EPiC Series in Computing

Volume 94, 2023, Pages 473–493

Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Logic of Differentiable Logics: Towards a Uniform Semantics of DL

Natalia Ślusarz[1], Ekaterina Komendantskaya[1], Matthew L. Daggitt[1], Robert Stewart[1], and Kathrin Stark[1]

Heriot-Watt University, Edinburgh, United Kingdom
nds1@hw.ac.uk

Abstract

Differentiable logics (DL) have recently been proposed as a method of training neural networks to satisfy logical specifications. A DL consists of a syntax in which specifications are stated and an interpretation function that translates expressions in the syntax into loss functions. These loss functions can then be used during training with standard gradient descent algorithms. The variety of existing DLs and the differing levels of formality with which they are treated makes a systematic comparative study of their properties and implementations difficult. This paper remedies this problem by suggesting a meta-language for defining DLs that we call the Logic of Differentiable Logics, or LDL. Syntactically, it generalises the syntax of existing DLs to FOL, and for the first time introduces the formalism for reasoning about vectors and learners. Semantically, it introduces a general interpretation function that can be instantiated to define loss functions arising from different existing DLs. We use LDL to establish several theoretical properties of existing DLs and to conduct their empirical study in neural network verification.

Keywords: Differentiable Logic, Fuzzy Logic, Probabilistic Logic, Machine Learning, Training with Constraints, Types.

1 Introduction

Logics for reasoning with uncertainty and probability have long been known and used in programming: e.g. fuzzy logic [46], probabilistic logic [32] and variants of thereof in logic programming domain [9, 28, 30, 33]. Recently, rising awareness of the problems related to *machine learning verification* opened a novel application area for those ideas. *Differentiable Logics* (DLs) is a family of methods that applies key insights from fuzzy logic and probability theory to enhance this domain with property-driven learning [16].

As a motivating example, consider the problem of verification of neural networks. A neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ parametrised by a set of weights $\mathbf{w}$. A training dataset $\mathcal{X}$ is a set of pairs $(\mathbf{x}, \mathbf{y})$ consisting of an input $\mathbf{x} \in \mathbb{R}^n$ and the desired output $\mathbf{y} \in \mathbb{R}^m$. It is assumed that the outputs $\mathbf{y}$ are generated from $\mathbf{x}$ by some function $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}^m$ and that $\mathbf{x}$ is drawn from some probability distribution over $\mathbb{R}^n$. The goal of training is to use the dataset $\mathcal{X}$ to find weights $\mathbf{w}$ such that f approximates $\mathcal{H}$ well over input regions with high probability

density. The standard approach is to use a *loss function* $\mathcal{L}$, that given a pair $(\mathbf{x}, \mathbf{y})$ calculates how much $f(\mathbf{x})$ differs from the desired output $\mathbf{y}$. Gradients of $\mathcal{L}$ with respect to the network's weights can then be used to update the weights during training.

However in addition to the dataset $\mathcal{X}$, in certain problem domains we may have additional information about $\mathcal{H}$ in the form of a mathematical property ϕ that we know $\mathcal{H}$ must satisfy. Given that $\mathcal{H}$ satisfies property ϕ, we would like to ensure that f also satisfies ϕ. A common example of such a property is that $\mathcal{H}$ should be *robust*, i.e. small perturbations to the input only result in small perturbations to the output. For example, in image classification tasks changing a single pixel should not significantly alter what the image is classified as [5, 37].

Definition 1.1 (ϵ-δ-robustness)**.** Given constants $\epsilon, \delta \in \mathbb{R}$, a function f is ϵ-δ-*robust* around a point $\hat{\mathbf{x}} \in \mathbb{R}^n$ if:

$$\forall \mathbf{x} \in \mathbb{R}^n : ||\mathbf{x} - \hat{\mathbf{x}}|| \leq \epsilon \implies ||f(\mathbf{x}) - f(\hat{\mathbf{x}})|| \leq \delta \tag{1}$$

Casadio et al. [5] formulate several possible notions of robustness. Throughout this paper, we will use the term "robustness property" to refer to the above definition. The problem of verifying the robustness of neural networks has received significant attention from the verification community [35, 44], and it is known to be difficult both theoretically [21] and in practice [3]. However, even leaving aside the challenges of undecidability of non-linear real arithmetic [23], and scalability [44], the biggest obstacle to verification is that the majority of neural networks do not succeed in learning ϕ from the training dataset $\mathcal{X}$ [16, 43].

The concept of a *differentiable logic* (DL) was introduced to address this challenge by verification-motivated training. This idea is sometimes referred to as continuous verification [25, 5], referring to the loop between training and verification. The key idea in differentiable logic is to use ϕ to generate an alternative *logical loss function* $\mathcal{L}_\phi$, that calculates a penalty depending on how much f deviates from satisfying ϕ. When combined with the standard data-driven loss function $\mathcal{L}$, the network is trained to both fit the data and satisfy ϕ. A DL therefore has two components: a suitable language for expressing the properties and an interpretation function that can translate expressions in the language into a suitable loss function.

Although the idea sounds simple, developing good principles of DL design has been surprisingly challenging. The machine-learning community has proposed several DLs such as DL2 for supervised learning [13], and a signal temporal logic based-DL for reinforcement learning (STL) [40]. However, both approaches had shortcomings from the perspective of formal logic: the former fell short of introducing quantifiers as part of the language (and thus modelled robustness semi-formally in all experiments), and the latter only covered propositional fragment (which is not sufficient to reasoning about robustness).

Solutions were offered from the perspective of formal logic. In [38, 39] it was shown that one can use various propositional fuzzy logics to create loss functions. However, these fuzzy DLs did not stretch to cover the features of the DLs that came from machine learning community, and in particular could not stretch to express the above robustness property (Definition 1.1), which needs not just quantification over infinite domains (quantifiers for finite domains were given in [38]), but also a formalism to express properties concerning vectors and functions over vectors.

The first problem is thus: *There does not exist a DL that formally covers a sufficient fragment of first-order logic to express key properties in machine learning verification, such as robustness.*

The second problem has to do with formalisation of different DLs. *In many of the existing DL approaches syntax, semantics and pragmatics are not well-separated, which inhibits their formal analysis.* We have already given an example of DL2 treating quantifiers empirically

outside of the language. But the problem runs deeper than modelling quantifiers. To illustrate, let us take a fragment of syntax on which all DLs are supposed to agree. It will give us a toy propositional language

$$a := p \mid a \wedge a$$

They assume that each propositional variable p is interpreted in a domain $\mathcal{D} \subseteq \mathbb{R}$. The domains vary vastly across the DLs (from fuzzy set $[0, 1]$ to (∞, ∞) in STL) and the choice of a domain can have important ramifications for both the syntax and the semantics. For example DL2's domain $[0, \infty]$ severely restricts the translation of negation compared to other DLs. Conjunction is interpreted by $+$ in DL2 [13], by min, $\times$ and other more complex operations in different fuzzy DLs in van Krieken et al. [39]. In STL [40], in order to make loss functions satisfy a *shadow-lifting property*, the authors propose a complex formula computing conjunction, that includes natural exponents alongside other operations. But this forces them to redefine the syntax for conjunction and thus obtain a different language

$$a := p | \bigwedge_M (a_1, \ldots, a_M)$$

where $\bigwedge_M$ denotes a (non-associative!) conjunction for M elements.

As consequence of the above two problems, the third problem is *lack of unified, general syntax and semantics able to express multiple different DLs and modular on the choice of DL, that would make it possible to choose one best suited for concrete task or design new DLs in an easy way.*

In this paper, we propose a solution to all of these problems. The solution comes in a form of a meta-DL, which we call *a logic of differentiable logics (LDL)*. On the syntactic side (Section 3), it is a typed first-order language with negation and universal and existential quantification that can express properties of functions and vectors.

On the semantic side (Section 4), interpretation is defined to be parametric on the choice of the interpretation domain $\mathcal{D}$ or a particular choice of logical connectives. This parametric nature of interpretation simplifies both the theoretical study and implementations that compare different DLs. Moreover, the language has an implicit formal treatment of neural networks via a special kind of context – a solution that we found necessary in achieving a sufficient level of generality in the semantics. For the first time the semantics formally introduces the notion of a probability distribution that corresponds to the data from which the data is assumed to be drawn. We demonstrate the power of this approach by using LDL to prove soundness of various DLs in Sections 5.1.3, 5.1.4 and systematically compare their geometric properties in Section 5.2. In Section 5.3 we use LDL to provide a uniform empirical comparison of performance of all mentioned DLs on improving robustness.

2 Background

In the previous section, we informally introduced the notions of probability distributions and loss functions. We now formally define these. In what follows, quantities such as $\mathbf{x}$ which are written using a bold font refer to vectors and the notation x_i refers to the i^{th} element of $\mathbf{x}$.

2.1 Probability Distributions and Expectations

Following the standard definitions [17, 34], we start by considering a random variable X that ranges over some domain D. The probability distribution for X characterises how probable it

is for a sample from it to take a given value in the domain D. Depending on whether D is discrete or continuous, the variable is called *discrete or continuous random variable*, respectively. Formally, given a continuous random variable X with domain D, *the probability distribution function (PDF)* $p_X : D \to [0,1]$ is a function that satisfies: $\int_D p_X(x)dx = 1$.

This definition can be generalised to random vector variables $\mathbf{X}$ over the domain $\mathbf{D} = D_1 \times \ldots \times D_n$ as follows. Each element of the vector is assumed to be drawn from a random variable X_i over domain D_i. The *joint PDF* is a function $p_{X_1,\ldots,X_n} : D_1 \times \ldots \times D_n \to [0,1]$ that satisfies:

$$\int_{D_1} \ldots \int_{D_n} p_{X_1,\ldots,X_n}(x_1,\ldots,x_n)dx_n \ldots dx_1 = 1$$

For brevity we will write $p_{\mathbf{X}}$ instead of $p_{X_1,\ldots,X_n}$ and $\int_{\mathbf{D}}$ and $d\mathbf{x}$ rather than the full integral above.

Given a function $g : \mathbb{R}^n \to \mathbb{R}$, we can calculate an average value that g takes on $\mathbf{X}$, given a probability distribution $p_{\mathbf{X}}$. Formally $\mathbb{E}[g(\mathbf{X})]$, the *expected value* for $g : \mathbb{R}^n \to \mathbb{R}$ over the random variable $\mathbf{X}$, is defined as:

$$\mathbb{E}[g(\mathbf{X})] = \int_{\mathbf{D}} p_{\mathbf{X}}(\mathbf{x})g(\mathbf{x})d\mathbf{x}. \tag{2}$$

Throughout the paper we assume that different random variables are independent.

2.2 Loss Functions

In standard machine learning training, given a neural network f, a loss function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ computes a penalty proportional to the difference between the output of f on a training input x and a desired output y. There has been a large number of well performing loss functions proposed, with the most popular in classification tasks being cross-entropy loss [29, 41]:

Given a classifier $f : \mathbb{R}^n \to [0,1]^m$, the cross-entropy loss $\mathcal{L}_{ce}$ is defined as

$$\mathcal{L}_{ce}(\mathbf{x},\mathbf{y}) = -\sum_{i=1}^{m} \mathbf{y}_i \log(f(\mathbf{x})_i) \tag{3}$$

where $\mathbf{y}_i$ is the true probability of $\mathbf{x}$ belonging to class i and $f(\mathbf{x})_i$ the probability for class i as predicted by f when applied to $\mathbf{x}$.

However, this notion of a loss function is not expressive enough to capture loss functions generated by DLs. Firstly, the property ϕ may depend on other parameters apart from a labelled input/output pair. For example, the definition of robustness in Definition 1.1, depends not only on some input $\hat{x}$ but also on parameters ϵ and δ. Secondly, properties may relate more than one neural network, for example, in student-teacher scenarios [18]. Therefore we generalise our notion of a loss function as follows:

Definition 2.1 (Generalised loss function)**.** Given a set of neural networks N and parameters Γ, a *loss function* $\mathcal{L}^{N,\Gamma}$ is any function of type $N \times \Gamma \to \mathbb{R}$.

Note that we recover cross-entropy loss from this definition, by setting $N = \mathbb{R}^n \to \mathbb{R}^m$ and $\Gamma = \mathbb{R}^n \times \mathbb{R}^m$, whereas in a teacher-student scenario it might be something like $N = (\mathbb{R}^n \to \mathbb{R}^m) \times (\mathbb{R}^n \to \mathbb{R}^m)$ and $\Gamma = \mathbb{R}^n \times \mathbb{R}$. We distinguish between the neural networks and the other parameters, because during training we will differentiate the loss function with respect to the former but not the latter.

$$
\begin{array}{ll}
\langle \textit{expr} \rangle \ni e ::= x \mid f \mid r \in \mathbb{R} \mid i \in \mathbb{N} \mid b \in \mathbb{B} & \langle \textit{type} \rangle \ni \tau ::= s \to \tau \mid \text{s} \\
\quad \mid \; e\ e \mid \text{lam}\ (x : \tau)\ .\ e \mid \text{let}\ (x : \tau) = e\ \text{in}\ e & \\
\quad \mid \; \wedge \mid \vee \mid \neg \mid \Rightarrow \mid + \mid \text{-} \mid \times & \langle \textit{simple type} \rangle \ni s ::= \\
\quad \mid \; \neq \mid \leq \mid \geq \mid < \mid > \mid == & \quad \mid \; \text{Bool} \\
\quad \mid \; [e, ..., e] \mid ! & \quad \mid \; \text{Real} \\
\quad \mid \; \forall (x : \tau)\ .\ e \mid \exists (x : \tau)\ .\ e & \quad \mid \; \text{Vec}\ n \mid \text{Index}\ n \qquad \text{for } n \in \mathbb{N}
\end{array}
$$

Figure 1: Specification Language of LDL: expressions and types. For readability binary operators will often be written using infix notation, but this is syntactic sugar for the prefix form.

3 Syntax and Type-system of LDL

Figure 1 formally defines the syntax of LDL. The first line of the definition of the set $\langle \textit{expr} \rangle$ of expressions includes: bound variables x, neural network variables f and constants of types Real, Index and Bool. The second line of $\langle \textit{expr} \rangle$ defines the standard syntax for lambda functions, applications and let bindings to facilitate language modularity. In this aspect the syntax is richer then any of the existing DLs.

The third and fourth lines contain standard operations on Booleans ($\wedge$, $\vee$, $\neg$, $\Rightarrow$) and Reals ($+$, -, $\times$, $\geq$, ==, ...). The fifth line contains vector operations: $[e, ..., e]$ is used to construct vectors, and the lookup operation ! retrieves the value of a vector at the provided index, i.e. $v\ !\ i$ accesses the value at the i^{th} position in the vector v. The final line contains quantifiers.

LDL is a typed language. In the first line of the block defining the set $\langle \textit{type} \rangle$, there is the function type constructor $\tau_1 \to \tau_2$, which represents the type of functions which take inputs of type τ_1 and produce outputs of type τ_2. Next there are the standard Bool and Real types. Finally, there are: Vec n, the type of vectors of length n, and Index n, the type of indices into vectors of length n. The fact that these two types are parametrised by the size of the Vector will allow the type-system to statically eliminate specifications with out-of-bounds errors.

To illustrate LDL in use, we now present one possible encoding of the robustness property from Definition 1.1 for a network f of size 784 (28×28 pixel images).

Example 3.1 (Encoding of robustness property in LDL)**.**

$$
\begin{array}{l}
e_{robust} = \quad \text{let}\ (\textit{bounded} : \text{Vec}\ 784 \to \text{Vec}\ 784 \to \text{Real} \to \text{Bool})\ = \\
\qquad \text{lam}\ (\mathbf{v} : \text{Vec}\ 784)\ .\ \text{lam}\ (\mathbf{u} : \text{Vec}\ 784)\ .\ \text{lam}\ (a : \text{Real})\ . \\
\qquad\quad \forall\ (i : \text{Index}\ 784)\ .\ \text{let}\ (d : \text{Real}) = \mathbf{v}\ !\ i - \mathbf{u}\ !\ i\ \text{in} - a \leq d\ \wedge\ d \leq a \\
\quad \text{in} \\
\qquad \text{lam}\ (\epsilon : \text{Real})\ .\ \text{lam}\ (\delta : \text{Real})\ .\ \text{lam}\ (\hat{\mathbf{x}} : \text{Vec}\ 784)\ . \\
\qquad\quad \forall\ (\mathbf{x} : \text{Vec}\ 784)\ .\ (\textit{bounded}\ \mathbf{x}\ \hat{\mathbf{x}}\ \epsilon)\ \Rightarrow\ (\textit{bounded}\ (f\ \mathbf{x})\ (f\ \hat{\mathbf{x}})\ \delta)
\end{array}
$$

This example illustrates how LDL has already fulfilled several of our goals defined in Section 1: a) While other DLs work with different fragments of propositional and first-order logics, LDL covers the whole subset of FOL, and in particular quantifiers are first-class constructs in the language. b) LDL allows one to express properties of neural networks concisely and at a high level of abstraction. The presence of the Vec type makes explicit that the logic is intended to reason over vectors of reals. c) LDL is strongly typed, which will allow us to define a type-system, and subsequently a rigorous notion of an interpretation function over well-typed terms. In our presentation, the types have to be written explicitly, but it would be simple to remove this constraint by using a standard type-inference algorithm to infer most of them.

We now define a typing relation for well-typed expressions in LDL. A *bound variable context*, Δ, is a partial function that assigns each bound variable x currently in scope a type τ. We will use the notation $\Delta[x \to \tau]$ to represent updating Δ with a new mapping from x to τ. A *network context*, Ξ, is a function that assigns each network variable f a pair of natural numbers

$$\frac{\Xi[f] = (m, n)}{\Xi, \Delta \Vdash f : \text{Vec } m \to \text{Vec } n}\ (networkVar) \qquad \frac{\Delta[x] = \tau}{\Xi, \Delta \Vdash x : \tau}\ (boundVar)$$

$$\frac{r \in \mathbb{R}}{\Xi, \Delta \Vdash r : \text{Real}}\ (real) \qquad \frac{i \in \{0, ..., n-1\}}{\Xi, \Delta \Vdash i : \text{Index } n}\ (index) \qquad \frac{b \in \langle\!\langle \text{Bool} \rangle\!\rangle}{\Xi, \Delta \Vdash b : \text{Bool}}\ (bool)$$

$$\frac{\Xi, \Delta \Vdash e_1 : \tau_1 \to \tau_2 \quad \Xi, \Delta \Vdash e_2 : \tau_1}{\Xi, \Delta \Vdash e_1\ e_2 : \tau_2}\ (app)$$

$$\frac{\Xi, \Delta[x \to \tau] \Vdash e : \tau_2}{\Xi, \Delta \Vdash \text{lam } (x : \tau_1)\ .\ e : \tau_1 \to \tau_2}\ (lam) \qquad \frac{\Xi, \Delta \Vdash e_1 : \tau_1 \quad \Xi, \Delta[x \to \tau] \Vdash e_2 : \tau_2}{\Xi, \Delta \Vdash \text{let } (x : \tau_1) = e_1 \text{ in } e_2 : \tau_2}\ (let)$$

$$\frac{}{\Xi, \Delta \Vdash \wedge, \vee, \Rightarrow : \text{Bool} \to \text{Bool} \to \text{Bool}}\ (and)(or)(implies) \qquad \frac{}{\Xi, \Delta \Vdash \text{not} : \text{Bool} \to \text{Bool}}\ (not)$$

$$\frac{}{\Xi, \Delta \Vdash +, \times : \text{Real} \to \text{Real} \to \text{Real}}\ (add)(mul) \qquad \frac{}{\Xi, \Delta \Vdash \bowtie : \text{Real} \to \text{Real} \to \text{Bool}}\ (\bowtie)$$

$$\frac{\Xi, \Delta \Vdash e_1 : \text{Real} \quad \ldots \quad \Xi, \Delta \Vdash e_n : \text{Real}}{\Xi, \Delta \Vdash [e_1, ..., e_n] : \text{Vec } n}\ (vec) \qquad \frac{}{\Xi, \Delta \Vdash\ ! : \text{Vec } n \to \text{Index } n \to \text{Real}}\ (lookup)$$

$$\frac{\Xi, \Delta[x \to \tau] \Vdash e : \text{Bool} \quad \tau \neq \tau_1 \to \tau_2}{\Xi, \Delta \Vdash \forall (x : \tau)\ .\ e : \text{Bool}}\ (forall) \qquad \frac{\Xi, \Delta[x \to \tau] \Vdash e : \text{Bool} \quad \tau \neq \tau_1 \to \tau_2}{\Xi, \Delta \Vdash \exists (x : \tau)\ .\ e : \text{Bool}}\ (exists)$$

Figure 2: The typing relation $\Xi, \Delta \Vdash e : \tau$ for LDL expressions; $\bowtie$ stands for comparison operators $==, \neq, \leq, \geq, <, >$.

(m, n) such that m is the number of inputs to the network and n is the number of outputs. We will use the notation $\Xi[f \to (m, n)]$ to represent updating Ξ with a new mapping from f to (m, n).The set of *well-typed expressions* is the collection of all e for which there exists contexts Ξ and Δ and type τ such that $\Xi, \Delta \Vdash e : \tau$, which is defined inductively in Figure 2, holds.

When $\forall (x : \tau)\ .\ e$ is well typed and $\tau = \text{Real}$ or $\tau = \text{Vec } n$, we will say that the quantifier $\forall$ is *infinite*, otherwise the quantifier is *finite*; similarly for $\exists$. Note that for simplicity, we assume all bound syntactic variable names for quantifiers are unique. In practice this can be achieved by applying typical binding techniques such as de Brujin indices [4].

4 Loss Function Semantics of LDL

We translate the LDL syntax into loss functions in a manner that is parametric on the choice of the concrete DL. Firstly, in Section 4.1, we interpret LDL types as sets of values. Secondly, in Section 4.2 we interpret expressions, splitting the definitions in two parts: expressions whose interpretation is independent of the choice of DL (e.g. application, lambda, vectors) and expressions whose interpretation is dependent on the choice of DL (e.g. logical connectives, comparison operators). In Section 4.3 we define semantics of quantifiers uniformly for all logics.

4.1 Semantics of LDL Types

We first define a mapping $\langle\!\langle \cdot \rangle\!\rangle$ from LDL types to the set of values that expressions of that type will be mapped to:

$$\langle\!\langle \text{Real} \rangle\!\rangle = \mathbb{R} \qquad \langle\!\langle \text{Vec } n \rangle\!\rangle = \langle\!\langle \text{Real} \rangle\!\rangle^n \qquad \langle\!\langle \text{Index } n \rangle\!\rangle = \{0, \ldots, n-1\} \qquad \langle\!\langle \tau_1 \to \tau_2 \rangle\!\rangle = \langle\!\langle \tau_2 \rangle\!\rangle^{\langle\!\langle \tau_1 \rangle\!\rangle}$$

Note that the type Bool is absent as it is dependent on the choice of DL, given in Figure 4.

4.2 Semantics of LDL Expressions

The next step is to interpret well-typed expressions, $\Xi, \Delta \Vdash e : \tau$, which will be dependent on the *semantic context* of the interpretation. An expression's semantic context is comprised of three parts:

- A semantic network context N is a function that maps each network variable $f \in \Xi$, such that $\Xi[f] = (m, n)$, to a function $\mathtt{f} : \mathbb{R}^m \to \mathbb{R}^n$, the actual (external) implementation of the neural network. We refer to the set of such functions as $\langle\!\langle \Xi \rangle\!\rangle$.
- Let $q(e)$ be the set of infinitely quantified syntactic variables within expression e. A semantic quantifier context Q is a function that maps each variable x in $q(e)$ to a random variable X, from which values for x are sampled from (by discussion of Section 2.1 this also extends to cover random vector variables). We refer to the set of such functions as $\langle\!\langle q(e) \rangle\!\rangle$.
- A semantic bound context Γ is a partial function that assigns each bound variable $x \in \Delta$ a semantic value in $\langle\!\langle \Delta(x) \rangle\!\rangle$. We refer to the set of such functions as $\langle\!\langle \Delta \rangle\!\rangle$.

Example 4.1 (Semantic context)**.** Consider the LDL expression in Example 3.1: e_{robust} contains a single network variable f, and therefore the network context will be of the form: $N = [f \mapsto (\mathtt{f} : \mathbb{R}^{784} \to \mathbb{R}^{10})]$ where $\mathtt{f}$ is the actual neural network implementation (e.g. the Tensorflow/PyTorch object). The expression contains a single infinitely quantified variable $\mathbf{x}$, and therefore the quantifier context Q will be of the form $[\mathbf{x} \mapsto \mathbf{X}]$, where $\mathbf{X}$ is random vector variable of size 784, and represents the underlying distribution that the input images are being drawn from. At the top-level of the expression, no bound variables in scope and therefore Γ is empty. However, when interpreting the subexpression *bounded* $\mathbf{x}$ $\hat{\mathbf{x}}$ ϵ,the bound context Γ will be of the form: $[\mathit{bounded} \mapsto v_1,\ \epsilon \mapsto v_2,\ \hat{\mathbf{x}} \mapsto v_3,\ \mathbf{x} \mapsto v_4]$ where v_1 is the interpretation of the let bound expression *bounded*, $v_2 \in \mathbb{R}$ is the concrete value of ϵ, and $v_3, v_4 \in \mathbb{R}^{784}$ are the value passed in for $\hat{\mathbf{x}}$ and the current value of the quantified variable $\mathbf{x}$ respectively.

In general the interpretation of an expression will also be dependent on some differentiable logic L. In this paper, L will stand for either of: $DL2$ [13], STL [40], or Fuzzy DLs (G, L, Y, p) [39]. In this section, we present the formalisation for the first three ($DL2$, STL, G) while the remaining fuzzy logic variants (L, Y, p) can be found in Ślusarz et al. [47].

Therefore, in general we will use the notation $[\![e]\!]_L^{N,Q,\Gamma}$ to represent the mapping of LDL expression e to the corresponding value in the loss function semantics using logic L in the semantic context (N, Q, Γ). Figure 3 shows the definition of $[\![\cdot]\!]_L^{N,Q,\Gamma}$ for the LDL expressions whose semantics are independent of the choice of L. Note that, with the exception of network variables, the semantics is standard and could belong to any functional language.

Figure 4 interprets the expressions that depend on the choice of DL, as follows.

Booleans. When giving the semantics of LDL types in Section 4.1, we intentionally omitted the interpretation of Bool, as it is dependent on the DL. In DL2, Bool was originally mapped

$$[\![x]\!]_L^{N,Q,\Gamma} = \Gamma[x] \qquad [\![f]\!]_L^{N,Q,\Gamma} = N[f] \qquad [\![r]\!]_L^{N,Q,\Gamma} = r \qquad [\![i]\!]_L^{N,Q,\Gamma} = i$$

$$[\![\text{lam } (x:\tau)\ .\ e]\!]_L^{N,Q,\Gamma} = \lambda(y : \langle\!\langle \tau \rangle\!\rangle).[\![e]\!]_L^{N,Q,\Gamma[x\to y]} \qquad [\![e_1\ e_2]\!]_L^{N,Q,\Gamma} = [\![e_1]\!]_L^{N,Q,\Gamma}([\![e_2]\!]_L^{N,Q,\Gamma})$$

$$[\![\text{let } (x:\tau) = e_1 \text{ in } e_2]\!]_L^{N,Q,\Gamma} = [\![e_2]\!]_L^{N,Q,\Gamma[x\to[\![e_1]\!]_L^{N,Q,\Gamma}]} \qquad [\![[e_1,...,e_n]]\!]_L^{N,Q,\Gamma} =< [\![e_1]\!]_L^{N,Q,\Gamma},\ldots,[\![e_n]\!]_L^{N,Q,\Gamma} >$$

$$[\![!]\!]_L^{N,Q,\Gamma} = \lambda(a_1 : \langle\!\langle \text{Vec } n \rangle\!\rangle), (a_2 : \langle\!\langle \text{Index } n \rangle\!\rangle)\ .a_{1a_2}$$

$$[\![+]\!]_L^{N,Q,\Gamma} = \lambda(a_1, a_2 : \langle\!\langle \text{Real} \rangle\!\rangle)\ .a_1 + a_2 \qquad [\![\times]\!]_L^{N,Q,\Gamma} = \lambda(a_1, a_2 : \langle\!\langle \text{Real} \rangle\!\rangle)\ .a_1 \times a_2$$

Figure 3: Semantics of LDL expressions that are independent of the choice of DL.

Syntax	DL2 interpretation	Gödel interpretation	STL interpretation
$[\![\text{Bool}]\!]_L$	$[-\infty, 0]$	$[0, 1]$	$[-\infty, \infty]$
$[\![\top]\!]_L$	0	1	∞
$[\![\bot]\!]_L$	$-\infty$	0	$-\infty$
$[\![\neg]\!]_L$	-	$\lambda a\ .1 - a$	$\lambda a\ .-a$
$[\![\wedge]\!]_L$	$\lambda a_1, a_2\ .a_1 + a_2$	$\lambda a_1, a_2\ .\min(a_1, a_2)$	and_S
$[\![\vee]\!]_L$	$\lambda a_1, a_2\ .-a_1 \times a_2$	$\lambda a_1, a_2\ .\max(a_1, a_2)$	or_S
$[\![\Rightarrow]\!]_L$	-	$\lambda a_1, a_2\ .\max(1 - a_1, a_2)$	-
$[\![==]\!]_L$	$\lambda a_1, a_2\ .-\lvert a_1 - a_2\rvert$	$\lambda a_1, a_2 .1 - \lvert\frac{a_1-a_2}{a_1+a_2}\rvert$	$\lambda a_1, a_2\ .-\lvert a_1 - a_2\rvert$
$[\![\leq]\!]_L$	$\lambda a_1, a_2\ .-\max(a_1 - a_2, 0)$	$\lambda a_1, a_2\ .1 - \max(\frac{a_1-a_2}{a_1+a_2}, 0)$	$\lambda a_1, a_2\ .a_2 - a_1$

$$and_S = \lambda a_1, \ldots, a_M\ . \begin{cases} \frac{\sum_i a_{\min} e^{\tilde{a_i}} e^{\nu \tilde{a_i}}}{\sum_i e^{\nu \tilde{a_i}}} & \text{if } a_{\min} < 0 \\ \frac{\sum_i a e^{-\nu \tilde{a_i}}}{\sum_i e^{-\nu \tilde{a_i}}} & \text{if } a_{\min} > 0 \\ 0 & \text{if } a_{\min} = 0 \end{cases} \text{ where } \begin{array}{c} \nu \in \mathbb{R}^+ \text{ (constant)} \\ a_{\min} = \min(a_1, \ldots, a_M) \\ \tilde{a_i} = \frac{a_i - a_{\min}}{a_{\min}} \end{array}$$

or_S = analogous to and_S

Figure 4: Semantics of LDL expressions dependent on the choice of DL. Colour denotes parts of semantics added for LDL and not defined originally. In the implication row "-" denotes that interpretation of implication was not provided separately and is defined with negation and disjunction. While $\top$ is part of the syntax in the STL interpretation the semantic interpretation was not provided. We take ∞ to stand for $lim_{n\to\infty} n$.

to $[0, \infty]$, where 0 is interpreted as true, and other values were corresponding to the degree of falsity. Thus, $\top$ is interpreted as 0, but $\bot$ did not exist in DL2. In Fuzzy DLs with the domain $[0, 1]$, 1 stood for absolute truth, and other values – for degrees of (partial) truth; finally, in STL interval $[-\infty, \infty]$, all values but 0 stood for degrees of falsity and truth. Note that we swap the domain of DL2 for $[\infty, 0]$ in order to fit with the ordering of truth values in other logics. We will now see how these choices determine interpretation for predicates and connectives.

Predicates. The predicates are given by comparisons in LDL. Here we interpret just $\leq$ and $==$, for the remaining comparisons see Appendix in Ślusarz et al. [47]. Originally, only DL2 [13] had these comparisons. Inequality of two terms a_1 and a_2 can be interpreted there by a measure of how different they are. We slightly modify the DL2 translation in order to adapt

it in a similar way to other DLs, mapping the difference between a_1 and a_2 inside of the chosen domain.

Logical Connectives. The interpretation of $\wedge$ and $\vee$ follows the definitions given in the literature. As Figure 4 makes it clear, the interpretation of negation for DL2 is not defined. By design, DL2 negation is pushed inwards to the level of comparisons between terms, see Appendix in Ślusarz et al. [47]. STL translation did not have a defined interpretation for implication.

4.3 Semantics of LDL Quantifiers

So far, we have given an interpretation for quantifier-free formulae. However, none of works studied so far have included infinite quantifiers as first class constructs in the DL syntax. In this section, we therefore propose novel semantics for $[\![\forall x : \tau.\ e]\!]_L^{N,Q,\Gamma}$ and $[\![\exists x : \tau.\ e]\!]_L^{N,Q,\Gamma}$.

Finite quantifiers were introduced in [38] via finitely composed conjunction and disjunction. We extend this idea to other DLs. Given an expression $\forall x : \tau.\ e$ with a finite quantifier over the variable x, and given the interpretation $\langle\!\langle \tau \rangle\!\rangle = \{d_1, \ldots, d_n\}$, $[\![\forall x : \tau.\ e]\!]_L^{N,Q,\Gamma} = [\![\ e[x/d_1] \wedge \ldots \wedge e[x/d_n]]\!]_L^{N,Q,\Gamma}$; analogously for $\exists$ and $\vee$. Note that we have only two finite types, Index n or Bool, and for the latter we take $\langle\!\langle \text{Bool} \rangle\!\rangle = \{[\![\top]\!]_L^{N,Q,\Gamma}, [\![\bot]\!]_L^{N,Q,\Gamma}\}$ to interpret the quantifiers.

To proceed with infinite quantifiers, recall that context Q gives us a probability distribution for every first-order variable. This gives us a way to use the definition of an expectation for a function from Section 2.1 to interpret quantifiers. Recall that, for $g : \mathbb{R}^n \to \mathbb{R}$ we had:

$$\mathbb{E}[g(\mathbf{X})] = \int_{-\infty}^{\infty} p_{\mathbf{X}}(\mathbf{x}) g(\mathbf{x}) d\mathbf{x}.$$

Fischer et al. [13] were the first to interpret universally quantified formulae via expectation maximisation methods, in which case the optimised parameters were neural networks weights, and the quantified properties concerned robustness of the given neural network. Our goal is to propose a unifying approach to both universal and existential quantification that will fit with all DLs that we study here, and will not make any restricting assumptions about the universal properties that the language can express. For example, we defined LDL to admit expressions that may or may not refer to neural networks (or weights), and may express properties more general than robustness.

With this in mind, we introduce the following notation. For a function $g : \mathbb{R}^n \to \mathbb{R}$, we say that $\mathbf{x}_{\min}$ is the *global minimum* (resp. *global maximum*) if $g(\mathbf{x}_{\min}) \leq g(\mathbf{y})$ (resp. $g(\mathbf{x}_{\max}) \geq g(\mathbf{y})$) for any $\mathbf{y}$ on which g is defined. We define a γ-ball around a point $\mathbf{x}$ as follows: $\mathbb{B}_{\mathbf{x}}^{\gamma} = \{\mathbf{y} \mid ||\mathbf{x} - \mathbf{y}|| \leq \gamma\}$. We call the expectation

$$\mathbb{E}_{\min}[g(\mathbf{X})] = \lim_{\gamma \to 0} \int_{\mathbf{x} \in \mathbb{B}_{\mathbf{x}_{\min}}^{\gamma}} p_{\mathbf{X}}(\mathbf{x}) g(\mathbf{x}) d\mathbf{x}$$

minimised expected value for g (over the random variable $\mathbf{X}$). Taking $\mathbf{x} \in \mathbb{B}_{\mathbf{x}_{\max}}^{\gamma}$ in the above formula will give the definition of $\mathbb{E}_{\max}[g(\mathbf{X})]$, the *maximised expected value for g (over the random variable $\mathbf{X}$).*

The key insight when applying this is that given a logic L and the context (N, Q, Γ), the loss for the body e of the quantified expressions $\forall x : \tau.\ e$ or $\exists x : \tau.\ e$ can be calculated for any particular given semantic value for the quantified variable x. Therefore we can construct a function $\lambda y.[\![e]\!]_L^{N,Q,\Gamma[x \to y]}$ of type $\langle\!\langle \tau \rangle\!\rangle \to \mathbb{R}$ that takes in the value and interprets the body of the quantifier with respect to it. This gives us interpretation of universal and existential

quantifiers as minimised (or maximised) expected values for the interpretation of their body:

$$[\![\forall x : \tau.\ e]\!]_L^{N,Q,\Gamma} = \mathbb{E}_{\min}[(\lambda y.[\![e]\!]_L^{N,Q,\Gamma[x\to y]})(Q[x])]$$
$$[\![\exists x : \tau.\ e]\!]_L^{N,Q,\Gamma} = \mathbb{E}_{\max}[(\lambda y.[\![e]\!]_L^{N,Q,\Gamma[x\to y]})(Q[x])]$$

The formulae above refer to the minimised (resp. maximised) expectations for the function $\lambda y.[\![e]\!]_L^{N,Q,\Gamma[x\to y]}$ over the random variable $Q[x]$. Note that this is the first time we use the semantic quantifier context Q to map a syntactic variable x to a random variable. Another notable feature of the resulting interpretation is that the interpretation for quantifiers is parametric on the choice of the logic L, and that it has the advantage of being compositional which opens a new degree of generality: it allows for arbitrary nesting of quantifiers (in well-formed expressions). This will greatly simplify the proof of soundness for LDL.

We can now see that DL2's interpretation of quantifiers can be expressed as a special case of this definition. In particular, for a quantified formula $\forall x.P(x)$, where P is a robustness property, Fischer et al. [13] take $||\mathbf{x} - \hat{\mathbf{x}}|| \leq \epsilon$ and empirically compute the "worst perturbation" in the ϵ-interval from $\hat{x}$ using a "PGD adversarial attack" projected within that interval. This worst perturbation is the global minimum within the chosen ϵ-ball. And the loss function that optimises the neural network weights minimises the expectation. This key example motivates our use of expectation terminology in the interpretation of quantifiers. An alternative would be to use just the global minimum (or maximum) of a function directly when defining $[\![\forall x : \tau.\ e]\!]_L^{N,Q,\Gamma}$ and $[\![\exists x : \tau.\ e]\!]_L^{N,Q,\Gamma}$. But such a choice will not generalise over the DL2 implementation (or any other optimisation method). Another attempt to model quantification over real domains was made by Badreddine et al. [2] within the framework of *"Logic Tensor Networks"*. There, all variables were mapped to finite sequences of real numbers, by-passing explicit use of the notions of random variables and probability distributions over random variables. Intuitively, each given data set has only a finite number of objects, thus giving a finite domain to map to. This solution would not be satisfactory for a DL, that must take into consideration the fact that loss functions are ultimately designed to compute approximations of the unknown probability distribution, from which the given data is sampled (cf. also the discussion given in Introduction).

Example 4.2 (Semantics of Quantified Expressions)**.** We calculate a loss function for the robustness property in Example 3.1 with respect to DL2 logic in the context (N, Q, Γ) where $N = [f \mapsto (\mathtt{f} : \mathbb{R}^{784} \to \mathbb{R}^{10})]$, $Q = [x \mapsto \mathbf{X}]$ and Γ is empty. Since DL2 has no separate definition of implication we translate the implication in robustness property in a standard manner using negation and disjunction and pushing the negation inwards to the level of comparisons:

$$[\![e^*]\!]_{DL2}^{N,Q,\Gamma} = \lambda\epsilon.\lambda\delta.\lambda\hat{\mathbf{x}}.\ \mathbb{E}_{\min}[(\lambda\mathbf{x}. - ([\![\text{bounded}]\!]\ \mathbf{x}\ \hat{\mathbf{x}}\ \epsilon) \times ([\![\text{bounded}]\!]\ f(\mathbf{x})\ f(\hat{\mathbf{x}})\ \delta))(\mathbf{X})]$$
$$\text{where } [\![\text{bounded}]\!] = \lambda\mathbf{x}.\lambda\mathbf{y}.\lambda v. \sum_{i=0}^{783} -\max(-v - (\mathbf{x}_i - \mathbf{y}_i), 0) - \max(v - (\mathbf{x}_i - \mathbf{y}_i), 0)$$

The final problem to consider is whether the interpretation of infinite quantifiers via global minima and maxima commutes with the interpretation for $\wedge$ and $\vee$: in general it does not. Only the Gödel interpretation for these connectives (based on *min* and *max* operators) commutes with quantifiers interpreted as minimised or maximised expected values. Section 5.2 considers this problem in detail.

$$\frac{\Xi, \Delta \Vdash e : \tau_1 \to \text{Bool} \quad \Xi, \Delta \Vdash e_1 : \tau_1}{\Xi, \Delta \Vdash\!\!\Vdash e\ e_2} \qquad \frac{\Xi, \Delta \Vdash\!\!\Vdash e_1 \quad \Xi, \Delta \Vdash\!\!\Vdash e_2 \quad \square \in \{\wedge, \vee, \Rightarrow\}}{\Xi, \Delta \Vdash\!\!\Vdash e_1 \square e_2}$$

$$\frac{\Xi, \Delta \Vdash\!\!\Vdash e}{\Xi, \Delta \Vdash\!\!\Vdash \neg e} \qquad \frac{\Xi, \Delta, x : \tau \Vdash\!\!\Vdash e}{\Xi, \Delta \Vdash\!\!\Vdash \forall x : \tau.\, e} \qquad \frac{\Xi, \Delta, x : \tau \Vdash\!\!\Vdash e}{\Xi, \Delta \Vdash\!\!\Vdash \exists x : \tau.\, e}$$

Figure 5: (Well-formed) FOL Formulae in LDL.

5 Using LDL to explore properties of DLs

5.1 Soundness and Completeness of DLs

In all existing work, soundness and completeness of DLs was proved for propositional fragments of those logics. In the proofs the standard Boolean semantics of propositional classical logic is taken as a decidable procedure for membership of the set of all true formulae $[\![\top]\!]_L^{N,Q,\Gamma}$. The main soundness result establishes that if $[\![e]\!]_L^{N,Q,\Gamma} \in [\![\top]\!]_L^{N,Q,\Gamma}$ then e is *true* in the propositional logic. Completeness shows that the implication holds in the other direction as well.

However, as LDL is equipped with quantifiers such a procedure is no longer decidable, we may no longer rely on this method. For a typed FOL, one could define a Kripke-style semantics [36, 27] for characterising the set of all true formulae, or take provable FOL formulae to characterise the set of true FOL formulae. In this section we take the latter approach. Specifically, we take the set of FOL formulae provable in logic **LJ** by Gentzen [15]. We will say that a DL is *sound*, if the set of formulae that it interprets as $[\![\top]\!]_L^{N,Q,\Gamma}$ is a subset of formulae provable in **LJ**.

5.1.1 Type Soundness of Interpretation

We start by proving soundness of the typing relation in LDL. The following result, proven by induction on the typing judgement, will be useful in proving soundness of LDL:

Theorem 5.1 (Type Soundness of LDL)**.** For all differentiable logics L and well typed expressions $\Xi, \Delta \Vdash\!\!\Vdash e : \tau$, then for all $N \in \langle\!\langle \Xi \rangle\!\rangle$, $Q \in \langle\!\langle q(e) \rangle\!\rangle$ and $\Gamma \in \langle\!\langle \Delta \rangle\!\rangle$ we have $[\![e]\!]_L^{N,Q,\Gamma} \in \langle\!\langle \tau \rangle\!\rangle$.

Proof. See Appendix in Ślusarz et al. [47]. □

We will use the fact that all expressions of type Real can be interpreted as real numbers in Section 5.1.3.

Lemma 5.1 (Arithmetic evaluation of real expressions)**.** If $\Xi, \Delta \Vdash\!\!\Vdash e : \text{Real}$ then $[\![e]\!]_L^{N,Q,\Gamma} \in \mathbb{R}$.

Proof. Obtained as a corollary of Theorem 5.1 by instantiating τ with Real. □

5.1.2 Sequent Calculus LJ

Figure 5 gives a formal definition of well-formed FOL formulae in LDL. This definition is a basis for defining sequents in **LJ**.

We define the set of *FOL formulae* by induction on the formula shape, in a standard way (see Figure 5). FOL formulae are a subset of well-typed expressions of LDL. We use Γ_T to denote a set of FOL formulae. The rules in Figure 6 follow the standard formulation of **LJ** [36] (including notation Γ_T, ψ for $\Gamma_T \cup \{\psi\}$). The equivalence closure of the reduction relation (convertibility) is denoted by $\equiv$. We will assume that for each comparison relation $\bowtie$ in the

$$\frac{\Gamma_T \vdash [\![e_1]\!]_L^{N,Q,\Gamma} \bowtie^* [\![e_2]\!]_L^{N,Q,\Gamma}}{\Gamma_T \vdash e_1 \bowtie e_2}\ \textbf{(Arith-R)} \quad \frac{\Gamma_T, [\![e_1]\!]_L^{N,Q,\Gamma} \bowtie^* [\![e_2]\!]_L^{N,Q,\Gamma} \vdash e}{\Gamma_T, e_1 \bowtie e_2 \vdash e}\ \textbf{(Arith-L)}$$

$$\frac{e' \in \Gamma_T \qquad e \equiv e'}{\Gamma_T \vdash e}\ \textbf{(Axiom)} \quad \frac{}{\Gamma_T, \bot \vdash e}\ (\bot) \quad \frac{}{\Gamma_T \vdash \top}\ (\top)$$

$$\frac{\Gamma_T, e_1 \vdash e}{\Gamma_T \vdash \neg e_1, e}\ \textbf{(¬-R)} \quad \frac{\Gamma_T \vdash e_1, e}{\Gamma_T, \neg e_1 \vdash e}\ \textbf{(¬-L)} \quad \frac{\Gamma_T \vdash e_1 \qquad \Gamma_T \vdash e_2}{\Gamma_T \vdash e_1 \wedge e_2}\ \textbf{(∧-R)}$$

$$\frac{\Gamma_T, e_i \vdash e \qquad i \in \{1,2\}}{\Gamma_T, e_1 \wedge e_2 \vdash e}\ \textbf{(∧-L)} \quad \frac{\Gamma_T, e_1 \vdash e_2}{\Gamma_T \vdash e_1 \Rightarrow e_2}\ \textbf{(→-R)}$$

$$\frac{\Gamma_T, e_2 \vdash e \qquad \Gamma_T \vdash e_1}{\Gamma_T, e_1 \Rightarrow e_2 \vdash e}\ \textbf{(→-L)} \quad \frac{\Gamma_T \vdash e \qquad x \notin FV(\Gamma_T)}{\Gamma_T \vdash \forall x : \tau.\, e}\ \textbf{(∀-R)} \quad \frac{\Gamma_T \vdash e\,[w/x]}{\Gamma_T \vdash \exists x : \tau.\, e}\ \textbf{(∃-R)}$$

$$\frac{\Gamma_T, e \vdash e_1 \qquad x \notin FV(\Gamma_T)}{\Gamma_T, \exists x : \tau.\, e \vdash e_1}\ \textbf{(∃-L)} \quad \frac{\Gamma_T, e_1\,[w/x] \vdash e_2}{\Gamma_T, \forall x : \tau.\, e_1 \vdash e_2}\ \textbf{(∀-L)}$$

*Figure 6: The rules for **LJ**, standard structural rules are assumed (see Appendix in Ślusarz et al. [47]). In the rules we use w to denote any closed expression that is well-typed with an empty context Δ.*

language, we have a corresponding oracle $\bowtie^*$ that decides, for any pair r_1, r_2 of real numbers whether $r_1 \bowtie^* r_2$ holds. Formally, $r_1 \bowtie^* r_2$ returns $\top$ if the relation holds, and $\bot$ otherwise. It is standard in intuitionistic logic to model negation $\neg e$ as $e \Rightarrow \bot$, but we use an equivalent formulation from [36] that introduces intuitionistic negation explicitly in the rules. The only additional rule we need here is **(Arith)**. We assume the standard structural rules for **LJ** which can be found in Appendix in Ślusarz et al. [47].

Given this we can now define what it means for a differentiable logic L to be *sound* and *complete* with respect to a set of formulae provable in **LJ**.

Definition 5.1 (Soundness)**.** A logic L is *sound* if for any well-typed formula $\Xi, \Delta \Vdash e$, and any contexts $N \in \langle\!\langle \Xi \rangle\!\rangle$, $Q \in \langle\!\langle q(e) \rangle\!\rangle$, $\Gamma \in \langle\!\langle \Delta \rangle\!\rangle$ if $[\![e]\!]_L^{N,Q,\Gamma} = [\![\top]\!]_L^{N,Q,\Gamma}$ then $\vdash e$.

Definition 5.2 (Completeness)**.** A logic L is *complete* if for any well-typed formula $\Xi, \Delta \Vdash e$, and any contexts $N \in \langle\!\langle \Xi \rangle\!\rangle$, $Q \in \langle\!\langle q(e) \rangle\!\rangle$, $\Gamma \in \langle\!\langle \Delta \rangle\!\rangle$ if $\vdash e$ then $[\![e]\!]_L^{N,Q,\Gamma} = [\![\top]\!]_L^{N,Q,\Gamma}$.

5.1.3 Most Fuzzy DLs are sound but incomplete

We start with showing that fuzzy DL comparison operators are valid.

Lemma 5.2 (Soundness of FDL comparisons)**.** If $\Xi, \Delta \Vdash e_1 \bowtie e_2$ and any contexts $N \in \langle\!\langle \Xi \rangle\!\rangle$, $Q \in \langle\!\langle q(e) \rangle\!\rangle$, $\Gamma \in \langle\!\langle \Delta \rangle\!\rangle$, then for all L in fuzzy differentiable logics (FDL) the following holds:
If $[\![e_1 \bowtie e_2]\!]_L^{N,Q,\Gamma} = [\![\top]\!]_L^{N,Q,\Gamma}$ then $[\![e_1]\!]_L^{N,Q,\Gamma} \bowtie^* [\![e_2]\!]_L^{N,Q,\Gamma} = \top$.
If $[\![e_1 \bowtie e_2]\!]_L^{N,Q,\Gamma} = [\![\bot]\!]_L^{N,Q,\Gamma}$ then $[\![e_1]\!]_L^{N,Q,\Gamma} \bowtie^* [\![e_2]\!]_L^{N,Q,\Gamma} = \bot$.

Proof. See Appendix in Ślusarz et al. [47]. This result uses Lemma 5.1. □

Lemma 5.3. Given a well-typed formula e, for any any contexts $N \in \langle\!\langle \Xi \rangle\!\rangle$, $Q \in \langle\!\langle q(e) \rangle\!\rangle$, $\Gamma \in \langle\!\langle \Delta \rangle\!\rangle$ the following hold:

1. If $[\![e]\!]_G^{N,Q,\Gamma} = [\![\top]\!]_G^{N,Q,\Gamma}$ then $\vdash e$.
2. If $[\![e]\!]_G^{N,Q,\Gamma} = [\![\bot]\!]_G^{N,Q,\Gamma}$ then $e \vdash$.

Proof. The auxiliary lemma is proven by case analysis on e, mutual induction, and relies on Lemma 5.2, see Appendix in Ślusarz et al. [47]. □

Theorem 5.2 (Soundness of Gödel DL)**.** Gödel DL is sound.

Proof. Soundness is obtained as a corollary of Lemma 5.3. □

All other fuzzy DLs, with the exception of Łukasiewicz, are also sound with respect to **LJ**, with proofs following the same scheme. For the Łukasiewicz DL, we cannot prove the equivalent of Lemma 5.3 unless we remove either negation or implication from the syntax. Intuitively, this is because Łukasiewicz implication is not strong enough: e.g. $[\![e_1 \Rightarrow e_2]\!]$ evaluates to 1 as long as $[\![e_1]\!] \leq [\![e_2]\!]$. Proper study of this phenomenon is left for future work. We note that the recent work by Bacci et al. [1] has shed light on logical properties of Łukasiewicz logic, and we intend to build upon those results.

Theorem 5.2 does not hold for the opposite direction of implication (and thus completeness fails). To see this, consider the following derivation for $\Gamma_T \vdash \neg(5 \leq 3)$ in **LJ**, which we would have liked to be equivalent to having $[\![\neg(5 \leq 3)]\!]_G \in [\![\top]\!]_G$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma_T, \bot \vdash}\,(\bot)}{\Gamma_T, 5 \leq 3 \vdash}\,\mathbf{(Arith-L)}}{\Gamma_T \vdash \neg(5 \leq 3)}\,(\neg - \mathbf{R})$$

However, taking $[\![5 \leq 3]\!]_G = 1 - 0.25 = 0.75$, we get $[\![\neg(5 \leq 3)]\!]_G = 1 - 0.75 = 0.25 \notin [\![\top]\!]_G$. This is the problem that will be common for all Fuzzy DLs.

5.1.4 DL2 is sound and incomplete

LDL helps to establish a generic approach to proving soundness results for a variety of DLs. Firstly, a variant of Lemma 5.2 holds for DL2. Next we obtain its soundness for DL2, where DL2 has just the connectives $\wedge, \vee$ and quantifiers, as $\Rightarrow$ and $\neg$ are not defined.

Theorem 5.3 (Soundness of DL2)**.** DL2 is sound.

Proof. See Appendix in Ślusarz et al. [47]. □

The proof follows the structure of similar proofs for FDLs. This is a welcome simplification, and it will be useful in future computer formalisations of these logics. Seeing the result is given for an incomplete set of connectives, we omit discussion of completeness for DL2.

5.1.5 STL is neither sound nor complete

Incompleteness of STL is discussed in §5.1.3. In addition, note that its connectives and quantifiers lack properties that are provable in **LJ**: e.g. associativity of conjunction, or commutativity of a universal quantifier with respect to conjunction.

The definition for soundness relies on the interpretation of $[\![\top]\!]_S^{N,Q,\Gamma}$. Varnai and Dimarogonas [40] do not formally define $[\![\top]\!]_S^{N,Q,\Gamma}$, and for that reason alone soundness is unattainable for the original STL. We show an attempt to propose an interpretation $[\![\top]\!]_S^{N,Q,\Gamma} = \infty$ and show how we use LDL's generic approach to analyse the result. Informally, their intuition is that any positive number in $[0, \infty)$ belongs to $[\![\top]\!]_S^{N,Q,\Gamma}$. This motivates, for example, $[\![3 \leq 5]\!]_S^{N,Q,\Gamma} = 2$ or $[\![3 == 3]\!]_S^{N,Q,\Gamma} = 0$. But none of these evaluates to ∞. Thus, although the soundness proof goes through, the set of expressions that a variant of Lemma 5.2 for STL covers will be empty: such soundness would say nothing about the FOL fragment of LDL. A solution would be to re-define interpretation for all predicates in a binary fashion: e.g. *for* $a_1 == a_2$*, return* ∞ *if* $a_1 = a_2$ *and return* $-\infty$ *otherwise*. But this would sacrifice continuity and smoothness of the resulting loss functions, – key properties for STL design.

5.2 Logical and Geometric Properties of DLs

Both Varnai and Dimarogonas [40] and van Krieken et al. [39] suggest a selection of desirable properties for their DLs. Some are logical properties (e.g. associativity or commutativity) and some have a geometric origin, such as (weak) smoothness or shadow-lifting. As some of the DLs we consider do not have associative connectives, we will use $\bigwedge_M$ as a notation for conjunction of exactly M conjuncts. Similar definition can be made for disjunction, which we omit. In all of the following definitions we will omit the contexts for clarity as they do not change. We will denote a well-typed formula by e. Starting with geometric properties, smoothness is generally desirable from optimisation perspective as it aids gradient based methods used in neural network training [26]:

Definition 5.3 (Weak smoothness)**.** The $[\![e]\!]_L$ is *weakly smooth* if it is continuous everywhere and its gradient is continuous at all points in the interval where there is a unique minimum.

Definition 5.4 (Scale invariance for $\bigwedge_M$)**.** The $[\![e]\!]_L$ is *scale-invariant* if, for any real $\alpha \leq 0$

$$\alpha[\![\bigwedge\nolimits_M(A_1, ..., A_M)]\!]_L = \left[\!\!\left[\bigwedge\nolimits_M\right]\!\!\right]_L (\alpha[\![A_1]\!]_L, ..., \alpha[\![A_M]\!]_L)$$

Shadow-lifting is a property original to Varnai and Dimarogonas [40] and motivates the STL conjunction. It characterises gradual improvement when training the neural network: if one conjunct increases, the value of entire conjunction should increase as well.

Definition 5.5 (Shadow-lifting property for $\bigwedge_M$)**.** The $[\![e]\!]_L$ satisfies the *shadow-lifting property* if, $\forall i.[\![A_i]\!]_L \neq 0$:

$$\frac{\partial[\![\bigwedge_M(A_1, ..., A_M)]\!]_L}{\partial[\![A_i]\!]_L}\bigg|_{A_1,...,A_M} > 0$$

where ∂ denotes partial differentiation.

Coming from the logic perspective, we have the following desirable properties:

Definition 5.6 (Commutativity, idempotence and associativity of $\bigwedge_M$)**.** The $[\![e]\!]_L$ is *commutative* if for any permutation π of the integers $i \in 1, ..., M$

$$[\![\bigwedge\nolimits_M(A_1, ..., A_M)]\!]_L = [\![\bigwedge\nolimits_M(A_{k_{\pi(1)}}, ..., A_{k_{\pi(M)}})]\!]_L$$

it is *idempotent* and *associative* if

$$[\![\bigwedge\nolimits_M(A, ..., A)]\!]_L = [\![A]\!]_L$$
$$[\![\bigwedge\nolimits_2(\bigwedge\nolimits_2(A_1, A_2), A_3)]\!]_L = [\![\bigwedge\nolimits_2(A_1, \bigwedge\nolimits_2(A_2, A_3))]\!]_L$$

The below property has not appeared in the literature before, but as previous sections have shown, it proves to be important when generalising DLs to FOL:

Definition 5.7 (Quantifier commutativity)**.** The $[\![e]\!]_L$ satisfies the *quantifier commutativity* if

$$[\![\forall x. \bigwedge\nolimits_M(A_1, ..., A_M)]\!]_L = [\![\bigwedge\nolimits_M(\forall x.A_1, \ldots, \forall x.A_M)]\!]_L$$

Similarly for $\exists$ and $\vee$.

Table 1: Three groups of properties of DLs: geometric, logical and semantical. Properties which have been established in other works have relevant citations. Entries with * denote DLs for which the smoothness property holds in propositional case.

Properties:	DL2	Gödel	Łukasiewicz	Yager	Product	STL
Weak Smoothness	yes*	no	no	no	yes*	yes
Shadow-lifting	yes	no	no	no	yes	yes [40]
Scale invariance	yes	yes	no	no	no	yes [40]
Idempotence	no	yes [39]	no [6]	no [22]	no [6]	yes [40]
Commutativity	yes	yes [39]	yes [39]	yes [39]	yes [39]	yes [40]
Associativity	yes	yes [39]	yes [39]	yes [39]	yes [39]	no [40]
Quantifier commutativity	no	yes	no	no	no	no
Soundness	yes	yes	yes	yes	yes	no

Table 1 summarises how different DLs compare and contains results already established in literature (as cited) as well as provides new results. While we do not provide proofs, the reader can fine an in-depth explanation of the reasons behind the mentioned properties in Appendix of Ślusarz et al. [47]. The table inspires the following observations.

i) Starting with geometric properties, smoothness is traditionally important for differentiation [26], and connectives for DL2 and Product DL satisfy the property. However, in the FOL extension only STL remains weakly smooth, see also discussion of § 5.1.5.

ii) For the remaining geometric properties, DL2 and STL satisfy shadow-lifting and scale invariance, but Fuzzy logics disagree on these. Thus, only STL satisfies all geometric properties.

iii) For logical properties, Varnai and Dimarogonas [40] suggested that a DL cannot both satisfy shadow-lifting, idempotence and associativity. However, their proof of this fact fails within LDL. We leave formal investigation of this conjecture for future work.

iv) Since the semantics of quantifiers are defined as global minima and maxima, only connectives of Gödel DL commute with respect to them. This is a highly desirable property, that has not been previously discussed in the literature. It impacts completeness of DLs (cf. § 5.1.3 and 5.1.5).

v) Only Gödel DL satisfies all logical properties, but it fails smoothness and shadow-lifting properties. On the opposite side of the spectrum, STL behaves well geometrically, but fails associativity and quantifier commutativity, and has unresolved issues with soundness (§ 5.1.5). It warrants future research whether a DL with optimal combination of logical and geometric properties may be formulated.

5.3 Empirical Evaluation

LDL offers a unified framework to evaluate different DLs. We have implemented the syntax and semantics as an extension to the Vehicle tool [24] which supports the general syntax of LDL. The implementation is highly modular due to the semantics of each DL sharing the same semantics for the core of the syntax as explained in Section 4. We explain the workflow for our running example – the robustness property.

Taking the specification of Example 3.1, it is compiled by Vehicle to a loss function in Python where it can be used to train a chosen neural network. The loss function resulting from the LDL semantics is typically used in combination with a standard loss function such as cross-entropy. If we denote cross-entropy loss together with its inputs as $\mathcal{L}_{CE}$ and the LDL loss together with its inputs and parameters as $\mathcal{L}_{DL}$ then the final loss will be of the form:

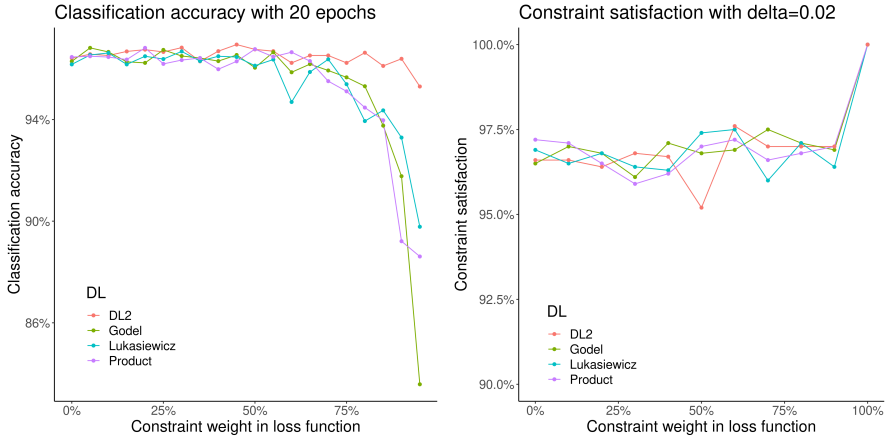

Figure 7: Performance of the network trained with different custom losses with varying parameters $\alpha, \beta \in \mathbb{R}$ measured by relative weight of β in relation to α.

$\mathcal{L} = \alpha \cdot \mathcal{L}_{CE} + \beta \cdot \mathcal{L}_{DL}$ where the weights $\alpha, \beta \in \mathbb{R}$ are parameters. For the preliminary tests we used a simple 2-layer network to classify MNIST images [11]. As seen in Figure 7 we have tested the effect of varying the $\alpha, \beta \in \mathbb{R}$ parameters and therefore the weight of the custom loss in training on accuracy and constraint satisfaction; the results are broadly consistent with trends reported in the literature previously.

We note that formulating heuristics for finding minima/maxima of loss functions, which are necessary for the implementation of quantifier interpretation, can be very complex, and are beyond the scope of this paper. Figure 7 shows results implemented using random sampling instead of sampling around global maxima or minima. We leave implementation of heuristics of finding global minima/maxima, as well as thorough experimental study, for future work.

6 Conclusions, Related and Future Work

Conclusions. We have presented a general language (LTL) for expressing properties of neural networks. Our contributions are: 1) LDL generalises other known DLs and provides a language that is rich enough to express complex verification properties; 2) with LDL we achieved the level of rigour that allows to formally separate the formal language from its semantics, and thus opens a way for systematic analysis of properties of different DLs; 3) by defining different DLs in LDL, we proved properties concerning their soundness and resulting loss functions, and opened the way for uniform empirical evaluation of DLs. We now discuss related work.

Learning Logical Properties. There are many methods of passing external knowledge in the form of logical constraints to neural networks. The survey by Giunchiglia et al. [16] discusses multitude of approaches including methods based on loss functions [45, 13, 39], to which LDL belongs, but also others such as tailoring neural network architectures [14], or guaranteeing constraint satisfaction at the level of neural network outputs [19, 12]. As Giunchiglia et al. [16] establish, the majority of approaches are tailored to specific problems, and only Fischer et al. [13] and van Krieken et al. [39] go as far as to include quantifiers. LDL generalises both.

Analysis of properties of loss functions. Property analysis, especially of smoothness [26] or bilateral properties [31], is a prominent field [42]. One of LDL's achievements is to expose trade-offs between satisfying desired geometric and logic properties of a loss functions.

Neural Network Verification. While this work does not attempt to *verify* neural networks, we draw our motivation from this area of research. It has been observed in verification literature that neural networks often fail to satisfy logical constraints [43]. One of proposed solutions is training the NN to satisfy a constraint prior to verifying them [20, 45]. This belongs to an approach referred to as *continuous verification* [25, 5] which focuses on the cycle between training and verification. LDL fits into this trend. Indeed, the tool Vehicle that implements LDL is also built to work with NN verifiers [8].

Logics for Uncertainty and Probabilistic Logics. LDLs have a strong connection to fuzzy logic [39]. Via the use of probability distributions and expectations, we draw our connection to Probabilistic Prolog and similar languages [9, 10]. We differ as none of those approaches can be used to formulate loss functions, which is the main goal of LDL.

Adversarial training. Starting with the seminal paper by Szegedy et al. [37], thousands of papers in machine learning literature have been devoted to adversarial attacks and robust training of neural networks. Majority of those papers does not use a formal logical language for attack or loss function generation. LDL opens new avenues for this community, as allows one to formulate other properties of interest apart from robustness, see e.g. [5, 7].

Future work. We intend to use LDL to further study the questions of a suitable semantics for DL, both proof-theoretic and denotational, possibly taking inspiration from [1]. For proof-theoretic semantics, we need to find new DLs with tighter correspondence to **LJ**; and moreover new calculi can be developed on the basis of **LJ** to suit the purpose. Since loss functions are used in computation, we conjecture that constructive logics (or semi-constructive logics as in [1]) will be useful in this domain. For denotational semantics, we intend to explore Kripke frame semantics. Future work could also include finding the best combination of logical and geometric properties in a DL and formulating new DLs that satisfy them. Thorough evaluation of performance of all the DLs is also left to future work: it includes formulating heuristics for finding local/global maxima for quantifier interpretation, and evaluation of LDL effectiveness within the continuous verification cycle. Finally, finding novel ways of defining quantifiers that commute with DL connectives is an interesting challenge.

7 Acknowledgements

This work was supported by the EPSRC grant EP/T026952/1, *AISEC: AI Secure and Explainable by Construction* and the EPSRC DTP Scholarship for N. Ślusarz. We thank anonymous referees, James McKinna, Wen Kokke, Bob Atkey and Emile Van Krieken for valuable comments on the early versions of this paper, and Marco Casadio for contributions to the Vehicle implementation.

References

[1] Giorgio Bacci, Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Propositional logics for the lawvere quantale. *CoRR*, abs/2302.01224, 2023. doi: 10.48550/arXiv.2302.01224. URL https://doi.org/10.48550/arXiv.2302.01224.

[2] Samy Badreddine, Artur S. d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artif. Intell.*, 303:103649, 2022.

[3] S. Bak, C. Liu, and T. Johnson. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results, 2021. Technical Report. http://arxiv.org/abs/2109.00498.

[4] N.G. Bruijn, de. *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem*, pages 375–388. Studies in logic and the foundations of mathematics. North-Holland Publishing Company, Netherlands, 1994. ISBN 0-444-89822-0.

[5] Marco Casadio, Ekaterina Komendantskaya, Matthew L. Daggitt, Wen Kokke, Guy Katz, Guy Amir, and Idan Refaeli. Neural network robustness as a verification property: A principled case study. In *Computer Aided Verification (CAV 2022)*, Lecture Notes in Computer Science. Springer, 2022.

[6] Petr Cintula, Petr Hájek, and Carles Noguera. Handbook of mathematical fuzzy logic (in 2 volumes), volume 37, 38 of studies in logic, mathematical logic and foundations, 2011.

[7] Matthew L. Daggitt, Wen Kokke, Robert Atkey, Luca Arnaboldi, and Ekaterina Komendantskaya. Vehicle: A high-level language for embedding logical specifications in neural networks, 2022.

[8] Matthew L Daggitt, Robert Atkey, Wen Kokke, Ekaterina Komendantskaya, and Luca Arnaboldi. Compiling higher-order specifications to smt solvers: How to deal with rejection constructively. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 102–120, 2023.

[9] Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100:5–47, 2015.

[10] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[11] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[12] Paolo Dragone, Stefano Teso, and Andrea Passerini. Neuro-symbolic constraint programming for structured prediction. In *Procedings of International Workshop on Neural-Symbolic Learning and Reasoning*, 2021.

[13] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and querying neural networks with logic. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1931–1941. PMLR, 09–15 Jun 2019.

[14] A Garcez, M Gori, LC Lamb, L Serafini, M Spranger, and SN Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4):611–632, 2019.

[15] Gerhard Gentzen. Investigations into logical deduction. In M.E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, volume 55 of *Studies in Logic and the Foundations of Mathematics*, pages 68 – 131. Elsevier, 1969.

[16] Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*, pages 5478–5485. ijcai.org, 2022.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[19] Nick Hoernle, Rafael Michael Karampatsis, Vaishak Belle, and Kobi Gal. Multiplexnet: Towards fully satisfied logical constraints in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5700–5709, 2022.

[20] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

[21] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *International Conference on Computer Aided Verification*, 2017.

[22] Erich Peter Klement, Radko Mesiar, and Endre Pap. Triangular norms. position paper ii: general constructions and parameterized families. *Fuzzy sets and Systems*, 145(3):411–438, 2004.

[23] Wen Kokke, Ekaterina Komendantskaya, Daniel Kienitz, Bob Atkey, and David Aspinall. Neural networks, secure by construction: An exploration of refinement types. In *APLAS*, 2020.

[24] Wen Kokke, Matthew L. Daggitt, Robert Atkey, Ekaterina Komendantskaya, Luca Arnaboldi, Natalia Slusarz, and Marco Casadio. Vehicle. https://github.com/vehicle-lang/vehicle, 2023.

[25] Ekaterina Komendantskaya, Wen Kokke, and Daniel Kienitz. Continuous verification of machine learning: a declarative programming approach. In *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming*, pages 1:1–1:3. ACM, 2020.

[26] Wonyeol Lee, Xavier Rival, and Hongseok Yang. Smoothness analysis for probabilistic programs with application to optimised variational inference. *Proceedings of the ACM on Programming Languages (POPL)*, 7:335–366, 2023.

[27] James Lipton and Susana Nieva. Kripke semantics for higher-order type theory applied to constraint logic programming languages. *Theoretical Computer Science*, 712: 1–37, 2018. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2017.11.005. URL https://www.sciencedirect.com/science/article/pii/S0304397517308356.

[28] Thomas Lukasiewicz. Probabilistic logic programming. In *European Conference on Artificial Intelligence*, pages 388–392, 1998.

[29] Kamil Nar, Orhan Ocal, S. Shankar Sastry, and Kannan Ramchandran. Cross-entropy loss and low-rank features have responsibility for adversarial examples, 2019.

[30] Raymond Ng and Venkatramanan Siva Subrahmanian. Probabilistic logic programming. *Information and computation*, 101(2):150–201, 1992.

[31] Feiping Nie, Zhanxuan Hu, and Xuelong Li. An investigation for loss functions widely used in machine learning. *Communications in Information and Systems*, 18(1):37–52, 2018.

[32] Nils J Nilsson. Probabilistic logic. *Artificial intelligence*, 28(1):71–87, 1986.

[33] Fabrizio Riguzzi. *Foundations of probabilistic logic programming: Languages, semantics, inference and learning*. CRC Press, 2022.

[34] George G Roussas. *An introduction to probability and statistical inference*. Elsevier, 2003.

[35] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3:41:1–41:30, 2019.

[36] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*. Studies in Logic. Elsevier, 2006.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[38] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy implications. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 893–903, 2020.

[39] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, 2022. ISSN 0004-3702.

[40] Peter Varnai and Dimos V. Dimarogonas. On robustness metrics for learning stl tasks. In *2020 American Control Conference (ACC)*, pages 5394–5399, 2020. doi: 10.23919/ACC45564.2020.9147692.

[41] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020.

[42] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9(2):187–212, 2022.

[43] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Advances in neural information processing systems*, 31, 2018.

[44] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems*, pages 29909–29921, 2021.

[45] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5502–5511. PMLR, 10–15 Jul 2018.

[46] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[47] Natalia Ślusarz, Ekaterina Komendantskaya, Matthew L. Daggitt, Robert Stewart, and Kathrin Stark. Logic of differentiable logics: Towards a uniform semantics of dl., 2023. arXiv:2303.10650.