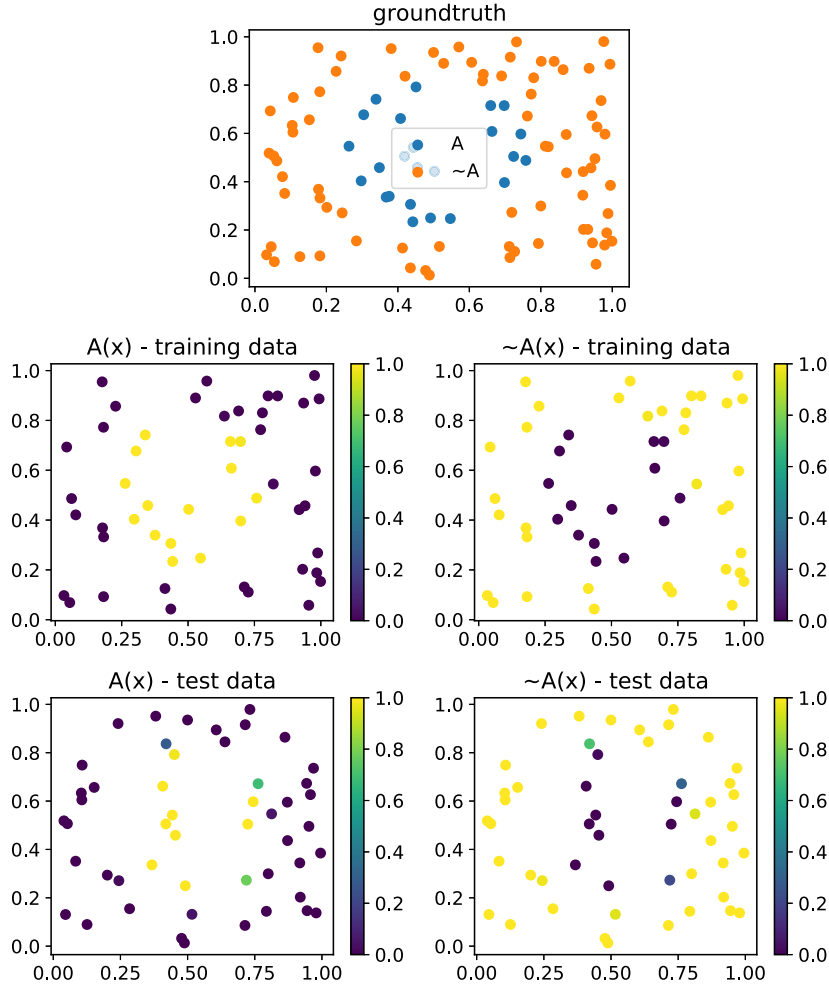[19] $\texttt{softmax}(x) = e^{x_i} / \sum_j e^{x_j}$.

Fig. 9. Binary Classification task (querying the trained predicate $A(x)$): it is interesting to see how $A(x)$ could be appropriately named as denoting the inside of the central region shown in the figure, and therefore $\neg A(x)$ represents the outside of the region. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Notice that rules about exclusiveness such as $\forall x(P(x, l_A) \rightarrow (\neg P(x, l_B) \wedge \neg P(x, l_C)))$ are not included since such constraints are already imposed by the grounding of P below, more specifically the `softmax` function.

Grounding:

$\mathcal{G}(\text{items}) = \mathbb{R}^4$, items are described by 4 features: the length and the width of the sepals and petals, in centimeters.

$\mathcal{G}(\text{labels}) = \mathbb{N}^3$, we use a one-hot encoding to represent classes.

$\mathcal{G}(x_A) \in \mathbb{R}^{m_1 \times 4}$, that is, $\mathcal{G}(x_A)$ is a sequence of m_1 examples of class A.

$\mathcal{G}(x_B) \in \mathbb{R}^{m_2 \times 4}$, $\mathcal{G}(x_B)$ is a sequence of m_2 examples of class B.

$\mathcal{G}(x_C) \in \mathbb{R}^{m_3 \times 4}$, $\mathcal{G}(x_C)$ is a sequence of m_3 examples of class C.

$\mathcal{G}(x) \in \mathbb{R}^{(m_1 + m_2 + m_3) \times 4}$, $\mathcal{G}(x)$ is a sequence of all the examples.

$\mathcal{G}(l_A) = [1, 0, 0]$, $\mathcal{G}(l_B) = [0, 1, 0]$, $\mathcal{G}(l_C) = [0, 0, 1]$.

$\mathcal{G}(P \mid \theta) : x, l \mapsto l^\top \cdot \texttt{softmax}(\text{MLP}_\theta(x))$, where the MLP has three output neurons corresponding to as many classes, and $\cdot$ denotes the dot product as a way of selecting an output for $\mathcal{G}(P \mid \theta)$; multiplying the MLP's output by the one-hot vector $l^\top$ gives the truth degree corresponding to the class denoted by l.

Learning:

The logical operators and connectives are approximated using the stable product configuration with $p = 2$ for A_{pME}. For the formula aggregator, A_{pME} is used also with $p = 2$.

The computational graph of Fig. 10 illustrates how $\text{SatAgg}_{\phi \in \mathcal{K}} \mathcal{G}_\theta(\phi)$ is obtained. If $\boldsymbol{U}$ denotes batches sampled from the data set of all examples, the loss function (to minimize) is:

$$\boldsymbol{L} = 1 - \text{SatAgg}_{\phi \in \mathcal{K}} \mathcal{G}_{\theta, x \leftarrow \boldsymbol{B}}(\phi).$$
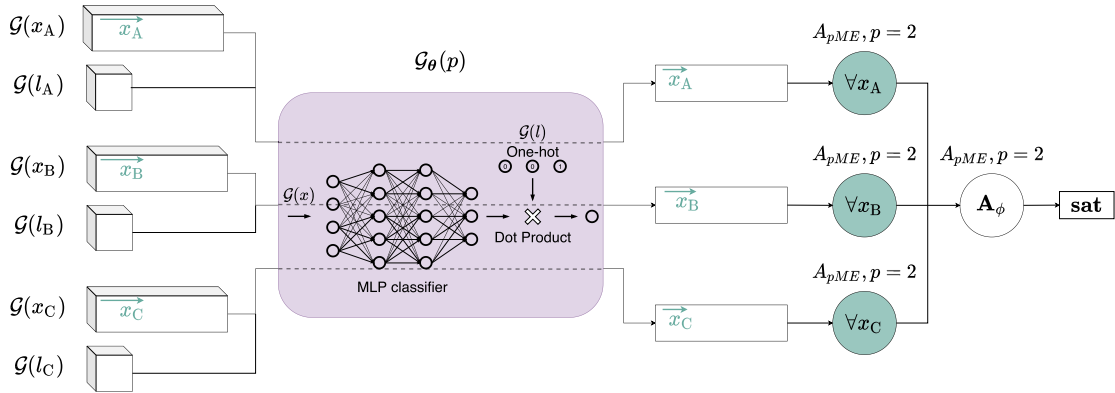
Fig. 10. Symbolic Tensor Computational Graph for the Multi-Class Single-Label Problem. As before, the dotted lines in the figure indicate the propagation of activation from each input through the network, in this case producing three outputs.
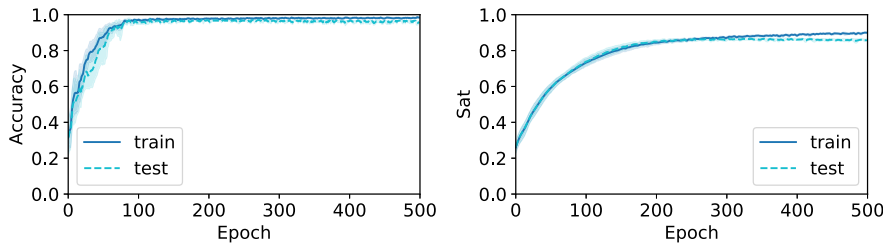

Fig. 11. Multi-Class Single-Label Classification: classification accuracy (left) and satisfaction level (right).

Fig. 11 shows the result of training with the Adam optimizer with batches of 64 examples. Accuracy measures the ratio of examples correctly classified, with example x labeled as $\operatorname{argmax}_l(P(x, l))$.[20] Classification accuracy reaches an average value near 1.0 for both the training and test data after some 100 epochs. Satisfaction levels of the Iris flower predictions continue to increase for the rest of the training (500 epochs) to more than 0.8.

It is worth contrasting the choice of using a binary predicate $(P(x, l))$ in this example with the option of using multiple unary predicates $(l_A(x), l_B(x), l_C(x))$, one for each class. Notice how each predicate is normally associated with an output neuron. In the case of the unary predicates, the networks would be disjoint (or modular), whereas weight-sharing takes place with the use of the binary predicate. Since l is instantiated into l_A, l_B, l_C, in practice $P(x, l)$ becomes $P(x, l_A)$, $P(x, l_B)$, $P(x, l_C)$, which is implemented via three output neurons to which a softmax function applies.

4.3. Multi-class multi-label classification

We now turn to multi-label classification, whereby multiple labels can be assigned to each example. As a first example of the reach of LTNs, we shall see how the previous example can be extended naturally using LTN to account for multiple labels, not always a trivial extension for most ML algorithms. The standard approach to the multi-label problem is to provide explicit negative examples for each class. By contrast, LTN can use background knowledge to relate classes directly to each other, thus becoming a powerful tool in the case of the multi-label problem when typically the labeled data is scarce. We explore the *Leptograpsus crabs* data set [10] consisting of 200 examples of 5 morphological measurements of 50 crabs. The task is to classify the crabs according to their color and sex. There are four labels: blue, orange, male, and female. The color labels are mutually exclusive, and so are the labels for sex. LTN will be used to specify such information logically.

Domains:
   items denoting the examples from the crabs dataset.
   labels denoting the class labels.
Variables:
   x_{blue}, x_{orange}, x_{male}, x_{female} for the positive examples of each class.
   x, used to denote all the examples.
   $\mathbf{D}(x_{\text{blue}}) = \mathbf{D}(x_{\text{orange}}) = \mathbf{D}(x_{\text{male}}) = \mathbf{D}(x_{\text{female}}) = \mathbf{D}(x) = $ items.

---

[20] This is also known as *top-1* accuracy, as proposed in [38]. Cross-entropy results $\sum(t\,log(y))$ could have been reported here as is common with the use of softmax, although it is worth noting that, of course, the loss function used by LTN is different.
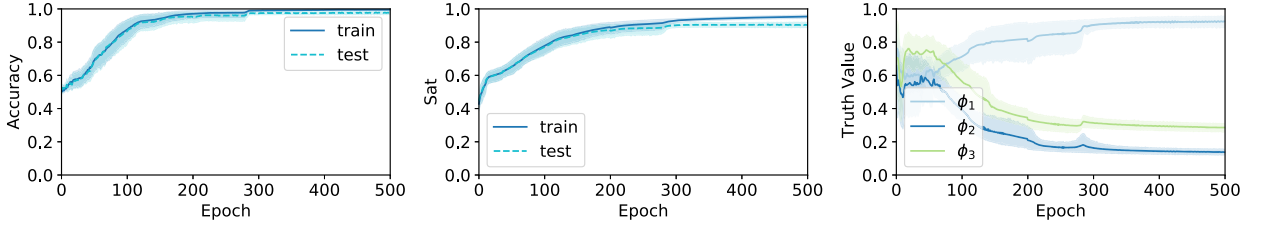
Fig. 12. Multi-Class Multi-Label Classification: classification accuracy (left), satisfiability level (middle), and querying of constraints (right).

Constants:

$l_{\text{blue}}, l_{\text{orange}}, l_{\text{male}}, l_{\text{female}}$ (the labels for each class).
$\mathbf{D}(l_{\text{blue}}) = \mathbf{D}(l_{\text{orange}}) = \mathbf{D}(l_{\text{male}}) = \mathbf{D}(l_{\text{female}}) = \text{labels}$.

Predicates:

$P(x, l)$, denotes the fact that item x is labeled as l.
$\mathbf{D_{in}}(P) = \text{items, labels}$.

Axioms:

$$\forall x_{\text{blue}} \; P(x_{\text{blue}}, l_{\text{blue}}) \tag{30}$$

$$\forall x_{\text{orange}} \; P(x_{\text{orange}}, l_{\text{orange}}) \tag{31}$$

$$\forall x_{\text{male}} \; P(x_{\text{male}}, l_{\text{male}}) \tag{32}$$

$$\forall x_{\text{female}} \; P(x_{\text{female}}, l_{\text{female}}) \tag{33}$$

$$\forall x \; \neg(P(x, l_{\text{blue}}) \wedge P(x, l_{\text{orange}})) \tag{34}$$

$$\forall x \; \neg(P(x, l_{\text{male}}) \wedge P(x, l_{\text{female}})) \tag{35}$$

Notice how logical rules (34) and (35) above represent the mutual exclusion of the labels on color and sex, respectively. As a result, negative examples are not used explicitly in this specification.

Grounding:

$\mathcal{G}(\text{items}) = \mathbb{R}^5$; the examples from the data set are described using 5 features.
$\mathcal{G}(\text{labels}) = \mathbb{N}^4$; one-hot vectors are used to represent class labels.[21]

$\mathcal{G}(x_{\text{blue}}) \in \mathbb{R}^{m_1 \times 5}$, $\mathcal{G}(x_{\text{orange}}) \in \mathbb{R}^{m_2 \times 5}$, $\mathcal{G}(x_{\text{male}}) \in \mathbb{R}^{m_3 \times 5}$, $\mathcal{G}(x_{\text{female}}) \in \mathbb{R}^{m_4 \times 5}$. These sequences are not mutually-exclusive, one example can for instance be in both x_{blue} and x_{male}.
$\mathcal{G}(l_{\text{blue}}) = [1, 0, 0, 0]$, $\mathcal{G}(l_{\text{orange}}) = [0, 1, 0, 0]$, $\mathcal{G}(l_{\text{male}}) = [0, 0, 1, 0]$, $\mathcal{G}(l_{\text{female}}) = [0, 0, 0, 1]$.
$\mathcal{G}(P \mid \theta) : x, l \mapsto l^\top \cdot \texttt{sigmoid}(\text{MLP}_\theta(x))$, with the MLP having four output neurons corresponding to as many classes. As before, $\cdot$ denotes the dot product which selects a single output. By contrast with the previous example, notice the use of a `sigmoid` function instead of a `softmax` function.

Learning:

As before, the fuzzy logic operators and connectives are approximated using the stable product configuration with $p = 2$ for A_{pME}, and for the formula aggregator, A_{pME} is also used with $p = 2$.
Fig. 12 shows the result of the Adam optimizer using backpropagation trained with batches of 64 examples. This time, the accuracy is defined as $1 - \text{HL}$, where HL is the average Hamming loss, i.e. the fraction of labels predicted incorrectly, with a classification threshold of 0.5 (given an example u, if the model outputs a value greater than 0.5 for class C then u is deemed as belonging to class C). The rightmost graph in Fig. 12 illustrates how LTN learns the constraint that a crab cannot have both blue and orange color, which is discussed in more detail in what follows.

Querying:

To illustrate the learning of constraints by LTN, we have queried three formulas that were not explicitly part of the knowledge-base, over time during learning:

$$\phi_1 : \quad \forall x \, (P(x, l_{\text{blue}}) \rightarrow \neg P(x, l_{\text{orange}})) \tag{36}$$

$$\phi_2 : \quad \forall x \, (P(x, l_{\text{blue}}) \rightarrow P(x, l_{\text{orange}})) \tag{37}$$

$$\phi_3 : \quad \forall x \, (P(x, l_{\text{blue}}) \rightarrow P(x, l_{\text{male}})) \tag{38}$$

---

[21] There are two possible approaches here: either each item is labeled with one multi-hot encoding or each item is labeled with several one-hot encodings. The latter approach was used in this example.

For querying, we use $p = 5$ when approximating the universal quantifiers with A_{pME}. A higher p denotes a stricter universal quantification with a stronger focus on outliers (see Section 2.4).[22] We should expect ϕ_1 to hold true (every blue crab cannot be orange and vice-versa,[23] and we should expect ϕ_2 (every blue crab is also orange) and ϕ_3 (every blue crab is male) to be false. The results are reported in the rightmost plot of Fig. 12. Prior to training, the truth-values of ϕ_1 to ϕ_3 are non-informative. During training one can see, with the maximization of the satisfaction of the knowledge-base, a trend towards the satisfaction of ϕ_1, and an opposite trend of ϕ_2 and ϕ_3 towards *false*.

4.4. Semi-supervised pattern recognition

Let us now explore two, more elaborate, classification tasks, which showcase the benefit of using logical reasoning alongside machine learning. With these two examples, we also aim to provide a more direct comparison with a related neurosymbolic system DeepProbLog [40]. The benchmark examples below were introduced in the DeepProbLog paper [40].

Single Digits Addition: Consider the predicate addition(X, Y, N), where X and Y are images of digits (the MNIST data set will be used), and N is a natural number corresponding to the sum of these digits. This predicate should return an estimate of the validity of the addition. For instance, addition(▨, ▨, 11) is a valid addition; addition(▨, ▨, 5) is not.

Multi Digits Addition: The experiment is extended to numbers with more than one digit. Consider the predicate addition([X_1, X_2], [Y_1, Y_2], N). [X_1, X_2] and [Y_1, Y_2] are lists of images of digits, representing two multi-digit numbers; N is a natural number corresponding to the sum of the two multi-digit numbers. For instance, addition([▨, ▨], [▨, ▨], 130) is a valid addition; addition([▨, ▨], [▨, ▨], 26) is not.

A natural neurosymbolic approach is to seek to learn a single-digit classifier and benefit from knowledge readily available about the properties of addition in this case. For instance, suppose that a predicate digit(x, d) gives the likelihood of an image x being of digit d. A definition for addition(▨, ▨, 11) in LTN is:

$$\exists d_1, d_2 : d_1 + d_2 = 11 \ (\text{digit}(▨, d_1) \land \text{digit}(▨, d_2))$$

In [40], the above task is made more complicated by not providing labels for the single-digit images during training. Instead, training takes place on pairs of images with labels made available for the result only, that is, the sum of the individual labels. The single-digit classifier is not explicitly trained by itself; its output is a piece of latent information that is used by the logic. However, this does not pose a problem for end-to-end neurosymbolic systems such as LTN or DeepProbLog for which the gradients can propagate through the logical structures.

We start by illustrating a LTN theory that can be used to learn the predicate digit. The specification of the theory below is for the single digit addition example, although it can be extended easily to the multiple digits case.

Domains:
    images, denoting the MNIST digit images,
    results, denoting the integers that label the results of the additions,
    digits, denoting the digits from 0 to 9.

Variables:
    x, y, ranging over the MNIST images in the data,
    n for the labels, i.e. the result of each addition,
    d_1, d_2 ranging over digits.
    $\mathbf{D}(x) = \mathbf{D}(y) = $ images,
    $\mathbf{D}(n) = $ results,
    $\mathbf{D}(d_1) = \mathbf{D}(d_2) = $ digits.

Predicates:
    digit(x, d) for the single digit classifier, where d is a term denoting a digit constant or a digit variable. The classifier should return the probability of an image x being of digit d.
    $\mathbf{D_{in}}(\text{digit}) = $ images, digits.

Axioms:
    Single Digit Addition:

$$\forall \text{Diag}(x, y, n)$$
$$(\exists d_1, d_2 : d_1 + d_2 = n \tag{39}$$

---

[22] Training should usually not focus on outliers, as optimizers would struggle to generalize and tend to get stuck in local minima. However, when querying ϕ_1, ϕ_2, ϕ_3, we wish to be more careful about the interpretation of our statement. See also 3.1.3.

[23] Notice how, strictly speaking, other colors remain possible since the prior knowledge did not specify the bi-conditional: $\forall x \ (P(x, l_{\text{blue}}) \leftrightarrow \neg P(x, l_{\text{orange}}))$.
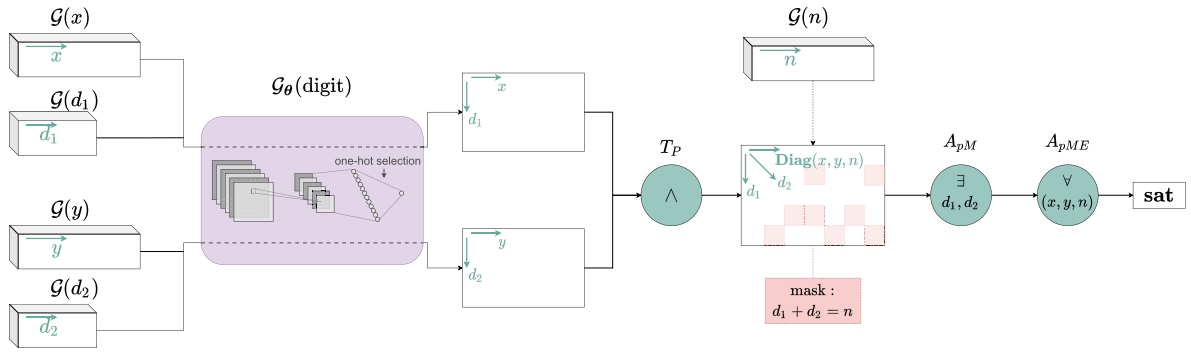
Fig. 13. Symbolic Tensor Computational Graph for the Single Digit Addition task. Notice that the figure does not depict accurate dimensions for the tensors; $\mathcal{G}(x)$ and $\mathcal{G}(y)$ are in fact 4D tensors of dimensions $m \times 28 \times 28 \times 1$. Computing results with the variables d_1 or d_2 corresponds to the addition of a further axes of dimension 10.

$$(\text{digit}(x, d_1) \wedge \text{digit}(y, d_2)))$$

Multiple Digit Addition:

$$\forall \text{Diag}(x_1, x_2, y_1, y_2, n)$$
$$(\exists d_1, d_2, d_3, d_4 : 10d_1 + d_2 + 10d_3 + d_4 = n \tag{40}$$
$$(\text{digit}(x_1, d_1) \wedge \text{digit}(x_2, d_2) \wedge \text{digit}(y_1, d_3) \wedge \text{digit}(y_2, d_4)))$$

Notice the use of Diag: when grounding x, y, n with three sequences of values, the i-th examples of each variable are matching. That is, $(\mathcal{G}(x)_i, \mathcal{G}(y)_i, \mathcal{G}(n)_i)$ is a tuple from our dataset of valid additions. Using the diagonal quantification, LTN aggregates pairs of images and their corresponding result, rather than any combination of images and results.

Notice also the guarded quantification: by quantifying only on the latent "digit labels" (i.e. d_1, d_2, ...) that can add up to the result label (n, given in the dataset), we incorporate symbolic information into the system. For example, in (39), if $n = 3$, the only valid tuples (d_1, d_2) are $(0, 3), (3, 0), (1, 2), (2, 1)$. Gradients will only backpropagate to these values.

Grounding:

$\mathcal{G}(\text{images}) = [0, 1]^{28 \times 28 \times 1}$. The MNIST data set has images of 28 by 28 pixels. The images are grayscale and have just one channel. The RGB pixel values from 0 to 255 of the MNIST data set are converted to the range $[0, 1]$.

$\mathcal{G}(\text{results}) = \mathbb{N}$.

$\mathcal{G}(\text{digits}) = \{0, 1, \ldots, 9\}$.

$\mathcal{G}(x) \in [0, 1]^{m \times 28 \times 28 \times 1}$, $\mathcal{G}(y) \in [0, 1]^{m \times 28 \times 28 \times 1}$, $\mathcal{G}(n) \in \mathbb{N}^m$.[24]

$\mathcal{G}(d_1) = \mathcal{G}(d_2) = \langle 0, 1, \ldots, 9 \rangle$.

$\mathcal{G}(\text{digit} \mid \theta) : x, d \mapsto \texttt{onehot}(d)^\top \cdot \texttt{softmax}(\texttt{CNN}_\theta(x))$, where $\texttt{CNN}$ is a Convolutional Neural Network with 10 output neurons for each class. Notice that, in contrast with the previous examples, d is an integer label; $\texttt{onehot}(d)$ converts it into a one-hot label.

Learning:

The computational graph of Fig. 13 shows the objective function for the satisfiability of the knowledge base. A stable product configuration is used with hyper-parameter $p = 2$ of the operator A_{pME} for universal quantification ($\forall$). Let $p_\exists$ denote the exponent hyper-parameter used in the generalized mean A_{pM} for existential quantification ($\exists$). Three scenarios are investigated and compared in the Multiple Digit experiment (Fig. 15):

1. $p_\exists = 1$ throughout the entire experiment,
2. $p_\exists = 2$ throughout the entire experiment, or
3. $p_\exists$ follows a schedule, changing from $p = 1$ to $p = 6$ gradually with the number of training epochs.

In the Single Digit experiment, only the last scenario above (schedule) is investigated (Fig. 14).

We train to maximize satisfiability by using batches of 32 examples of image pairs, labeled by the result of their addition. As done in [40], the experimental results vary the number of examples in the training set to emphasize the generalization abilities of a neurosymbolic approach. Accuracy is measured by predicting the digit values using the predicate digit and reporting the ratio of examples for which the addition is correct. A comparison is made with the same baseline method used in [40]: given a pair of MNIST images, a non-pre-trained CNN outputs embeddings for each image (Siamese neural network). The embeddings are provided as input to dense layers that classify the addition into one of the 19 (respectively, 199) possible results of the Single Digit Addition (respectively, Multiple Digit Addition) experiments. The baseline is trained using a cross-entropy loss between the labels and the predictions. As expected,

---

[24] Notice the use of the same number m of examples for each of these variables as they are supposed to match one-to-one due to the use of Diag.
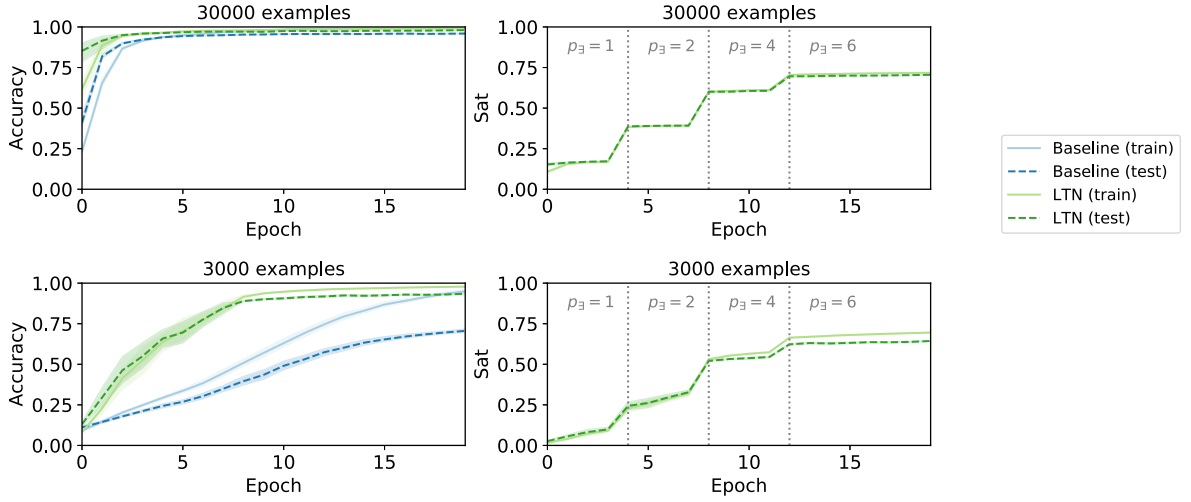
Fig. 14. Single Digit Addition Task: accuracy and satisfiability results (top) and results in the presence of fewer examples (bottom) in comparison with standard Deep Learning using a CNN (blue lines).

Fig. 15. Multiple Digit Addition Task: accuracy and satisfiability results (top) and results in the presence of fewer examples (bottom) in comparison with standard Deep Learning using a CNN (blue lines).
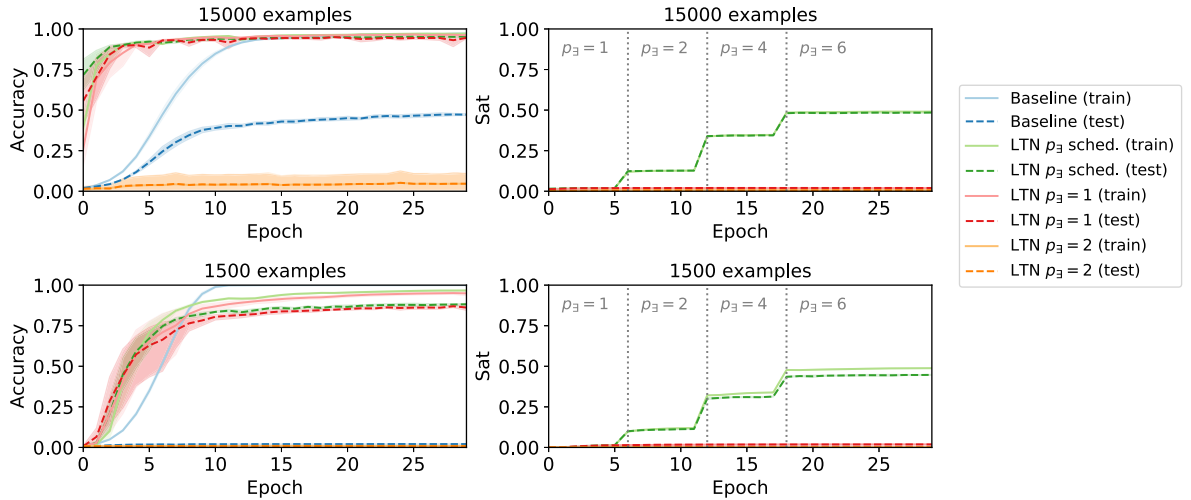
such a standard deep learning approach struggles with the task without the provision of symbolic meaning about intermediate parts of the problem.

Experimentally, we find that the optimizer for the neurosymbolic system gets stuck in a local optimum at the initialization in about 1 out of 5 runs. We, therefore, present the results on an average of the 10 best outcomes out of 15 runs of each algorithm (that is, for the baseline as well). The examples of digit pairs selected from the full MNIST data set are randomized at each run.

Fig. 15 shows that the use of $p_\exists = 2$ from the start produces poor results. A higher value for $p_\exists$ in A_{pM} weighs up the instances with a higher truth-value (see also Appendix C for a discussion). Starting already with a high value for $p_\exists$, the classes with a higher initial truth-value for a given example will have higher gradients and be prioritized for training, which does not make practical sense when randomly initializing the predicates. Increasing $p_\exists$ by following a schedule is the most promising approach. In this particular example, $p_\exists = 1$ is also shown to be adequate purely from a learning perspective. However, $p_\exists = 1$ implements a simple average which does not account for the meaning of $\exists$ well; the resulting satisfaction value is not meaningful within a reasoning perspective.

Table 1 shows that the training and test times of LTN are of the same order of magnitude as those of the CNN baselines. Table 2 shows that LTN reaches similar accuracy as that reported by DeepProbLog.

Table 1
The computation time of training and test steps on the single and multiple digit addition tasks, measured on a computer with a single Nvidia Tesla V100 GPU and averaged over 1000 steps. Each step operates on a batch of 32 examples. The computational efficiency of the LTN and the CNN baseline systems are of the same order of magnitude.

Model	(Single Digits)		(Multi Digits)	
	Train	Test	Train	Test
baseline	2.72 ± 0.23 ms	1.45 ± 0.21 ms	3.87 ± 0.24 ms	2.10 ± 0.30 ms
LTN	5.36 ± 0.25 ms	3.44 ± 0.39 ms	8.51 ± 0.72 ms	5.72 ± 0.57 ms

Table 2
Accuracy (in %) on the test set: comparison of the final results obtained with LTN and those reported with DeepProbLog [40]. Although it is difficult to compare directly the results over time (the frameworks are implemented in different libraries), while achieving similar computational efficiency as the CNN baseline, LTN also reaches similar accuracy as that reported by DeepProbLog.

Model	Number of training examples			
	(Single Digits)		(Multi Digits)	
	30 000	3 000	15 000	1 500
baseline	95.95 ± 0.27	70.59 ± 1.45	47.19 ± 0.69	2.07 ± 0.12
LTN	98.04 ± 0.13	93.49 ± 0.28	95.37 ± 0.29	88.21 ± 0.63
DeepProbLog	97.20 ± 0.45	92.18 ± 1.57	95.16 ± 1.70	87.21 ± 1.92

4.5. Regression

Another important problem in Machine Learning is regression where a relationship is estimated between one independent variable X and a continuous dependent variable Y. The essence of regression is, therefore, to approximate a function $f(x) = y$ by a function f^*, given examples (x_i, y_i) such that $f(x_i) = y_i$. In LTN one can model a regression task by defining f^* as a learnable function whose parameter values are constrained by data. Additionally, a regression task requires a notion of equality. We, therefore, define the predicate eq as a smooth version of the symbol $=$ to turn the constraint $f(x_i) = y_i$ into a smooth optimization problem.

In this example, we explore regression using a problem from a real estate data set[25] with 414 examples, each described in terms of 6 real-numbered features: the transaction date (converted to a float), the age of the house, the distance to the nearest station, the number of convenience stores in the vicinity, and the latitude and longitude coordinates. The model has to predict the house price per unit area.

Domains:
> samples, denoting the houses and their features.
> prices, denoting the house prices.

Variables:
> x for the samples.
> y for the prices.
> $\mathbf{D}(x) =$ samples.
> $\mathbf{D}(y) =$ prices.

Functions:
> $f^*(x)$, the regression function to be learned.
> $\mathbf{D_{in}}(f^*) =$ samples, $\mathbf{D_{out}}(f^*) =$ prices.

Predicates:
> $eq(y_1, y_2)$, a smooth equality predicate that measures how similar y_1 and y_2 are.
> $\mathbf{D_{in}}(eq) =$ prices, prices.

Axioms:

$$\forall \text{Diag}(x, y) \, eq(f^*(x), y) \tag{41}$$

Notice again the use of Diag: when grounding x and y onto sequences of values, this is done by obeying a one-to-one correspondence between the sequences. In other words, we aggregate pairs of corresponding samples and prices, instead of any combination thereof.
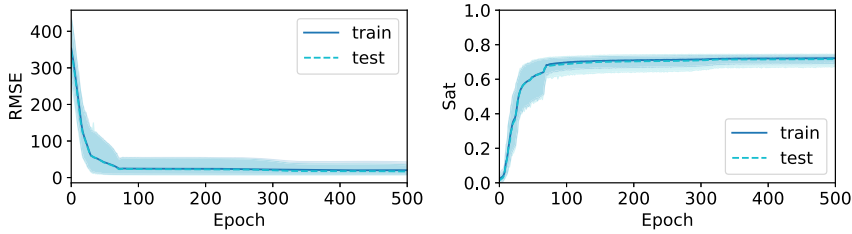

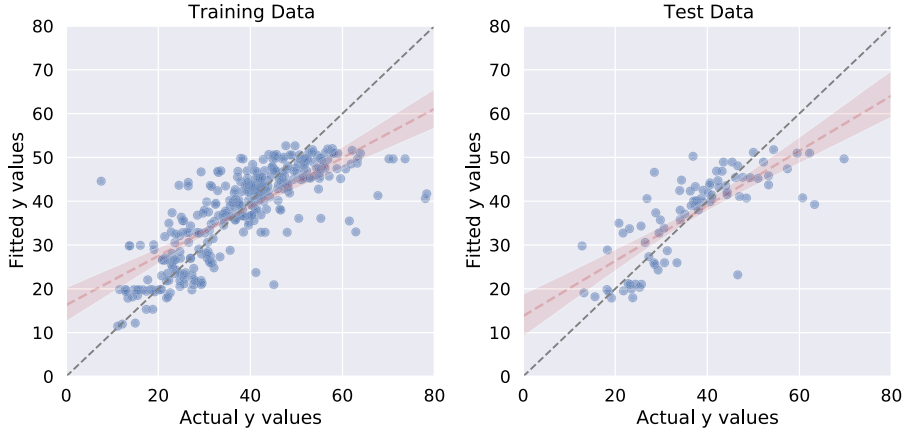**Fig. 16.** Regression task: RMSE and satisfaction level over time.

Fig. 17. Visualization of LTN solving a regression problem.

Grounding:

$\mathcal{G}(\text{samples}) = \mathbb{R}^6$.

$\mathcal{G}(\text{prices}) = \mathbb{R}$.

$\mathcal{G}(x) \in \mathbb{R}^m \times 6$, $\mathcal{G}(y) \in \mathbb{R}^m \times 1$. Notice that this specification refers to the same number m of examples for x and y due to the above one-to-one correspondence obtained with the use of Diag.

$\mathcal{G}(\text{eq}(\mathbf{u}, \mathbf{v})) = \exp\left(-\alpha\sqrt{\sum_j (u_j - v_j)^2}\right)$, where the hyper-parameter α is a real number that scales how strict the smooth equality is.[26] In our experiments, we use $\alpha = 0.05$.

$\mathcal{G}(f^*(x) \mid \theta) = \text{MLP}_\theta(x)$, where MLP_θ is a multilayer perceptron which ends in one neuron corresponding to a price prediction, with a linear output layer (no activation function).

Learning:

The theory is constrained by the parameters of the model of f^*. LTN is used to estimate such parameters by maximizing the satisfaction of the knowledge-base, in the usual way. Approximating $\forall$ using A_{pME} with $p = 2$, as before, we randomly split the data set into 330 examples for training and 84 examples for testing. Fig. 16 shows the satisfaction level over 500 epochs. We also plot the Root Mean Squared Error (RMSE) between the predicted prices and the labels (i.e. actual prices, also known as target values). We visualize in Fig. 17 the strong correlation between actual and predicted prices at the end of one of the runs.

4.6. Unsupervised learning (clustering)

In unsupervised learning, labels are either not available or are not used for learning. Clustering is a form of unsupervised learning whereby, without labels, the data is characterized by constraints alone. LTN can formulate such constraints, such as:

- clusters should be disjoint,
- every example should be assigned to a cluster,
- a cluster should not be empty,
- if the points are near, they should belong to the same cluster,

---

[26] Intuitively, the smooth equality is $\exp(-\alpha\, d(\mathbf{u}, \mathbf{v}))$, where $d(\mathbf{u}, \mathbf{v})$ is the Euclidean distance between $\mathbf{u}$ and $\mathbf{v}$. It produces a 1 if the distance is zero; as the distance increases, the result decreases exponentially towards 0. In case an exponential decrease is undesirable, one can adopt the following alternative equation: $\text{eq}(\mathbf{u}, \mathbf{v}) = \frac{1}{1 + \alpha d(\mathbf{u}, \mathbf{v})}$.

• if the points are far, they should belong to different clusters, etc.

Domains:
 points, denoting the data to cluster.
 points_pairs, denoting pairs of examples.
 clusters, denoting the cluster.
Variables:
 x, y for all points.
 $\mathbf{D}(x) = \mathbf{D}(y) =$ points.
 $\mathbf{D}(c) =$ clusters.
Predicates:
 $C(x, c)$, the truth degree of a given point belonging in a given cluster.
 $\mathbf{D_{in}}(C) =$ points, clusters.
Axioms:

$$\forall x \, \exists c \; C(x, c) \tag{42}$$

$$\forall c \, \exists x \; C(x, c) \tag{43}$$

$$\forall (c, x, y : |x - y| < \text{th}_{\text{close}}) \; (C(x, c) \leftrightarrow C(y, c)) \tag{44}$$

$$\forall (c, x, y : |x - y| > \text{th}_{\text{distant}}) \; \neg(C(x, c) \wedge C(y, c)) \tag{45}$$

Notice the use of guarded quantifiers: all the pairs of points with Euclidean distance lower (resp. higher) than a value th_{close} (resp. $\text{th}_{\text{distant}}$) should belong in the same cluster (resp. should not). th_{close} and $\text{th}_{\text{distant}}$ are arbitrary threshold values that define some of the closest and most distant pairs of points. In our example, they are set to, respectively, 0.2 and 1.0.

As done in the example of Section 4.2, the clustering predicate has mutually exclusive satisfiability scores for each cluster using a softmax layer. Therefore, there is no explicit constraint about clusters being disjoint.

Grounding:
 $\mathcal{G}(\text{points}) = [-1, 1]^2$.
 $\mathcal{G}(\text{clusters}) = \mathbb{N}^4$, we use one-hot vectors to represent a choice of 4 clusters.
 $\mathcal{G}(x) \in [-1, 1]^{m \times 2}$, that is, x is a sequence of m points. $\mathcal{G}(y) = \mathcal{G}(x)$.
 $\text{th}_{\text{close}} = 0.2$, $\text{th}_{\text{distant}} = 1.0$.
 $\mathcal{G}(c) = \langle [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1] \rangle$.
 $\mathcal{G}(C \mid \theta) : x, c \mapsto c^\top \cdot \text{softmax}(\text{MLP}_\theta(x))$, where MLP has 4 output neurons corresponding to the 4 clusters.
Learning:
 We use the stable real product configuration to approximate the logical operators. For $\forall$, we use A_{pME} with $p = 4$. For $\exists$, we use A_{pM} with $p = 1$ during the first 100 epochs, and $p = 6$ thereafter, as a simplified version of the schedule used in Section 4.4. The formula aggregator is approximated by A_{pME} with $p = 2$. The model is trained for a total of 1000 epochs using the Adam optimizer, which is sufficient for LTN to solve the clustering problem shown in Fig. 18. Ground-truth data for this task was generated artificially by creating 4 centers, and generating 50 random samples from a multivariate Gaussian distribution around each center. The trained LTN achieves a satisfaction level of the clustering constraints of 0.857.

4.7. Learning embeddings with LTN

A classic example of Statistical Relational Learning is the smokers-friends-cancer example introduced in [54]. Below, we show how this example can be formalized in LTN using semi-supervised embedding learning.

There are 14 people divided into two groups $\{a, b, \ldots, h\}$ and $\{i, j, \ldots, n\}$. Within each group, there is complete knowledge about smoking habits. In the first group, there is complete knowledge about who has and who does not have cancer. Knowledge about the friendship relation is complete within each group only if symmetry is assumed, that is, $\forall x, y \; (friends(x, y) \rightarrow friends(y, x))$. Otherwise, knowledge about friendship is incomplete in that it may be known that e.g. a is a friend of b, and it may be not known whether b is a friend of a. Finally, there is general knowledge about smoking, friendship, and cancer, namely that smoking causes cancer, friendship is normally symmetric and anti-reflexive, everyone has a friend, and smoking propagates (actively or passively) among friends. All this knowledge is represented in the axioms further below.

Domains:
 people, to denote the individuals.
Constants:
 $a, b, \ldots, h, i, j, \ldots, n$, the 14 individuals. Our goal is to learn an adequate embedding for each constant.
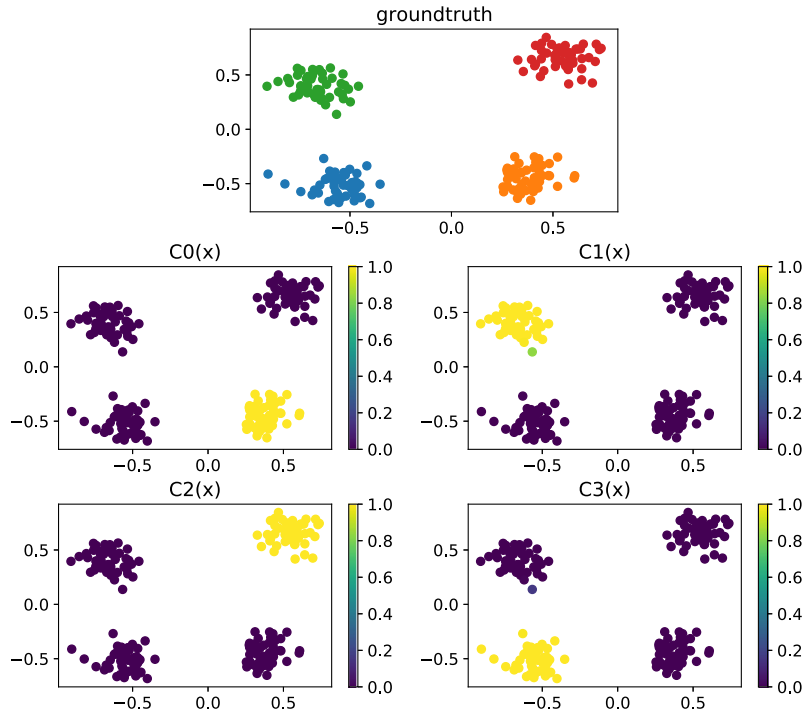 $\mathbf{D}(a) = \mathbf{D}(b) = \cdots = \mathbf{D}(n) =$ people.

Fig. 18. LTN solving a clustering problem by constraint optimization: ground-truth (top) and querying of each cluster $C0$, $C1$, $C2$ and $C3$, in turn.

Variables:

x, y ranging over the individuals.

$\mathbf{D}(x) = \mathbf{D}(y) = $ people.

Predicates:

$S(x)$ for *smokes*, $F(x, y)$ for *friends*, $C(x)$ for *cancer*.

$\mathbf{D}(S) = \mathbf{D}(C) = $ people. $\mathbf{D}(F) = $ people, people.

Axioms:

Let $\mathcal{X}_1 = \{a, b, \ldots, h\}$ and $\mathcal{X}_2 = \{i, j, \ldots, n\}$ be the two groups of individuals.

Let $\mathcal{S} = \{a, e, f, g, j, n\}$ be the smokers; knowledge is complete in both groups.

Let $\mathcal{C} = \{a, e\}$ be the individuals with cancer; knowledge is complete in $\mathcal{X}_1$ only.

Let $\mathcal{F} = \{(a, b), (a, e), (a, f), (a, g), (b, c), (c, d), (e, f), (g, h), (i, j), (j, m), (k, l), (m, n)\}$ be the set of friendship relations; knowledge is complete if assuming symmetry.

These facts are illustrated in Fig. 20a.

We have the following axioms:

$$F(u, v) \qquad \text{for } (u, v) \in \mathcal{F} \tag{46}$$

$$\neg F(u, v) \qquad \text{for } (u, v) \notin \mathcal{F}, \ u > v \tag{47}$$

$$S(u) \qquad \text{for } u \in \mathcal{S} \tag{48}$$

$$\neg S(u) \qquad \text{for } u \in (\mathcal{X}_1 \cup \mathcal{X}_2) \setminus \mathcal{S} \tag{49}$$

$$C(u) \qquad \text{for } u \in \mathcal{C} \tag{50}$$

$$\neg C(u) \qquad \text{for } u \in \mathcal{X}_1 \setminus \mathcal{C} \tag{51}$$

$$\forall x \, \neg F(x, x) \tag{52}$$

$$\forall x, y \, (F(x, y) \rightarrow F(y, x)) \tag{53}$$

$$\forall x \exists y \, F(x, y) \tag{54}$$

$$\forall x, y \, ((F(x, y) \land S(x)) \rightarrow S(y)) \tag{55}$$

$$\forall x \, (S(x) \rightarrow C(x)) \tag{56}$$

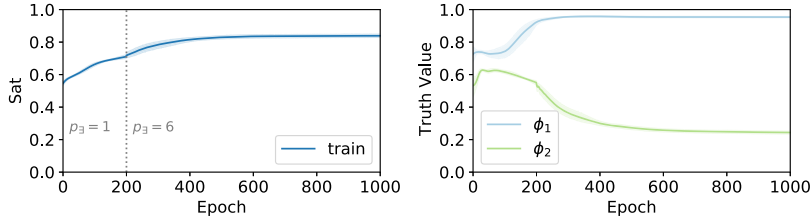$$\forall x \, (\neg C(x) \rightarrow \neg S(x)) \tag{57}$$

Fig. 19. Smoker-Friends-Cancer example: satisfiability levels during training (left) and truth-values of queries ϕ_1 and ϕ_2 over time (right).

Notice that the knowledge base is not satisfiable in the strict logical sense of the word. For instance, f is said to smoke but not to have cancer, which is inconsistent with the rule $\forall x \, (S(x) \to C(x))$. Hence, it is important to adopt a fuzzy approach as done with MLN or a many-valued fuzzy logic interpretation as done with LTN.

Grounding:

$\mathcal{G}(\text{people}) = \mathbb{R}^5$. The model is expected to learn embeddings in $\mathbb{R}^5$.

$\mathcal{G}(a \,|\, \theta) = \mathbf{v}_\theta(a), \ldots, \mathcal{G}(n \,|\, \theta) = \mathbf{v}_\theta(n)$. Every individual is associated with a vector of 5 real numbers. The embedding is initialized randomly uniformly.

$\mathcal{G}(x \,|\, \theta) = \mathcal{G}(y \,|\, \theta) = \langle \mathbf{v}_\theta(a), \ldots, \mathbf{v}_\theta(n) \rangle$.

$\mathcal{G}(S \,|\, \theta) : x \mapsto \texttt{sigmoid}(\texttt{MLP\_S}_\theta(x))$, where $\texttt{MLP\_S}_\theta$ has 1 output neuron.

$\mathcal{G}(F \,|\, \theta) : x, y \mapsto \texttt{sigmoid}(\texttt{MLP\_F}_\theta(x, y))$, where $\texttt{MLP\_F}_\theta$ has 1 output neuron.

$\mathcal{G}(C \,|\, \theta) : x \mapsto \texttt{sigmoid}(\texttt{MLP\_C}_\theta(x))$, where $\texttt{MLP\_C}_\theta$ has 1 output neuron.

The `MLP` models for S, F, C are kept simple, so that most of the learning is focused on the embedding.

Learning:

We use the stable real product configuration to approximate the operators. For $\forall$, we use A_{pME} with $p = 2$ for all the rules, except for rules (52) and (53), where we use $p = 6$. The intuition behind this choice of p is that no outliers are to be accepted for the friendship relation since it is expected to be symmetric and anti-reflexive, but outliers are accepted for the other rules. For $\exists$, we use A_{pM} with $p = 1$ during the first 200 epochs of training, and $p = 6$ thereafter, with the same motivation as that of the schedule used in Section 4.4. The formula aggregator is approximated by A_{pME} with $p = 2$.

Fig. 19 shows the satisfiability over 1000 epochs of training. At the end of one of these runs, we query $S(x)$, $F(x, y)$, $C(x)$ for each individual; the results are shown in Fig. 20b. We also plot the principal components of the learned embeddings [50] in Fig. 21. The friendship relations are learned as expected: (56) "smoking implies cancer" is inferred for group 2 even though such information was not present in the knowledge base. For group 1, the given facts for smoking and cancer for the individuals f and g are slightly altered, as these were inconsistent with the rules. (the rule for smoking propagating via friendship (55) is incompatible with many of the given facts). Increasing the satisfaction of this rule would require decreasing the overall satisfaction of the knowledge base, which explains why it is partly ignored by LTN during training. Finally, it is interesting to note that the principal components for the learned embeddings seem to be linearly separable for the smoking and cancer classifiers (cf. Fig. 21, top right and bottom right plots).

Querying:

To illustrate querying in LTN, we query over time two formulas that are not present in the knowledge-base:

$$\phi_1: \qquad \forall p : C(p) \to S(p) \tag{58}$$

$$\phi_2: \qquad \forall p, q : (C(p) \lor C(q)) \to F(p, q) \tag{59}$$

We use $p = 5$ when approximating $\forall$ since the impact of an outlier at querying time should be seen as more important than at learning time. It can be seen that as the grounding approaches satisfiability of the knowledge-base, ϕ_1 approaches *true*, whereas ϕ_2 approaches *false* (cf. Fig. 20a).

4.8. Reasoning in LTN

The essence of reasoning is to find out if a closed formula ϕ is the logical consequence of a knowledge-base $(\mathcal{K}, \mathcal{G}_\theta, \Theta)$. Section 3.4 introduced two approaches to this problem in LTN:

- By simply *querying after learning*[27] one seeks to verify if for the grounded theories that maximally satisfy $\mathcal{K}$, the grounding of ϕ gives a truth-value greater than a threshold q. This often requires checking an infinite number of groundings. Instead, the user approximates the search for these grounded theories by running the optimization a fixed number of times only.

---

[27] Here, learning refers to Section 3.2, which is optimizing using the satisfaction of the knowledge base as an objective.
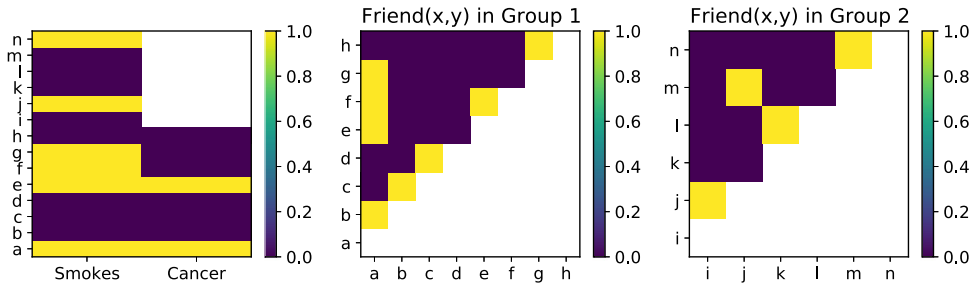
(a) Incomplete facts in the knowledge-base: axioms for smokers and cancer for individuals a to n (left), friendship relations in group 1 (middle), and friendship relations in group 2 (right).
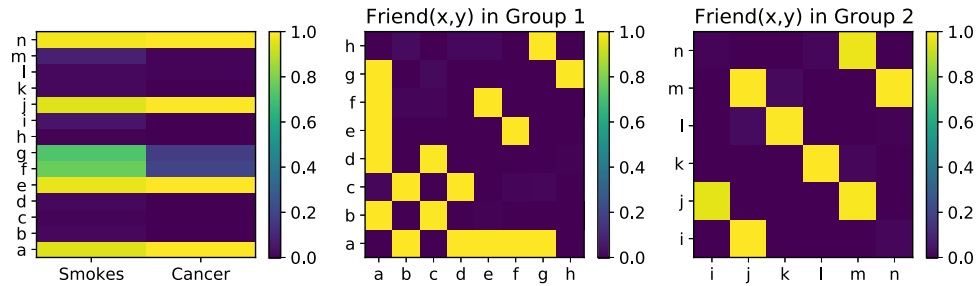
(b) Querying all the truth-values using LTN after training: smokers and cancer (left), friendship relations (middle and right).

Fig. 20. Smoker-Friends-Cancer example: illustration of the facts before and after training.
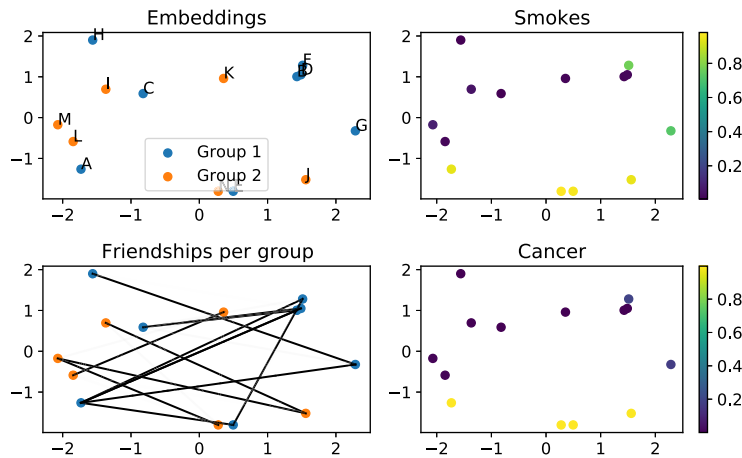
Fig. 21. Smoker-Friends-Cancer example: learned embeddings showing the result of applying PCA on the individuals (top left); truth-values of smokes and cancer predicates for each embedding (top and bottom right); illustration of the friendship relations which are satisfied after learning (bottom left).

- Reasoning by *refutation* one seeks to find out a counter-example: a grounding that satisfies the knowledge-base $\mathcal{K}$ but not the formula ϕ given the threshold q. A search is performed here using a different objective function.

We now demonstrate that reasoning by refutation is the preferred option using a simple example where we seek to find out whether $(A \vee B) \models_q A$.

Propositional Variables:

The symbols A and B denote two propositional variables.

27

Fig. 22. Querying after learning: 10 runs of the optimizer with objective $\mathcal{G}^* = \text{argmax}_{\mathcal{G}_\theta}(\mathcal{G}_\theta(\mathcal{K}))$. All runs converge to the optimum $\mathcal{G}_1$; the grid search misses the counter-example.

Fig. 23. Reasoning by refutation: one run of the optimizer with objective $\mathcal{G}^* = \text{argmin}_{\mathcal{G}_\theta}(\mathcal{G}_\theta(\phi) + \text{elu}(\alpha, \b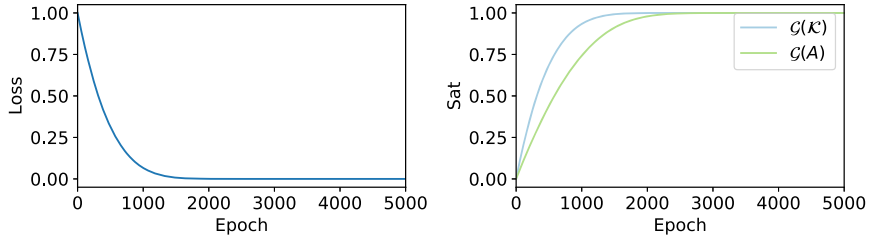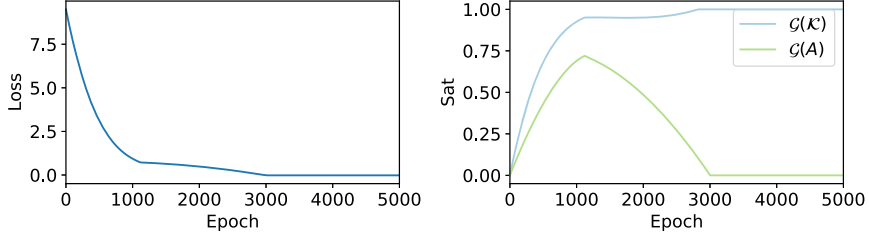eta(q - \mathcal{G}_\theta(\mathcal{K}))))$, $q = 0.95$, $\alpha = 0.05$, $\beta = 10$. In the first training epochs, the directed search prioritizes the satisfaction of the knowledge base. Then, the minimization of $\mathcal{G}_\theta(\phi)$ starts to weigh in more and the search focuses on finding a counter-example. Eventually, the run converges to the optimum $\mathcal{G}_3$, which refutes the logical consequence.

Axioms:

$$A \vee B \tag{60}$$

Grounding:

$\mathcal{G}(A) = a$, $\mathcal{G}(B) = b$, where a and b are two real-valued parameters. The set of parameters is therefore $\theta = \{a, b\}$. At initialization, $a = b = 0$.

We use the probabilistic-sum S_P to approximate $\vee$, resulting in the following satisfiability measure[28]:

$$\mathcal{G}_\theta(\mathcal{K}) = \mathcal{G}_\theta(A \vee B) = a + b - ab. \tag{61}$$

There are infinite global optima maximizing the satisfiability of the theory, as any $\mathcal{G}_\theta$ such that $\mathcal{G}_\theta(A) = 1$ (resp. $\mathcal{G}_\theta(B) = 1$) gives a satisfiability $\mathcal{G}_\theta(\mathcal{K}) = 1$ for any value of $\mathcal{G}_\theta(B)$ (resp. $\mathcal{G}_\theta(A)$). As expected, the following groundings are examples of global optima:

$\mathcal{G}_1$: $\mathcal{G}_1(A) = 1$, $\mathcal{G}_1(B) = 1$, $\mathcal{G}_1(\mathcal{K}) = 1$,
$\mathcal{G}_2$: $\mathcal{G}_2(A) = 1$, $\mathcal{G}_2(B) = 0$, $\mathcal{G}_2(\mathcal{K}) = 1$,
$\mathcal{G}_3$: $\mathcal{G}_3(A) = 0$, $\mathcal{G}_3(B) = 1$, $\mathcal{G}_3(\mathcal{K}) = 1$.

Reasoning:

$(A \vee B) \models_q A$? That is, given the threshold $q = 0.95$, does every $\mathcal{G}_\theta$ such that $\mathcal{G}_\theta(\mathcal{K}) \geqslant q$ verify $\mathcal{G}_\theta(\phi) \geqslant q$. Immediately, one can notice that this is not the case. For instance, the grounding $\mathcal{G}_3$ is a counter-example.

If one simply reasons by querying multiple groundings after learning with the usual objective $\text{argmax}_{(\mathcal{G}_\theta)} \mathcal{G}_\theta(\mathcal{K})$, the results will all converge to $\mathcal{G}_1$: $\frac{\partial \mathcal{G}_\theta(\mathcal{K})}{\partial a} = 1 - b$ and $\frac{\partial \mathcal{G}_\theta(\mathcal{K})}{\partial b} = 1 - a$. Every run of the optimizer will increase a and b simultaneously until they reach the optimum $a = b = 1$. Because the grid search always converges to the same point, no counter-example is found and the logical consequence is mistakenly assumed true. This is illustrated in Fig. 22.

Reasoning by refutation, however, the objective function has an incentive to find a counter-example with $\neg A$, as illustrated in Fig. 23. LTN converges to the optimum $\mathcal{G}_3$, which refutes the logical consequence.

5. Related work

The past years have seen considerable work aiming to integrate symbolic systems and neural networks. We shall focus on work whose objective is to build computational models that integrate deep learning and logical reasoning into a so-called end-to-end (fully differentiable) architecture. We summarize a categorization in Fig. 24 where the class containing LTN is further expanded into three sub-classes. The sub-class highlighted in red is the one that contains LTN. The reason

---

[28] We use the notation $\mathcal{G}(\mathcal{K}) := \text{SatAgg}_{\phi \in \mathcal{K}}(\mathcal{K}, \mathcal{G})$.
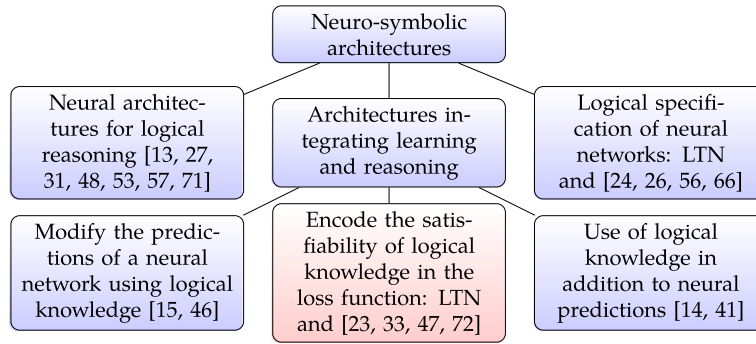
Fig. 24. Three classes of neurosymbolic approaches with *Architectures Integrating Learning and Reasoning* further sub-divided into three sub-classes, with LTN belonging to the sub-class highlighted in red.

why one may wish to combine symbolic AI and neural networks into a neurosymbolic AI system may vary, cf. [17] for a recent comprehensive overview of approaches and challenges for neurosymbolic AI.

5.1. Neural architectures for logical reasoning

These use neural networks to perform (probabilistic) inference on logical theories. Early work in this direction has shown correspondences between various logical-symbolic systems and neural network models [27,31,51,61,63]. They have also highlighted the limits of current neural networks as models for knowledge representation. In a nutshell, current neural networks (including deep learning) have been shown capable of representing propositional logic, nonmonotonic logic programming, propositional modal logic, and fragments of first-order logic, but not full first-order or higher-order logic. Recently, there has been a resurgence of interest in the topic with many proposals emerging [13,47,52]. In [13], each clause of a Stochastic Logic Program is converted into a factor graph with reasoning becoming differentiable so that it can be implemented by deep networks. In [48], a differentiable unification algorithm is introduced with theorem proving sought to be carried out inside the neural network. Furthermore, in [11,48] neural networks are used to learn reasoning strategies and logical rule induction.

Reasoning with LTN (Section 3.4) is reminiscent of this category, given that knowledge is not represented in a traditional logical language but in Real Logic.

5.2. Logical specification of neural network architectures

Here the goal is to use a logical language to specify the architecture of a neural network. Examples include [13,24,26, 55,64]. In [26], the languages of extended logic programming (logic programs with negation by failure) and answer set programming are used as background knowledge to set up the initial architecture and set of weights of a recurrent neural network, which is subsequently trained from data using backpropagation. In [24], first-order logic programs in the form of Horn clauses are used to define a neural network that can solve Inductive Logic Programming tasks, starting from the most specific hypotheses covering the set of examples. Lifted relational neural networks [64] is a declarative framework where a Datalog program is used as a compact specification of a diverse range of existing advanced neural architectures, with a particular focus on Graph Neural Networks (GNNs) and their generalizations. In [55] a weighted Real Logic is introduced and used to specify neurons in a highly modular neural network that resembles a tree structure, whereby neurons with different activation functions are used to implement the different logic operators.

To some extent, it is also possible to specify neural architectures using logic in LTN. For example, a user can define a classifier $P(x, y)$ as the formula $P(x, y) = (Q(x, y) \land R(y)) \lor S(x, y)$. $\mathcal{G}(P)$ becomes a computational graph that combines the sub-architectures $\mathcal{G}(Q)$, $\mathcal{G}(R)$, and $\mathcal{G}(S)$ according to the syntax of the logical formula.

5.3. Neurosymbolic architectures for the integration of inductive learning and deductive reasoning

These architectures seek to enable the integration of inductive and deductive reasoning in a unique fully differentiable framework [15,23,40,45,46]. The systems that belong to this class combine a neural component with a logical component. The former consists of one or more neural networks, the latter provides a set of algorithms for performing logical tasks such as model checking, satisfiability, and logical consequence. These two components are tightly integrated so that learning and inference in the neural component are influenced by reasoning in the logical component and vice versa. Logic Tensor Networks belong to this category. Neurosymbolic architectures for integrating learning and reasoning can be further separated into three sub-classes:

1. Approaches that introduce additional layers to the neural network to encode logical constraints which modify the predictions of the network. This sub-class includes Deep Logic Models [45] and Knowledge Enhanced Neural Networks [15].

2. Approaches that integrate logical knowledge as additional constraints in the objective function or loss function used to train the neural network (LTN and [23,32,46]).
3. Approaches that apply (differentiable) logical inference to compute the consequences of the predictions made by a set of base neural networks. Examples of this sub-class are DeepProblog [40] and Abductive Learning [14].

In what follows, we revise recent neurosymbolic architectures in the same class as LTN: *Integrating learning and reasoning.*

Systems that modify the predictions of a base neural network   Among the approaches that modify the predictions of the neural network using logical constraints are Deep Logic Models [45] and Knowledge Enhanced Neural Networks [15]. Deep Logic Models (DLM) are a general architecture for learning with constraints. Here, we will consider the special case where constraints are expressed by logical formulas. In this case, a DLM predicts the truth-values of a set of n ground atoms of a domain $\Delta = \{a_1, \ldots, a_k\}$. It consists of two models: a neural network $f(\boldsymbol{x} \mid \boldsymbol{w})$ which takes as input the features $\boldsymbol{x}$ of the elements of Δ and produces as output an evaluation $\boldsymbol{f}$ for all the ground atoms, i.e. $\boldsymbol{f} \in [0, 1]^n$, and a probability distribution $p(\boldsymbol{y} \mid \boldsymbol{f}, \lambda)$ which is modeled by an undirected graphical model of the exponential family with each logical constraint characterized by a clique that contains the ground atoms, rather similarly to GNNs. The model returns the assignment to the atoms that maximize the weighted truth-value of the constraints and minimize the difference between the prediction of the neural network and a target value $\boldsymbol{y}$. Formally:

$$\text{DLM}(\boldsymbol{x} \mid \lambda, \boldsymbol{w}) = \underset{\boldsymbol{y}}{\text{argmax}} \left(\sum_c \lambda_c \Phi_c(\boldsymbol{y}_c) - \frac{1}{2} ||\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x} \mid \boldsymbol{w})||^2 \right)$$

Each $\Phi_c(\boldsymbol{y}_c)$ corresponds to a ground propositional formula which is evaluated w.r.t. the target truth assignment $\boldsymbol{y}$, and λ_c is the weight associated with formula Φ_c. Intuitively, the upper model (the undirected graphical model) should modify the prediction of the lower model (the neural network) minimally to satisfy the constraints. $\boldsymbol{f}$ and $\boldsymbol{y}$ are truth-values of all the ground atoms obtained from the constraints appearing in the upper model in the domain specified by the data input.

Similar to LTN, DLM evaluates constraints using fuzzy semantics. However, it considers only propositional connectives, whereas universal and existential quantifiers are supported in LTN.

Inference in DLM requires maximizing the prediction of the model, which might be prohibitive in the presence of a large number of instances. In LTN, inference involves only a forward pass through the neural component which is rather simple and can be carried out in parallel. However, in DLM the weight associated with constraints can be learned, while in LTN they are specified in the background knowledge.

The approach taken in Knowledge Enhanced Neural Networks (KENN) [15] is similar to that of DLM. Starting from the predictions $\boldsymbol{y} = f_{nn}(\boldsymbol{x} \mid \boldsymbol{w})$ made by a base neural network $f_{nn}(\cdot \mid \boldsymbol{w})$, KENN adds a *knowledge enhancer*, which is a function that modifies $\boldsymbol{y}$ based on a set of weighted constraints formulated in terms of clauses. The formal model can be specified as follows:

$$\text{KENN}(\boldsymbol{x} \mid \lambda, \boldsymbol{w}) = \sigma \left(f'_{nn}(\boldsymbol{x} \mid \boldsymbol{w}) + \sum_c \lambda_c \cdot (\text{softmax}(\text{sign}(c) \odot f'_{nn}(\boldsymbol{x} \mid \boldsymbol{w})) \odot \text{sign}(c)) \right)$$

where $f'_{nn}(\boldsymbol{x} \mid \boldsymbol{w})$ are the pre-activations of $f_{nn}(\boldsymbol{x} \mid \boldsymbol{w})$, $\text{sign}(c)$ is a vector of the same dimension of $\boldsymbol{y}$ containing $1, -1$ and $-\infty$, such that $\text{sign}(c)_i = 1$ (resp. $\text{sign}(c)_i = -1$) if the i-th atom occurs positively (resp. negatively) in c, or $-\infty$ otherwise, and $\odot$ is the element-wise product. KENN learns the weights λ of the clauses in the background knowledge and the base network parameters $\boldsymbol{w}$ by minimizing some standard loss, (e.g. cross-entropy) on a set of training data. If the training data is inconsistent with the constraint, the weight of the constraint will be close to zero. This intuitively implies that the latent knowledge present in the data is preferred to the knowledge specified in the constraints. In LTN, instead, training data and logical constraints are represented uniformly with a formula, and we require that they are both satisfied. A second difference between KENN and LTN is the language: while LTN supports constraints written in full first-order logic, constraints in KENN are limited to universally quantified clauses.

Systems that add knowledge to a neural network by adding a term to the loss function   In [32], a framework is proposed that learns simultaneously from labeled data and logical rules. The proposed architecture is made of a *student* network f_{nn} and a *teacher* network, denoted by q. The student network is trained to do the actual predictions, while the teacher network encodes the information of the logical rules. The transfer of information from the teacher to the student network is done by defining a joint loss $\mathscr{L}$ for both networks as a convex combination of the loss of the student and the teacher. If $\tilde{\boldsymbol{y}} = f_{nn}(\boldsymbol{x} \mid \boldsymbol{w})$ is the prediction of the student network for input $\boldsymbol{x}$, the loss is defined as:

$$(1 - \pi) \cdot \mathscr{L}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) + \pi \cdot \mathscr{L}(q(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}), \tilde{\boldsymbol{y}})$$

where $q(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}) = \exp\left(-\sum_c \lambda_c (1 - \phi_c(\boldsymbol{x}, \tilde{\boldsymbol{y}}))\right)$ measures how much the predictions $\tilde{\boldsymbol{y}}$ satisfy the constraints encoded in the set of clauses $\{\lambda_c : \phi_c\}_{c \in C}$. Training is iterative. At every iteration, the parameters of the student network are optimized to minimize the loss that takes into account the feedback of the teacher network on the predictions from the previous step. The main difference between this approach and LTN is how the constraints are encoded in the loss. LTN integrates the

constraints in the network and optimizes directly their satisfiability with no need for additional training data. Furthermore, the constraints proposed in [32] are universally quantified formulas only.

The approach adopted by LYRICS [46] is analogous to the first version of LTN [59]. Logical constraints are translated into a loss function that measures the (negative) satisfiability level of the network. Differently from LTN, formulas in LYRICS can be associated with weights that are hyper-parameters. In [46], a logarithmic loss function is also used when the product t-norm is adopted. Notice that weights can also be added (indirectly) to LTN by introducing a 0-ary predicate p_w to represent a constraint of the form $p_w \wedge \phi$. An advantage of this approach would be that the weights could be learned.

In [69], a neural network computes the probability of some events being true. The neural network should satisfy a set of propositional logic constraints on its output. These constraints are compiled into arithmetic circuits for weighted model counting, which are then used to compute a loss function. The loss function then captures how close the neural network is to satisfying the propositional logic constraints.

Systems that apply logical reasoning on the predictions of a base neural network   The most notable architecture in this category is DeepProblog [40]. DeepProblog extends the ProbLog framework for probabilistic logic programming to allow the computation of probabilistic evidence from neural networks. A ProbLog program is a logic program where facts and rules can be associated with probability values. Such values can be learned. Inference in ProbLog to answer a query q is performed by knowledge compilation into a function $p(q \mid \lambda)$ that computes the probability that q is true according to the logic program with relative frequencies λ. In DeepProbLog, a neural network f_{nn} that outputs a probability distribution $\boldsymbol{t} = (t_1, \ldots, t_n)$ over a set of atoms $\boldsymbol{a} = (a_1, \ldots, a_n)$ is integrated into ProbLog by extending the logic program with $\boldsymbol{a}$ and the respective probabilities $\boldsymbol{t}$. The probability of a query q is then given by $p'(q \mid \lambda, f_{nn}(x \mid \boldsymbol{w}))$, where x is the input of f_{nn} and p' is the function corresponding to the logic program extended with $\boldsymbol{a}$. Given a set of queries $\boldsymbol{q}$, input vectors $\boldsymbol{x}$ and ground-truths $\boldsymbol{y}$ for all the queries, training is performed by minimizing a loss function that measures the distance between the probabilities predicted by the logic program and the ground-truths, as follows:

$$\mathscr{L}(\boldsymbol{y}, p'(\boldsymbol{q} \mid \lambda, f_{nn}(\boldsymbol{x} \mid \boldsymbol{w})))$$

The most important difference between DeepProbLog and LTN concerns the logic on which they are based. DeepProbLog adopts probabilistic logic programming. The output of the base neural network is interpreted as the probability of certain atoms being true. LTN instead is based on many-valued logic. The predictions of the base neural network are interpreted as fuzzy truth-values (though previous work [65] also formalizes Real Logic as handling probabilities with relaxed constraints). This difference of logic leads to the second main difference between LTN and DeepProblog: their inference mechanism. DeepProblog performs probabilistic inference (based on model counting) while LTN inference consists of computing the truth-value of a formula starting from the truth-values of its atomic components. The two types of inference are incomparable. However, computing the fuzzy truth-value of a formula is more efficient than model counting, resulting in a more scalable inference task that allows LTN to use full first-order logic with function symbols. In DeepProblog, to perform probabilistic inference, a closed-world assumption is made and a function-free language is used. Typically, DeepProbLog clauses are compiled into Sentential Decision Diagrams (SDDs) to accelerate inference considerably [35], although the compilation step of clauses into the SDD circuit is still costly.

An approach that extends the predictions of a base neural network using abductive reasoning is [14]. Given a neural network $f_{nn}(\boldsymbol{x} \mid \boldsymbol{w})$ that produces a crisp output $\boldsymbol{y} \in \{0, 1\}^n$ for n predicates $p_1, \ldots, p_n$ and background knowledge in the form of a logic program p, parameters $\boldsymbol{w}$ of f_{nn} are learned alongside a set of additional rules Δ_C that define a new concept C w.r.t. $p_1, \ldots, p_n$ such that, for every object o with features $\boldsymbol{x}_o$:

$$\begin{aligned} p \cup f_{nn}(\boldsymbol{x}_o \mid \boldsymbol{w}) \cup \Delta_C &\models C(o) \quad && \text{if } o \text{ is an instance of } C \\ p \cup f_{nn}(\boldsymbol{x}_o \mid \boldsymbol{w}) \cup \Delta_C &\models \neg C(o) \quad && \text{if } o \text{ is not an instance of } C \end{aligned} \qquad (62)$$

The task is solved by iterating the following three steps:

1. Given the predictions of the neural network $\{f_{nn}(\boldsymbol{x}_o \mid \boldsymbol{w})\}_{o \in O}$ on the set O of training objects, search for the best Δ_C that maximize the number of objects for which (62) holds;
2. For each object o, compute by abduction on $p \cup \Delta_C$, the explanation $\boldsymbol{p}'(o)$;
3. Retrain f_{nn} with the training set $\{\boldsymbol{x}_o, \boldsymbol{p}'(o)\}_{o \in O}$.

Differently from LTN, in [14] the optimization is done separately in an iterative way. The semantics of the logic is crisp, neither fuzzy nor probabilistic, and therefore not fully differentiable. Abductive reasoning is adopted, which is a potentially relevant addition for comparison with symbolic ML and Inductive Logic Programming approaches [49].

Various other loosely-coupled approaches have been proposed recently such as [43], where image classification is carried out by a neural network in combination with reasoning from text data for concept learning at a higher level of abstraction than what is normally possible with pixel data alone. The proliferation of such approaches has prompted Henry Kautz to propose a taxonomy for neurosymbolic AI in [33] (also discussed in [17]), including recent work combining neural networks with graphical models and graph neural networks [4,39,56], statistical relational learning [21,54], and even verification of neural multi-agent systems [2,8].

6. Conclusions and future work

In this paper, we have specified the theory and exemplified the reach of Logic Tensor Networks as a model and system for neurosymbolic AI. LTN is capable of combining approximate reasoning and deep learning, knowledge and data.

For ML practitioners, learning in LTN (see Section 3.2) can be understood as optimizing under first-order logic constraints relaxed into a loss function. For logic practitioners, learning is similar to inductive inference: given a theory, learning makes generalizations from specific observations obtained from data. Compared to other neuro-symbolic architectures (see Section 5), the LTN framework has useful properties for gradient-based optimization (see Section 2.4) and a syntax that supports many traditional ML tasks and their inductive biases (see Section 4), all while remaining computationally efficient (see Table 1).

Section 3.4 discussed reasoning in LTN. Reasoning is normally under-specified within neural networks. Logical reasoning is the task of proving if some knowledge follows from the facts which are currently known. It is traditionally achieved semantically using model theory or syntactically via a proof system. The current LTN framework approaches reasoning semantically, although it should be possible to use LTN and querying alongside a proof system. When reasoning by refutation in LTN, to find out if a statement ϕ is a logical consequence of given data and knowledge-base $\mathcal{K}$, a proof by refutation attempts to find a semantic counterexample where $\neg\phi$ and $\mathcal{K}$ are satisfied. If the search fails then ϕ is assumed to hold. This approach is efficient in LTN when we allow for a direct search to find counterexamples via gradient-descent optimization. It is assumed that ϕ, the statement to prove or disprove, is known. Future work could explore automatically inducing which statement ϕ to consider, possibly using syntactical reasoning in the process.

The paper formalizes Real Logic, the language supporting LTN. The semantics of Real Logic are close to the semantics of Fuzzy FOL with the following major differences: 1) Real Logic domains are typed and restricted to real numbers and real-valued tensors, 2) Real Logic variables are sequences of fixed length, whereas FOL variables are a placeholder for any individual in a domain, 3) Real Logic relations are interpreted as mathematical functions, whereas Fuzzy Logic relations are interpreted as fuzzy set membership functions. Concerning the semantics of connectives and quantifiers, some LTN implementations correspond to semantics for t-norm fuzzy logic, but not all. For example, the conjunction operator in stable product semantics is not a t-norm, as pointed out at the end of Section 2.4.

Integrative neural-symbolic approaches are known for either seeking to bring neurons into a symbolic system (neurons into symbols) [40] or to bring symbols into a neural network (symbols into neurons) [58]. LTN adopts the latter approach but maintaining a close link between the symbols and their grounding into the neural network. The discussion around these two options - neurons into symbols vs. symbols into neurons - is likely to take center stage in the debate around neurosymbolic AI in the next decade. LTN and related approaches are well placed to play an important role in this debate by offering a rich logical language tightly coupled with an efficient distributed implementation into TensorFlow computational graphs.

The close connection between first-order logic and its implementation in LTN makes LTN very suitable as a model for the neural-symbolic cycle [27,29], which seeks to translate between neural and symbolic representations. Such translations can take place at the level of the structure of a neural network, given a symbolic language [27], or at the level of the loss functions, as done by LTN and related approaches [13,44,45]. LTN opens up a number of promising avenues for further research:

Firstly, a continual learning approach might allow one to start with very little knowledge, build up and validate knowledge over time by querying the LTN network. Translations to and from neural and symbolic representations will enable reasoning also to take place at the symbolic level (e.g. alongside a proof system), as proposed recently in [68] with the goal of improving fairness of the network model.

Secondly, LTN should be compared in large-scale practical use cases with other recent efforts to add structure to neural networks such as the neuro-symbolic concept learner [43] and high-level capsules which were used recently to learn the *part-of* relation [37], similarly to how LTN was used for semantic image interpretation in [19].

Finally, LTN should also be compared with Tensor Product Representations, e.g. [57], which show that state-of-the-art recurrent neural networks may fail at simple question-answering tasks, despite achieving very high accuracy. Efforts in the area of transfer learning, mostly in computer vision, which seek to model systematicity could also be considered a benchmark [5]. Experiments using fewer data and therefore lower energy consumption, out-of-distribution extrapolation, and knowledge-based transfer are all potentially suitable areas of application for LTN as a framework for neurosymbolic AI based on learning from data and compositional knowledge.

Declaration of competing interest

Acknowledgement

Table A.3

Overview of the neural network architectures used in each example. Notice that in the examples, the networks are usually used with some additional layer(s) to ground symbols. For instance, in experiment 4.2, in $\mathcal{G}(P) : x, l \mapsto l^\top \texttt{softmax}(\texttt{MLP}(x))$, the softmax layer normalizes the raw predictions of MLP to probabilities in $[0, 1]$, and the multiplication with the one-hot label l selects the probability for one given class.

Task	Network	Architecture
4.1	MLP	Dense(16)*, Dense(16)*, Dense(1)
4.2	MLP	Dense(16)*, Dropout(0.2), Dense(16)*, Dropout(0.2), Dense(8)*, Dropout(0.2), Dense(1)
4.3	MLP	Dense(16)*, Dense(16)*, Dense(8)*, Dense(1)
4.4	CNN	MNISTConv, Dense(84)*, Dense(10)
	baseline − SD	MNISTConv × 2, Dense(84)*, Dense(19), Softmax
	baseline − MD	MNISTConv × 4, Dense(128)*, Dense(199), Softmax
4.5	MLP	Dense(8)*, Dense(8)*, Dense(1)
4.6	MLP	Dense(16)*, Dense(16)*, Dense(16)*, Dense(1)
4.7	MLP_S	Dense(8)*, Dense(8)*, Dense(1)
	MLP_F	Dense(8)*, Dense(8)*, Dense(1)
	MLP_C	Dense(8)*, Dense(8)*, Dense(1)

*: layer ends with an `elu` activation.
Dense(n): regular fully-connected layer of n units.
Dropout(r): dropout layer with rate r.
Conv(f, k): 2D convolution layer with f filters and a kernel of size k.
MP(w, h): max pooling operation with a $w \times h$ pooling window.
MNISTConv: Conv(6, 5)*, MP(2, 2), Conv(16, 5)*, MP(2, 2), Dense(100)*.

Appendix A. Implementation details

The LTN library is implemented in Tensorflow 2 [1] and is available from GitHub.[29] Every logical operator is grounded using Tensorflow primitives. The LTN code implements directly a Tensorflow graph. Due to Tensorflow built-in optimization, LTN is relatively efficient while providing the expressive power of FOL.

Table A.3 shows an overview of the network architectures used to obtain the results of the examples in Section 4. The LTN repository includes the code for these examples. Except if explicitly mentioned otherwise, the reported results are averaged over 10 runs using a 95% confidence interval. Every example uses a stable real product configuration to approximate Real Logic operators, and the Adam optimizer [34] with a learning rate of 0.001 to train the parameters.

Appendix B. Fuzzy operators and properties

This appendix presents the most common operators used in fuzzy logic literature (see Table B.4) and some noteworthy properties (see Table B.6) [28,36,62,67].

B.1. Negation

Definition 7. A negation is a function $N : [0, 1] \rightarrow [0, 1]$ that at least satisfies:

N1. Boundary conditions: $N(0) = 1$ and $N(1) = 0$,
N2. Monotonically decreasing: $\forall(x, y) \in [0, 1]^2, \ x \leqslant y \rightarrow N(x) \geqslant N(y)$.

Moreover, a negation is said to be *strict* if N is continuous and strictly decreasing. A negation is said to be *strong* if $\forall x \in [0, 1], \ N(N(x)) = x$.

We commonly use the standard strict and strong negation $N_S(a) = 1 - a$.

B.2. Conjunction

Definition 8. A conjunction is a function $C : [0, 1]^2 \rightarrow [0, 1]$ that at least satisfies:

C1. boundary conditions: $C(0, 0) = C(0, 1) = C(1, 0) = 0$ and $C(1, 1) = 1$,
C2. monotonically increasing: $\forall(x, y, z) \in [0, 1]^3$, if $x \leqslant y$, then $C(x, z) \leqslant C(y, z)$ and $C(z, x) \leqslant C(z, y)$.

In fuzzy logic, t-norms are widely used to model conjunction operators.

---

[29] https://github.com/logictensornetworks/logictensornetworks.

Table B.4
Common symmetric configurations.

Name	$a \wedge b$	$a \vee b$	$a \rightarrow_R c$	$a \rightarrow_S c$
Goedel	$\min(a, b)$	$\max(a, b)$	$\begin{cases} 1, & \text{if } a \leqslant c \\ c, & \text{otherwise} \end{cases}$	$\max(1 - a, c)$
Goguen/Product	$a \cdot b$	$a + b - a \cdot b$	$\begin{cases} 1, & \text{if } a \leqslant c \\ \frac{c}{a}, & \text{otherwise} \end{cases}$	$1 - a + a \cdot c$
Łukasiewicz	$\max(a + b - 1, 0)$	$\min(a + b, 1)$	$\min(1 - a + c, 1)$	$\min(1 - a + c, 1)$

Definition 9. A t-norm (triangular norm) is a function $t : [0, 1]^2 \rightarrow [0, 1]$ that at least satisfies:

T1. boundary conditions: $T(x, 1) = x$,
T2. monotonically increasing,
T3. commutative,
T4. associative.

Example 4. Three commonly used t-norms are:

$$T_M(x, y) = \min(x, y) \qquad \text{(minimum)}$$

$$T_P(x, y) = x \cdot y \qquad \text{(product)}$$

$$T_L(x, y) = \max(x + y - 1, 0) \qquad \text{(Łukasiewicz)}$$

B.3. Disjunction

Definition 10. A disjunction is a function $D : [0, 1]^2 \rightarrow [0, 1]$ that at least satisfies:

D1. boundary conditions: $D(0, 0) = 0$ and $D(0, 1) = D(1, 0) = D(1, 1) = 1$,
D2. monotonically increasing: $\forall (x, y, z) \in [0, 1]^3$, if $x \leqslant y$, then $D(x, z) \leqslant D(y, z)$ and $D(z, x) \leqslant D(z, y)$.

Disjunctions in fuzzy logic are often modeled with t-conorms.

Definition 11. A t-conorm (triangular conorm) is a function $S : [0, 1]^2 \rightarrow [0, 1]$ that at least satisfies:

S1. boundary conditions: $S(x, 0) = x$,
S2. monotonically increasing,
S3. commutative,
S4. associative.

Example 5. Three commonly used t-conorms are:

$$S_M(x, y) = \max(x, y) \qquad \text{(maximum)}$$

$$S_P(x, y) = x + y - x \cdot y \qquad \text{(probabilistic sum)}$$

$$S_L(x, y) = \min(x + y, 1) \qquad \text{(Łukasiewicz)}$$

Note that the only distributive pair of t-norm and t-conorm is T_M and S_M – that is, distributivity of the t-norm over the t-conorm, and inversely.

Definition 12. The N-dual t-conorm S of a t-norm T w.r.t. a strict fuzzy negation N is defined as:

$$\forall (x, y) \in [0, 1]^2, S(x, y) = N(T(N(x), N(y))). \tag{B.1}$$

If N is a strong negation, we also get:

$$\forall (x, y) \in [0, 1]^2, T(x, y) = N(S(N(x), N(y))). \tag{B.2}$$

Table B.5

Popular fuzzy implications and their classes. Strong implications (S-Implications) are defined using a fuzzy negation and fuzzy disjunction. Residuated implications (R-Implications) are defined using a fuzzy conjunction.

Name	$I(x, y) =$	S-Implication	R-Implication
Kleene-Dienes I_{KD}	$\max(1 - x, y)$	$S = S_M$ $N = N_S$	-
Goedel I_G	$\begin{cases} 1, & x \leqslant y \\ y, & \text{otherwise} \end{cases}$	-	$T = T_M$
Reichenbach I_R	$1 - x + xy$	$S = S_P$ $N = N_S$	-
Goguen I_P	$\begin{cases} 1, & x \leqslant y \\ y/x, & \text{otherwise} \end{cases}$	-	$T = T_P$
Łukasiewicz I_{Luk}	$\min(1 - x + y, 1)$	$S = S_L$ $N = N_S$	$T = T_L$

B.4. Implication

Definition 13. An implication is a function $I : [0, 1]^2 \to [0, 1]$ that at least satisfies:

I1. boundary Conditions: $I(0, 0) = I(0, 1) = I(1, 1) = 1$ and $I(1, 0) = 0$

Definition 14. There are two main classes of implications generated from the fuzzy logic operators for negation, conjunction and disjunction.

S-Implications *Strong implications* are defined using $x \to y = \neg x \vee y$ (*material implication*).
R-Implications *Residuated implications* are defined using $x \to y = \sup\{z \in [0, 1] \mid x \wedge z \leqslant y\}$. One way of understanding this approach is a generalization of *modus ponens*: the consequent is at least as true as the (fuzzy) conjunction of the antecedent and the implication.

Example 6. Popular fuzzy implications and their classes are presented in Table B.5.

B.5. Aggregation

Definition 15. An aggregation operator is a function $A : \bigcup_{n \in \mathbb{N}} [0, 1]^n \to [0, 1]$ that at least satisfies:

A1. $A(x_1, \ldots, x_n) \leqslant A(y_1, \ldots, y_n)$ whenever $x_i \leqslant y_i$ for all $i \in \{1, \ldots, n\}$,
A2. $A(x) = x$ for all $x \in [0, 1]$,
A3. $A(0, \ldots, 0) = 0$ and $A(1, \ldots, 1) = 1$.

Example 7. Candidates for universal quantification $\forall$ can be obtained using t-norms with $A_T(x_i) = x_i$ and $A_T(x_1, \ldots, x_n) = T(x_1, A_T(x_2, \ldots, x_n))$:

$$A_{T_M}(x_1, \ldots, x_n) = \min(x_1, \ldots, x_n) \qquad \text{(minimum)}$$

$$A_{T_P}(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i \qquad \text{(product)}$$

$$A_{T_L}(x_1, \ldots, x_n) = \max(\sum_{i=1}^{n} x_i - n + 1, 0) \qquad \text{(Łukasiewicz)}$$

Similarly, candidates for existential quantification $\exists$ can be obtained using s-norms with $A_S(x_i) = x_i$ and $A_S(x_1, \ldots, x_n) = S(x_1, A_S(x_2, \ldots, x_n))$:

$$A_{S_M}(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n) \qquad \text{(maximum)}$$

$$A_{S_P}(x_1, \ldots, x_n) = 1 - \prod_{i=1}^{n} (1 - x_i) \qquad \text{(probabilistic sum)}$$

Table B.6

Common properties for different configurations.

	(T_M, S_M, N_S)		(T_P, S_P, N_S)		(T_L, S_L, N_S)
	I_{KD}	I_G	I_R	I_P	I_{Luk}
Commutativity of $\wedge$, $\vee$	✓	✓	✓	✓	✓
Associativity of $\wedge$, $\vee$	✓	✓	✓	✓	✓
Distributivity of $\wedge$ over $\vee$	✓	✓			
Distributivity of $\vee$ over $\wedge$	✓	✓			
Distrib. of $\rightarrow$ over $\vee$, $\wedge$	✓	✓			
Double negation $\neg\neg p = p$	✓	✓	✓	✓	✓
Law of excluded middle					✓
Law of non contradiction					✓
De Morgan's laws	✓	✓	✓	✓	✓
Material Implication	✓		✓		✓
Contraposition	✓		✓		✓

$$A_{S_L}(x_1, \ldots, x_n) = \min(\sum_{i=1}^{n} x_i, 1) \tag{Łukasiewicz}$$

Following are other common aggregators:

$$A_M(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{mean}$$

$$A_{pM}(x_1, \ldots, x_n) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i^p \right)^{\frac{1}{p}} \tag{p-mean}$$

$$A_{pME}(x_1, \ldots, x_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - x_i)^p \right)^{\frac{1}{p}} \tag{p-mean error}$$

Where A_{pM} is the generalized mean, and A_{pME} can be understood as the generalized mean measured w.r.t. the errors. That is, A_{pME} measures the power of the deviation of each value from the ground truth 1. A few particular values of p yield special cases of aggregators. Notably:

- $\lim_{p \to +\infty} A_{pM}(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n)$,
- $\lim_{p \to -\infty} A_{pM}(x_1, \ldots, x_n) = \min(x_1, \ldots, x_n)$,
- $\lim_{p \to +\infty} A_{pME}(x_1, \ldots, x_n) = \min(x_1, \ldots, x_n)$,
- $\lim_{p \to -\infty} A_{pME}(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n)$.

These "smooth" min (resp. max) approximators are good candidates for $\forall$ (resp. $\exists$) in a fuzzy context. The value of p leaves more or less room for outliers depending on the use case and its needs. Note that A_{pME} and A_{pM} are related in the same way that $\exists$ and $\forall$ are related using the definition $\exists \equiv \neg\forall\neg$, where $\neg$ would be approximated by the standard negation.

We propose to use A_{pME} with $p \geqslant 1$ to approximate $\forall$ and A_{pM} with $p \geqslant 1$ to approximate $\exists$. When $p \geqslant 1$, these operators resemble the l^p norm of a vector $u = (u_1, u_2, \ldots, u_n)$, where $\|u\|_p = \left(|u_1|^p + |u_2|^p + \cdots + |u_n|^p \right)^{1/p}$. In our case, many properties of the l^p norm can apply to A_{pM} (positive homogeneity, triangular inequality, ...).

Appendix C. Analyzing gradients of generalized mean aggregators

[67] show that some operators used in Fuzzy Logics are unsuitable for use in a differentiable learning setting. Three types of gradient problems commonly arise in fuzzy logic operators.

Single-Passing The derivatives of some operators are non-null for only one argument. The gradients propagate to only one input at a time.

Vanishing Gradients The gradients vanish on some part of the domain. The learning does not update inputs that are in the vanishing domain.

Exploding Gradients Large error gradients accumulate and result in unstable updates.

Tables C.7 and C.8 summarize their conclusions for the most common operators. Also, we underline here exploding gradients issues that arise experimentally in A_{pM} and A_{pME}, which are not in the original report. Given the truth values of n propositions $(x_1, \ldots, x_n)$ in $[0, 1]^n$:

Table C.7
Gradient problems for some binary connectives. ($\boldsymbol{X}$) means that the problem only appears on an edge case.

	Single-Passing	Vanishing	Exploding
Goedel (minimum)			
T_M, S_M	$\boldsymbol{X}$		
I_{KD}	$\boldsymbol{X}$		
I_G	$\boldsymbol{X}$	$\boldsymbol{X}$	
Goguen (product)			
T_P, S_P		($\boldsymbol{X}$)	
I_R		($\boldsymbol{X}$)	
I_{KD}		$\boldsymbol{X}$	($\boldsymbol{X}$)
Łukasiewicz			
T_L, S_L		$\boldsymbol{X}$	
I_{Luk}		$\boldsymbol{X}$	

Table C.8
Gradient problems for some aggregators. ($\boldsymbol{X}$) means that the problem only appears on an edge case.

	Single-Passing	Vanishing	Exploding
A_{T_M}/A_{S_M}	$\boldsymbol{X}$		
A_{T_P}/A_{S_P}		$\boldsymbol{X}$	
A_{T_L}/A_{S_L}		$\boldsymbol{X}$	
A_{pM}			($\boldsymbol{X}$)
A_{pME}			($\boldsymbol{X}$)

1. $A_{pM}(x_1, \ldots, x_n) = \left(\frac{1}{n} \sum_i x_i^p \right)^{\frac{1}{p}}$

The partial derivatives are $\frac{\partial A_{pM}(x_1,\ldots,x_n)}{\partial x_i} = \frac{1}{n}^{\frac{1}{p}} \left(\sum_{j=1}^n x_j^p \right)^{\frac{1}{p}-1} x_i^{p-1}$.

When $p > 1$, the operator weights more for inputs with a higher true value –i.e. their partial derivative is also higher – and suits for existential quantification. When $p < 1$, the operator weights more for inputs with a lower true value and suits for universal quantification.

Exploding Gradients When $p > 1$, if $\sum_{j=1}^n x_j^p \to 0$, then $\left(\sum_{j=1}^n x_j^p \right)^{\frac{1}{p}-1} \to \infty$ and the gradients explode. When $p < 1$,

if $x_i \to 0$, then $x_i^{p-1} \to \infty$.

2. $A_{pME}(x_1, \ldots, x_n) = 1 - \left(\frac{1}{n} \sum_i (1-x_i)^p \right)^{\frac{1}{p}}$

The partial derivatives are $\frac{\partial A_{pME}(x_1,\ldots,x_n)}{\partial x_i} = \frac{1}{n}^{\frac{1}{p}} \left(\sum_{j=1}^n (1-x_j)^p \right)^{\frac{1}{p}-1} (1-x_i)^{p-1}$. When $p > 1$, the operator weights more

for inputs with a lower true value –i.e. their partial derivative is also higher – and suits for universal quantification. When $p < 1$, the operator weights more for inputs with a higher true value and suits for existential quantification.
Exploding Gradients

When $p > 1$, if $\sum_{j=1}^n (1-x_j)^p \to 0$, then $\left(\sum_{j=1}^n (1-x_j)^p \right)^{\frac{1}{p}-1} \to \infty$ and the gradients explode. When $p < 1$, if $1-x_i \to 0$, then $(1-x_i)^{p-1} \to \infty$.

We propose the following *stable product configuration* that does not have any of the aforementioned gradient problems:

$$\pi_0(x) = (1-\epsilon)x + \epsilon \tag{C.1}$$

$$\pi_1(x) = (1-\epsilon)x \tag{C.2}$$

$$N_S(x) = 1 - x \tag{C.3}$$

$$T'_P(x, y) = \pi_0(x)\pi_0(y) \tag{C.4}$$

$$S'_P(x, y) = \pi_1(x) + \pi_1(y) - \pi_1(x)\pi_1(y) \tag{C.5}$$

$$I'_R(x, y) = 1 - \pi_0(x) + \pi_0(x)\pi_1(y) \tag{C.6}$$

$$A'_{pM}(x_1, \ldots, x_n) = \left(\frac{1}{n} \sum_{i=1}^{n} \pi_0(x_i)^p \right)^{\frac{1}{p}} \qquad p \geqslant 1 \tag{C.7}$$

$$A'_{pME}(x_1, \ldots, y_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - \pi_1(x_i))^p \right)^{\frac{1}{p}} \qquad p \geqslant 1 \tag{C.8}$$

N_S is the operator for negation, T'_P for conjunction, S'_P for disjunction, I'_P for implication, A'_{pM} for existential aggregation, A'_{pME} for universal aggregation.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, TensorFlow: large-scale machine learning on heterogeneous systems, 2015, software available from tensorflow.org.
[2] Michael Akintunde, Elena Botoeva, Panagiotis Kouvaros, Alessio Lomuscio, Verifying strategic abilities of neural multi-agent systems, in: Proceedings of 17th International Conference on Principles of Knowledge Representation and Reasoning, KR2020, Rhodes, Greece, September 2020.
[3] Samy Badreddine, Michael Spranger, Injecting prior knowledge for transfer learning into reinforcement learning algorithms using logic tensor networks, arXiv:1906.06576 [cs, stat], June 2019.
[4] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, Koray kavukcuoglu, Interaction networks for learning about objects, relations and physics, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, Curran Associates Inc., USA, 2016, pp. 4509–4517.
[5] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Nan Rosemary Ke, Sebastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, Christopher Pal, A meta-transfer objective for learning to disentangle causal mechanisms, in: International Conference on Learning Representations, 2020.
[6] Federico Bianchi, Pascal Hitzler, On the capabilities of logic tensor networks for deductive reasoning, in: Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE 2019), Stanford University, Palo Alto, California, USA, March 25-27, 2019, 2019.
[7] Federico Bianchi, Matteo Palmonari, Pascal Hitzler, Luciano Serafini, Complementing logical reasoning with sub-symbolic commonsense, in: International Joint Conference on Rules and Reasoning, Springer, 2019, pp. 161–170.
[8] Rafael Borges, Artur d'Avila Garcez, Luís Lamb, Learning and representing temporal knowledge in recurrent networks, IEEE Trans. Neural Netw. 22 (12 2011) 2409–2421.
[9] Liber Běhounek, Petr Cintula, Petr Hájek, Introduction to mathematical fuzzy logic, in: Petr Cintula, Petr Hájek, Carles Noguera (Eds.), Handbook of Mathematical Fuzzy Logic, vol. 1, in: Studies in Logic, Mathematical Logic and Foundations, vol. 37, College Publications, 2011, pp. 1–102.
[10] N.A. Campbell, R.J. Mahon, A multivariate study of variation in two species of rock crab of the genus Leptograpsus, Aust. J. Zool. 22 (3) (1974) 417–425.
[11] Andres Campero, Aldo Pareja, Tim Klinger, Josh Tenenbaum, Sebastian Riedel, Logical rule induction and theory learning using neural theorem proving, CoRR, arXiv:1809.02193 [abs], 2018.
[12] Benhui Chen, Xuefen Hong, Lihua Duan, Jinglu Hu, Improving multi-label classification performance by label constraints, in: The 2013 International Joint Conference on Neural Networks (IJCNN), IEEE, 2013, pp. 1–5.
[13] William W. Cohen, Fan Yang, Kathryn Mazaitis, Tensorlog: a probabilistic database implemented using deep-learning infrastructure, J. Artif. Intell. Res. 67 (2020) 285–325.
[14] W.-Z. Dai, Q. Xu, Y. Yu, Z.-H. Zhou, Bridging machine learning and logical reasoning by abductive learning, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, NeurIPS'19, Curran Associates Inc., USA, 2019.
[15] Alessandro Daniele, Luciano Serafini, Knowledge enhanced neural networks, in: Pacific Rim International Conference on Artificial Intelligence, Springer, 2019, pp. 542–554.
[16] Artur d'Avila Garcez, Marco Gori, Luís C. Lamb, Luciano Serafini, Michael Spranger, Son N. Tran, Neural-symbolic computing: an effective methodology for principled integration of machine learning and reasoning, FLAP 6 (4) (2019) 611–632.
[17] Artur d'Avila Garcez, Luis C. Lamb, Neurosymbolic AI: the 3rd wave, 2020.
[18] Ivan Donadello, Luciano Serafini, Compensating supervision incompleteness with prior knowledge in semantic image interpretation, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019, pp. 1–8.
[19] Ivan Donadello, Luciano Serafini, Artur d'Avila Garcez, Logic tensor networks for semantic image interpretation, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 1596–1602.
[20] Dheeru Dua, Casey Graff, UCI machine learning repository, 2017.
[21] Richard Evans, Edward Grefenstette, Learning explanatory rules from noisy data, J. Artif. Intell. Res. 61 (2018) 1–64.
[22] Ronald Fagin, Ryan Riegel, Alexander Gray, Foundations of reasoning with uncertainty via real-valued logics, 2020.
[23] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, Martin Vechev, Dl2: training and querying neural networks with logic, in: International Conference on Machine Learning, 2019, pp. 1931–1941.
[24] Manoel Franca, Gerson Zaverucha, Artur d'Avila Garcez, Fast relational learning using bottom clause propositionalization with artificial neural networks, Mach. Learn. 94 (01 2014) 81–104.
[25] Dov M. Gabbay, John Woods (Eds.), The Many Valued and Nonmonotonic Turn in Logic, Handbook of the History of Logic, vol. 8, Elsevier, 2007.
[26] Artur d'Avila Garcez, Dov M. Gabbay, Krysia B. Broda, Neural-Symbolic Learning System: Foundations and Applications, Springer-Verlag, Berlin, Heidelberg, 2002.
[27] Artur d'Avila Garcez, Lus C. Lamb, Dov M. Gabbay, Neural-Symbolic Cognitive Reasoning, 1st edition, Springer Publishing Company, Incorporated, 2008.
[28] Petr Hajek, Metamathematics of Fuzzy Logic, Kluwer Academic Publishers, 1998.
[29] Barbara Hammer, Pascal Hitzler (Eds.), Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence, vol. 77, Springer, 2007.
[30] Stevan Harnad, The symbol grounding problem, Phys. D: Nonlinear Phenom. 42 (1–3) (1990) 335–346.
[31] Steffen Hölldobler, Franz J. Kurfess, CHCL - a connectionist inference system, in: Bertram Fronhöfer, Graham Wrightson (Eds.), Parallelization in Inference Systems, International Workshop, Dagstuhl Castle, Germany, December 17-18, 1990, Proceedings, in: Lecture Notes in Computer Science, vol. 590, Springer, 1990, pp. 318–342.
[32] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, Eric Xing, Harnessing deep neural networks with logic rules, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Berlin, Germany, August 2016, pp. 2410–2420.

[33] Henry Kautz, The third AI Summer, AAAI Robert S. Engelmore Memorial Lecture, in: Thirty-Fourth AAAI Conference on Artificial Intelligence, New York, NY, February 10, 2020.

[34] Diederik P. Kingma, Jimmy Ba, Adam: a method for stochastic optimization, arXiv:1412.6980 [cs], January 2017.

[35] Doga Kisa, Guy Van den Broeck, Arthur Choi, Adnan Darwiche, Probabilistic sentential decision diagrams, in: Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR), July 2014.

[36] Erich Peter Klement, Radko Mesiar, Endre Pap, Triangular Norms, Trends in Logic, vol. 8, Springer Netherlands, Dordrecht, 2000.

[37] Adam Kosiorek, Sara Sabour, Yee Whye Teh, Geoffrey E. Hinton, Stacked capsule autoencoders, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 15512–15522.

[38] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - vol. 1, NIPS'12, Curran Associates Inc., Red Hook, NY, USA, 2012, pp. 1097–1105.

[39] Luís C. Lamb, Artur d'Avila Garcez, Marco Gori, Marcelo O.R. Prates, Pedro H.C. Avelar, Moshe Y. Vardi, Graph neural networks meet neural-symbolic computing: a survey and perspective, in: Christian Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020 [scheduled for July 2020, Yokohama, Japan, postponed due to the Corona pandemic], 2020, pp. 4877–4884, ijcai.org.

[40] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, Luc De Raedt, Deepproblog: neural probabilistic logic programming, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NeurIPS'18, Curran Associates Inc., USA, 2018, pp. 3753–3763.

[41] Francesco Manigrasso, Filomeno Davide Miro, Lia Morra, Fabrizio Lamberti, Faster-LTN: a neuro-symbolic, end-to-end object detection architecture, arXiv:2107.01877 [cs], July 2021.

[42] Vasco Manquinho, Joao Marques-Silva, Jordi Planes, Algorithms for weighted Boolean optimization, in: International Conference on Theory and Applications of Satisfiability Testing, Springer, 2009, pp. 495–508.

[43] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, Jiajun Wu, The neuro-symbolic concept learner: interpreting scenes, words, and sentences from natural supervision, CoRR, arXiv:1904.12584 [abs], 2019.

[44] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, Marco Gori, Constraint-based visual generation, in: Igor V. Tetko, Vera Kurková, Pavel Karpov, Fabian J. Theis (Eds.), Artificial Neural Networks and Machine Learning - ICANN 2019: Image Processing - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part III, in: Lecture Notes in Computer Science, vol. 11729, Springer, 2019, pp. 565–577.

[45] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, Marco Gori, Integrating learning and reasoning with deep logic models, in: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II, in: Lecture Notes in Computer Science, vol. 11907, Springer, 2019, pp. 517–532.

[46] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, Marco Gori, Lyrics: a general interface layer to integrate logic inference and deep learning, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2019, pp. 283–298.

[47] Giuseppe Marra, Ondřej Kuželka, Neural Markov logic networks, arXiv preprint, arXiv:1905.13462, 2019.

[48] Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, Tim Rocktäschel, Learning reasoning strategies in end-to-end differentiable proving, 2020.

[49] Stephen H. Muggleton, Dianhuan Lin, Niels Pahlavi, Alireza Tamaddoni-Nezhad, Meta-interpretive learning: application to grammatical inference, Mach. Learn. 94 (1) (January 2014) 25–49.

[50] Karl Pearson, Liii. On lines and planes of closest fit to systems of points in space, Lond. Edinb. Dublin Philos. Mag. J. Sci. 2 (11) (1901) 559–572.

[51] Gadi Pinkas, Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge, Artif. Intell. 77 (2) (1995) 203–247.

[52] Meng Qu, Jian Tang, Probabilistic logic neural networks for reasoning, in: Advances in Neural Information Processing Systems, 2019, pp. 7712–7722.

[53] Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, Giuseppe Marra, From statistical relational to neuro-symbolic artificial intelligence, 2020.

[54] Matthew Richardson, Pedro Domingos, Markov logic networks, Mach. Learn. 62 (1–2) (February 2006) 107–136.

[55] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, Shajith Ikbal, Hima Karanam, Sumit Neelam, Ankita Likhyani, Santosh Srivastava, Logical neural networks, arXiv:2006.13155 [cs], June 2020.

[56] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, Gabriele Monfardini, The graph neural network model, Trans. Neural Netw. 20 (1) (January 2009) 61–80.

[57] Imanol Schlag, Jürgen Schmidhuber, Learning to reason with third-order tensor products, CoRR, arXiv:1811.12143 [abs], 2018.

[58] Imanol Schlag, Paul Smolensky, Roland Fernandez, Nebojsa Jojic, Jürgen Schmidhuber, Jianfeng Gao, Enhancing the transformer with explicit relational encoding for math problem solving, CoRR, arXiv:1910.06611 [abs], 2019.

[59] Luciano Serafini, Artur d'Avila Garcez, Logic tensor networks: deep learning and logical reasoning from data and knowledge, arXiv preprint, arXiv: 1606.04422, 2016.

[60] Luciano Serafini, Artur d'Avila Garcez, Learning and reasoning with logic tensor networks, in: Conference of the Italian Association for Artificial Intelligence, Springer, 2016, pp. 334–348.

[61] Lokendra Shastri, Advances in SHRUTI-A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony, Appl. Intell. 11 (1) (1999) 79–108.

[62] Yun Shi, A deep study of fuzzy implications, PhD thesis, Ghent University, 2009.

[63] Paul Smolensky, Géraldine Legendre, The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar Volume I: Cognitive Architecture (Bradford Books), The MIT Press, 2006.

[64] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, Ondrej Kuzelka, Lifted relational neural networks: efficient learning of latent relational structures, J. Artif. Intell. Res. 62 (2018) 69–100.

[65] Emile van Krieken, Erman Acar, Frank van Harmelen, Semi-supervised learning using differentiable reasoning, arXiv:1908.04700 [cs], August 2019.

[66] Emile van Krieken, Erman Acar, Frank van Harmelen, Analyzing differentiable fuzzy implications, in: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, 9 2020, pp. 893–903.

[67] Emile van Krieken, Erman Acar, Frank van Harmelen, Analyzing differentiable fuzzy logic operators, arXiv:2002.06100 [cs], February 2020.

[68] Benedikt Wagner, Artur d'Avila Garcez, Neural-symbolic integration for fairness in AI, in: Proceedings of the AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering 2021, 2021, p. 14.

[69] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, Guy Broeck, A semantic loss function for deep learning with symbolic knowledge, in: International Conference on Machine Learning, ISSN 2640-3498, PMLR, July 2018, pp. 5502–5511.