Extending Real Logic with Aggregate Functions

Samy Badreddine[1], Michael Spranger[1,2]

[1]Sony AI Inc, 1-7-1 Konan Minato-ku, Tokyo, 108-0075 Japan
[2]Sony Computer Science Laboratories Inc, 3-14-13 Higashi Gotanda, Tokyo, 141-0022 Japan

Abstract
Real Logic is a recently introduced first-order language where formulas have fuzzy truth values in the interval $[0, 1]$ and semantics are defined concretely with real domains. The Logic Tensor Networks (LTN) framework has applied Real Logic to many important AI tasks through querying, learning, and reasoning. Motivated by real-life relational database applications, we study adding aggregate functions, such as averaging elements of a relation table, to Real Logic. The key contribution of this paper is the formalization of such functions within Real Logic. This extension is straightforward and fits coherently in the end-to-end differentiable language that Real Logic is. We illustrate it on FooDB, a food chemistry database, and query foods and their nutrients. The resulting framework combines strengths of descriptive statistics modeled by fuzzy predicates, FOL to write complex queries and formulas, and SQL-like expressiveness to aggregate insights from data tables.

Keywords
Many-Valued Logic, Relational Database, Querying, Aggregation, Neurosymbolic AI

1. Introduction

Real Logic is introduced in Logic Tensor Networks (LTN) [1, 2], a neurosymbolic framework that supports querying, learning, and reasoning with rich data and abstract knowledge about the world. Real Logic is a first-order language with concrete semantics such that every expression has an interpretation in real and continuous domains. In particular, LTN converts Real Logic formulas, e.g. $\forall x \exists y(\mathsf{is\_friend}(x, y) \land \mathsf{Italian}(y))$, into TensorFlow computational graphs. Recent works have applied Real Logic to many of the most important AI tasks, including supervised classification, data clustering, semi-supervised learning, embedding learning, reasoning or query answering [1, 3, 4, 5, 6].

Querying, however, remains limited by syntactic limitations of First-Order Logic (FOL). Despite being a powerful query language on knowledge graphs, FOL cannot express many useful queries written in the most popular database applications such as SQL. These databases organize relations in tables defined with rows and columns. We commonly apply *aggregate functions* to extract insight features over sets of row values. For example, the following query proposed in [7] over a relation `R1` with attributes ”employee” and ”department”, and a relation `R2` with attributes ”employee” and ”salary”, is trivial in SQL:

NESY'20/21 : 15th International Workshop on Neural-Symbolic Learning and Reasoning, October 25–27, 2021, IJCLR.
badreddine.samy@gmail.com (S. Badreddine); michael.spranger@gmail.com (M. Spranger)
0000-0003-1624-9188 (S. Badreddine); 0000-0001-9443-7008 (M. Spranger)

```
SELECT R1.Dept, AVG(R2.Salary)
FROM R1, R2
WHERE R1.Employee = R2.Employee
GROUPBY R1.Dept
HAVING SUM(R2.Salary) > 1000000
```

Comparable queries are not adequately captured by traditional logical languages, as their semantics are not naturally equipped with aggregate functions such as `SUM` or `AVG`.

In this paper, we study adding aggregate functions to Real Logic. Real Logic can naturally support aggregate functions for two reasons: 1) individuals and domains are grounded with real features, 2) variables are interpreted as finite sequences of individuals. These sequences and real features are reminiscent of the rows in SQL-like tables and constitute ranges over which the aggregate functions can straightforwardly operate.

The paper starts with an overview of Real Logic in Section 2. In Section 3, we formalize aggregate functions as mathematical operators and as part of a Real Logic signature, the latter being the key contribution of this paper. In Section 4, we illustrate the functions with simple queries on FooDB, a food chemistry dataset.

In [7], authors dress a comprehensive summary for adding aggregation functions in the context of logics. The authors study aggregates for a 2-sorted logics. The first sort is interpreted as a usual logical domain, with ”abstract” semantics and set relationships. The second denotes the rationals $\mathbb{Q}$, on which the aggregate functions operate. In contrast, the concrete semantics of Real Logic does not require a distinction between such sorts. Aggregate operators have also been studied in Prolog and Datalog-like languages [8, 9, 10].

Perhaps the closest fit to this paper are fuzzy querying systems for relational databases [11, 12, 13]. Real Logic similarly grounds the semantics of formulas with fuzzy truth-values. Let x be a variable that denotes people. To represent $\mathsf{is\_tall}(x)$, rather than using a crisp boolean condition $\mathsf{height}(x) > 180$, one can use a continuous truth-value in $[0, 1]$ whose scaling depends on descriptive statistics of the height in the population. Notice that these truth values do not denote probabilities like in probabilistic databases [14]. Fuzzy querying systems allow making flexible queries about traditional (crisp) or fuzzy data. Incorporating aggregate functions in Real Logic gives such expressive power to LTN, and integrates coherently with its other aspects.

2. Real Logic

2.1. Basics and Notations

Real Logic [1, 2] is defined on a first-order language $\mathcal{L}$ with a signature that contains a set of *constant symbols* (individuals), a set of *functional symbols*, a set of *relational symbols* (predicates), and a set of *variable symbols*. It allows to specify relational knowledge about the world, e.g. the atomic formula $\mathsf{is\_friend}(\mathsf{bob}, \mathsf{alice})$ states that alice is a friend of bob and the formula $\forall u \forall v(\mathsf{is\_friend}(u, v) \to \mathsf{is\_friend}(v, u))$ states that the relation $\mathsf{is\_friend}$ is symmetric, where u, v are variables, $\mathsf{bob}, \mathsf{alice}$ are constants and $\mathsf{is\_friend}$ is a predicate.

Contrarily to the abstract semantics of FOL, the elements of the signature are grounded to data, mathematical functions, and neural computational graphs. The connectives are grounded to fuzzy semantics. To emphasize that symbols are grounded using real-valued features, we use the term *grounding*, denoted by $\mathcal{G}$. Individuals are grounded as tensors [1] of real features. Functions are grounded as real functions, and predicates as real functions that specifically project onto a value in the interval $[0,1]$. Consequently, $\mathcal{L}$-formulas are not assigned to boolean true or false values, but instead are grounded to a truth-value in the continous interval $[0,1]$. For example, in the expression is_friend(bob, alice), $\mathcal{G}$(bob) and $\mathcal{G}$(alice) can be vector embeddings in $\mathbb{R}^m$. The friendship relationship can be approximated by a cosine similarity function $\mathcal{G}(\text{is\_friend}) : (\mathbf{x},\mathbf{y}) \mapsto \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$, or a more complex function, depending on the application.

Individuals are typed and belong to *domains*. Logical domains are grounded to real domains that shape the individuals. For example, alice, bob are individuals of the domain people, and $\mathcal{G}(\text{people}) = \mathbb{R}^m$. $\mathcal{L}$-functions and predicates are defined with an arity and a domain for each argument. For functions, we also define an output domain. For example, the function best_friend(x) takes an argument from people and returns an individual from people.

A variable x is grounded as an explicit *sequence* of n_x individuals from a domain, with $0 < n_x < \infty$. [2] This differs from a FOL variable, which is a placeholder for any individual from the universe. Consequently, a term $t(x)$ or a formula $P(x)$ constructed recursively with a free variable x, will be grounded to a sequence of n_x values too.

Terms and formulas are constructed recursively in the usual way, given that functional and predicate symbols are applied to an appropriate number of terms with appropriate domains.

Complex formulas are constructed using the usual logical connectives, $\wedge$, $\vee$, $\rightarrow$, $\neg$, and the quantifiers $\forall$, $\exists$. Real Logic interprets connectives using fuzzy semantics, and quantifiers using special aggregator functions. In particular, [1] recommends to use the *product configuration*, which has been shown in [15] to be better suited for gradient-based optimization (see Section 2.2), with minor modifications for numerical stability. In the product configuration, conjunctions ($\wedge$) are grounded with the product t-norm $\mathrm{T}_{\mathrm{prod}}$, disjunctions ($\vee$) with the product t-conorm $\mathrm{S}_{\mathrm{prod}}$, implications ($\rightarrow$) with the Reichenbach implication I_{R} and negations ($\neg$) with the standard fuzzy negation N_{S}. Existential quantifiers ($\exists$) are grounded with the generalized mean M_p, with $p \geq 1$, which also corresponds to the p-norm of the inputs in this particular setting. Universal quantifiers ($\forall$) are grounded with the generalized mean w.r.t. the error values ME_p, with $p \geq 1$. $\lim_{p\to\infty} \mathrm{M}_p(u_1,\dots,u_n) = \max\{u_1,\dots,u_n\}$ and $\lim_{p\to\infty} \mathrm{ME}_p(u_1,\dots,u_n) = \min\{u_1,\dots,u_n\}$.

$$\mathrm{T}_{\mathrm{prod}}(u,v) = uv \tag{1}$$

$$\mathrm{S}_{\mathrm{prod}}(u,v) = u + v - uv \tag{2}$$

$$\mathrm{I}_{\mathrm{R}}(u,v) = 1 - u + uv \tag{3}$$

$$\mathrm{N}_{\mathrm{S}}(u) = 1 - u \tag{4}$$

$$\mathrm{M}_p(u_1,\dots,u_n) = \left(\frac{1}{n}\sum_{i=1}^{n} u_i^p\right)^{\frac{1}{p}} \tag{5}$$

[1]Here, a "tensor" denotes a multidimensional numerical array, as commonly used in the ML community. This is not to confuse with how tensors are defined in mathematics or physics.

[2]Notice that, because a variable x is grounded as a sequence instead of a set, the same value can occur multiple times in x. However, the order of the values in the sequence does not matter.

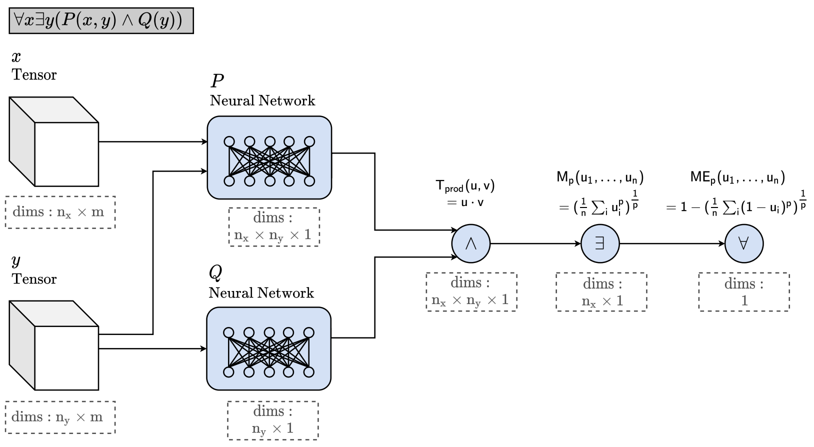
$\forall x \exists y (P(x,y) \wedge Q(y))$
x
Tensor
dims : $n_x \times m$
y
Tensor
dims : $n_y \times m$
P
Neural Network
dims : $n_x \times n_y \times 1$
Q
Neural Network
dims : $n_y \times 1$
$T_{prod}(u,v) = u \cdot v$
$\wedge$
dims : $n_x \times n_y \times 1$
$M_p(u_1,\ldots,u_n) = (\frac{1}{n}\sum_i u_i^p)^{\frac{1}{p}}$
$\exists$
dims : $n_x \times 1$
$ME_p(u_1,\ldots,u_n) = 1 - (\frac{1}{n}\sum_i (1-u_i)^p)^{\frac{1}{p}}$
$\forall$
dims : 1

Figure 1: A Real Logic sentence grounded to a computational graph.

$$\mathrm{ME}_p(u_1,\ldots,u_n) = 1 - \left(\frac{1}{n}\sum_{i=1}^{n}(1-u_i)^p\right)^{\frac{1}{p}} \tag{6}$$

In [1], authors also introduce diagonal quantification and guarded quantifiers. Guarded quantifiers restrict the quantification over the individuals with groundings that satisfy some boolean condition. For example, in:

$$\forall x (\exists y : \mathrm{age}(x) > \mathrm{age}(y) \, (\mathrm{is\_parent}(x,y))) \tag{7}$$

the boolean condition $\mathrm{age}(x) > \mathrm{age}(y)$ restricts the quantification over y.

We do not use diagonal quantification in this paper, but the interested reader will find more information in [1]. Intuitively, it is a special form of quantification that performs a Python's `zip` over two or more variables before applying the aggregator.[3]

2.2. Knowledge, Querying and Optimization

Knowledge is represented not only by logical formulas but also by the grounding $\mathcal{G}$ which explicitly connects symbols occurring in formulas and what concretely holds in the domain. For example, the grounding of alice in real features that describe her height, age, etc., is knowledge. This grounding definition can be parametric if $\mathcal{G}(\mathrm{alice})$ is a set of embedded features to learn. In that case, we write $\mathcal{G}(\mathrm{alice} \mid \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the set of trainable parameters. Similarly, $\mathrm{is\_friend}(\mathrm{alice}, \mathrm{bob})$ can be grounded using an explicit cosine similarity function or can be grounded using a trainable neural network. Consequently, learning knowledge in Real Logic is not limited to inferring new formulas, but also concerns the parameters of a grounding.

We define the *satisfaction* of a formula as its grounding. *Querying* a formula ϕ is therefore equivalent to evaluating $\mathcal{G}(\phi)$, which returns a truth-value in $[0,1]$. One can also query a term t by evaluating $\mathcal{G}(t)$, which returns an individual (or a sequence of individuals, if the term has free variables) from some real domain.

[3] https://docs.python.org/3/library/functions.html#zip

Let $\mathcal{K}$ define a collection of formulas, as represented in traditional logical knowledge bases. The satisfaction of $\mathcal{K}$ under a parametric grounding $\mathcal{G}(\cdot \mid \boldsymbol{\theta})$ is defined as the aggregation of the satisfactions of each $\phi \in \mathcal{K}$. The result depends on the choice of aggregate operator, denoted by SatAgg. Often, SatAgg is defined as ME_p, also used to ground the $\forall$ operator. In LTN, one can search the optimal grounding, that is the optimal set of parameters, to satisfy a theory according to the following objective:

$$\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} \underset{\phi \in \mathcal{K}}{\mathrm{SatAgg}}\, \mathcal{G}(\phi \mid \boldsymbol{\theta}) \tag{8}$$

Because Real Logic grounds expressions in real and continuous domains, LTN attaches gradients to every sub-expression and consequently learns through gradient-descent optimization.

3. Aggregate Functions

3.1. Definition

An aggregate function is a mathematical operation involving a range of values that results in a single representative value. Existing literature usually defines aggregate functions on (sub)sets of real or rational numbers [9, 16, 7]. We adapt the definition for Real Logic, where logical domains can be grounded to many types of real domains. Let $\mathbb{I}_{\text{in}}$ and $\mathbb{I}_{\text{out}}$ denote such domains.

Definition 1. An *aggregate function* is a collection of functions $\mathscr{A} = \{\mathrm{A}^{(0)}, \mathrm{A}^{(1)}, \dots, \mathrm{A}^{(\infty)}\}$, where $\mathrm{A}^{(n)} : (\mathbb{I}_{\text{in}})^n \to \mathbb{I}_{\text{out}}$. Each n-ary function $\mathrm{A}^{(n)}$ describes how the aggregator behaves on an n-element input. In particular, $\mathrm{A}^{(0)}$ is the constant produced on an empty set of inputs, and $\mathrm{A}^{(\infty)}$ defines the behavior when applied to an infinite set of inputs. When no confusion can arise, we simply write A instead of $\mathrm{A}^{(n)}$.

In many cases, $\mathbb{I}_{\text{in}} = \mathbb{I}_{\text{out}}$. For example, if $\mathbb{I}_{\text{in}} = \mathbb{R}$, then mean, min, max are all operators that return objects in the same domain. On the other hand, if $\mathbb{I}_{\text{in}} = \mathbb{R}^3$, a function count will output results in $\mathbb{R}$ (more precisely, $\mathbb{N}$) and therefore $\mathbb{I}_{\text{in}} \neq \mathbb{I}_{\text{out}}$.

Notice that, for an application in Real Logic, the behavior of $\mathrm{A}^{(\infty)}$ does not have to be specified as variables cannot be sequences of infinite length for practical reasons.

Notice also that the aggregators used to approximate $\forall, \exists$ are a special case of aggregate functions with $\mathbb{I}_{\text{in}} = \mathbb{I}_{\text{out}} = [0, 1]$.

3.2. As Real Logic Functions

Incorporate such aggregate functions into $\mathcal{L}$ is straightforward. The concrete, real domains of Real Logic can be mapped to the ones in Definition 1, Additionally, $\mathcal{L}$-variables are sequences, and any term recursively constructed from a variable is a sequence too. Aggregate functions can operate over these sequences.

Definition 2. An *$\mathcal{L}$-aggregate function* symbol A with input domain D_{in} and output domain D_{out}, is grounded using an aggregate function A (Definition 1) such that $\mathcal{G}(D_{\text{in}}) = \mathbb{I}_{\text{in}}$ and

$\mathcal{G}(D_{\text{out}}) = \mathbb{I}_{\text{out}}$. Let x be a variable symbol, and $|\mathcal{G}(x)| = n$, that is x is grounded as a sequence of n individuals. For any term $t(x)$ with free variable x, such that $\mathcal{G}(t(x)) \in \mathcal{G}(D_{\text{in}})$, we have:

$$\mathcal{G}(Ax\, t(x)) = \mathrm{A}_{i=1,\dots,n}(\mathcal{G}(t(x))_{(i)}) \tag{9}$$

where $\mathcal{G}(t(x))_{(i)}$ is the evaluation of $\mathcal{G}(t(x))$ using the i-th individual of x.

The syntax is similar to that of quantifiers, e.g. $\forall x\, \phi(x)$. Here, $\forall$ is replaced by an $\mathcal{L}$-aggregate function symbol A, and the formula $\phi(x)$ is replaced by a term $t(x)$. For ease of notation, and if no confusion can arise, we simply write A to denote both the $\mathcal{L}$-symbol A and its grounding A.

Example 1. Let x be a variable that denotes people. In particular, let $\mathcal{G}(x) = \langle \text{alice}, \text{bob}, \text{charlie} \rangle$ denote a sequence of three individuals. We can query the average height of x, using an aggregate function mean, with:

$$\mathcal{G}(\text{mean}_x \text{height}(x)) = \text{mean}_{i=1}^{3}\, \mathcal{G}(\text{height}(x))_{(i)} \tag{10}$$

$$= \frac{\mathcal{G}(\text{height}(\text{alice})) + \mathcal{G}(\text{height}(\text{bob})) + \mathcal{G}(\text{height}(\text{charlie}))}{3} \tag{11}$$

The output is a new term, embedding the average height in the population, which can be used as an input to other formulas.

Notice that we can combine aggregate functions with guarded quantification (see Equation 7) by re-using the same syntax and functionality.

Example 2. Following the Example in 1, let us consider that alice and bob are Italian, but charlie is not. Let Italian(x) be a boolean predicate that returns either 1 (true) or 0 (false). One can query the average height of Italians with:

$$\mathcal{G}(\text{mean}_{x\,:\,Italian(x)} \text{height}(x)) = \text{mean}^{3}_{\substack{i\,=\,1 \\ s.t.\ \mathcal{G}(\text{Italian}(x))_{(i)}\,=\,1}}\, \mathcal{G}(\text{height}(x))_{(i)} \tag{12}$$

$$= \frac{\mathcal{G}(\text{height}(\text{alice})) + \mathcal{G}(\text{height}(\text{bob}))}{2} \tag{13}$$

Most of the common aggregate functions (except count) are differentiable. Consequently, they still integrate nicely with the end-to-end differentiability of Real Logic and LTN. In the following experiments, we only explore querying with aggregate functions. However, aggregate functions could also be integrated with optimization in LTN.

4. Experiments

4.1. FooDB Dataset

FooDB [4] is a database that provides information on food chemistry and food constituents, including macronutrients, micronutrients, and many of the constituents that give foods their

[4]www.foodb.ca

Table 1
Excerpt of macronutrient data gathered from FooDB. Proteins, fats, carbohydrates and fibers numbers are listed in g/100g. Energy is in kcal/100g. In total, there is macronutrient data for 797 ingredients.

food name	food group	kcal	proteins	fats	carbs	fibers
Savoy cabbage	Vegetables	27.0	2.0	0.1	6.1	3.1
Kiwi	Fruits	60.5	1.1	0.6	14.2	2.6
Garlic	Herbs and Spices	245.0	11.5	0.6	52.9	5.8
Cashew nut	Nuts	579.8	16.6	47.2	30.4	2.9
Cucumber	Gourds	35.5	0.5	0.2	8.3	1.0
Pineapple	Fruits	65.8	0.5	0.1	16.4	0.8
Dill	Herbs and Spices	196.4	12.6	5.1	36.2	10.2
Peanut	Nuts	580.2	24.6	49.8	18.8	7.7
Asparagus	Vegetables	19.9	2.3	0.3	3.4	1.7
Oat	Cereals and cereal products	385.3	11.7	16.6	53.8	8.7

flavor, color, taste, texture, and aroma. The data is compiled from existing literature and databases, in order to provide the most comprehensive food chemical database.

We focus on querying macronutrient data from FooDB. In total, FooDB lists macronutrient information for 797 ingredients. We average the information sourced from the USDA[5] and from the Technical University of Denmark[6]. We also use some of the simple ontologies about food groups defined in FooDB.

4.2. Grounding

Ingredients are grounded using their macronutrient attributes and their food groups, the latter being encoded as integer labels. Functional symbols are mostly getters of these attributes. We use explicit vocabulary for most of the terms. For example, let x be a variable for ingredients. The function symbol prot(x) gives the protein level of ingredients, kcal(x) gives their energy level, and so forth.

For the aggregate functions, we use traditional operators such as mean, max, or std. Predicates such as is_cereal(x) assert if the ingredient belongs to the corresponding food group. In this experiment, such ontology predicates evaluate to 0 (false) or 1 (true).

For grounding the connectives of the language, we use the product configuration presented in Section 2. ME_p ($\forall$) is implemented with $p = 20$, making it relatively close to the min operator.

4.2.1. Predicate is_high

We use two fuzzy predicates is_high and is_low to describe values on a scale. Let x be a real value sampled from a distribution X. To define how high is x compared to the other values in the distribution, we use the concept of Cumulative Distribution Functions (CDF).

$$\text{is\_high}_X(x) = P_X(X \leq x) \tag{14}$$

[5]U.S. Department of Agriculture, Agricultural Research Service. FoodData Central, 2019. https://fdc.nal.usda.gov/
[6]Food data (frida.fooddata.dk), version 4, 2019, National Food Institute, Technical University of Denmark

Intuitively, we use the probability that another sample will take a value less than or equal to x as a truth degree that defines how high the value of x is.

In this experiment, we assume that logistic distributions approximately fit the nutrient data.[7] Let μ be the mean value of the distribution and σ be its standard deviation. Using the formula of the CDF of a logistic distribution, the model for is_high computes:

$$\mathsf{is\_high}(x, \mu, \sigma) = \mathrm{S}(\frac{x - \mu}{\sigma}) \tag{15}$$

where $\mathrm{S}(y) = \frac{1}{1+e^{-y}}$ is a sigmoid function.

is_low is computed by querying ¬is_high. Aggregate functions are used to compute the mean μ and standard deviation σ in Equation 15.

4.3. Queries

We illustrate using queries about food nutrients in ingredients, that mix descriptive statistics and complex logical queries. In particular, we ask:

Q1. What cereal product is not high in calories?

Q2. What cereal product is high in protein?

Q3. What cereal product is high in protein and not high in calories?

Q4. For every ingredient, being high in protein implies being high in calories?

Q5. For every ingredient, being high in fats implies being high in calories?

The detailed formulas are in Table 2. We evaluate if a cereal is high in, for example, calories, using other cereals as a scale (notice the guarded quantifier with condition is_cereal(x)). When querying about free variables (Q1, Q2, Q3), we present the top 3 ranking results.

Using a similar approach, we could integrate information on price, seasonability, etc., to rank alternatives on continuous scales. Virtually, there is no limitation to the complexity of queries.

5. Conclusion

In this paper, primarily motived by commercial database applications, we studied adding aggregate functions to Real Logic. As a querying system, it combines strengths of 1) descriptive statistics, modeled through fuzzy predicates, 2) FOL syntax to write complex queries, and 3) SQL-like expressiveness to aggregate and collect insights from data tables.

As many common aggregate functions are differentiable, Real Logic with aggregate functions can still be end-to-end differentiable. If the knowledge graph entries are associated with embeddings, one can use continuous optimization techniques to efficiently answer queries [17, 18, 19]. In particular, in [17], authors propose an approach based on gradient-descent that uses a logical language akin to Real Logic, where fuzzy semantics approximate connectives. The same approach could incorporate aggregate functions and support a wider range of queries.

[7] In real-life applications, adequate statistical tests should validate the choice of distribution. In this paper, we do not conduct such tests as our results are illustrative and not part of our contribution.

Table 2
Querying about calorific and protein levels of cereal products in FooDB.

id	Query	Top-3 Results	Scores
Q1	$y?\ \ \text{is\_cereal}(y) \land \neg\text{is\_high}(\text{kcal}(y), \text{mean}_{x:\text{is\_cereal}(x)}\text{kcal}(x), \text{std}_{x:\text{is\_cereal}(x)}\text{kcal}(x))$	Bulgur Wild rice Corn	0.84 0.79 0.79
Q2	$y?\ \ \text{is\_cereal}(y) \land \text{is\_high}(\text{prot}(y), \text{mean}_{x:\text{is\_cereal}(x)}\text{prot}(x), \text{std}_{x:\text{is\_cereal}(x)}\text{prot}(x))$	Multigrain bread Triticale Potato bread	0.92 0.88 0.84
Q3	$y?\ \ \text{Q1}(y) \land \text{Q2}(y)$	Multigrain bread Potato bread Oriental wheat	0.56 0.55 0.50
Q4	$\forall y\, (\text{is\_high}(\text{prot}(y), \text{mean}_x\text{prot}(x), \text{std}_x\text{prot}(x)) \rightarrow \text{is\_high}(\text{kcal}(y), \text{mean}_x\text{kcal}(x), \text{std}_x\text{kcal}(x)))$		0.56
Q5	$\forall y\, (\text{is\_high}(\text{fats}(y), \text{mean}_x\text{fats}(x), \text{std}_x\text{fats}(x)) \rightarrow \text{is\_high}(\text{kcal}(y), \text{mean}_x\text{kcal}(x), \text{std}_x\text{kcal}(x)))$		0.74

References

[1] S. Badreddine, A. d. Garcez, L. Serafini, M. Spranger, Logic Tensor Networks, arXiv:2012.13635 [cs] (2021). URL: http://arxiv.org/abs/2012.13635, arXiv: 2012.13635.

[2] L. Serafini, A. d. Garcez, Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge, arXiv:1606.04422 [cs] (2016). URL: http://arxiv.org/abs/1606.04422, arXiv: 1606.04422.

[3] F. Bianchi, M. Palmonari, P. Hitzler, L. Serafini, Complementing Logical Reasoning with Sub-symbolic Commonsense, 2019, pp. 161–170. doi:10.1007/978-3-030-31095-0_11.

[4] F. Bianchi, P. Hitzler, On the Capabilities of Logic Tensor Networks for Deductive Reasoning, in: AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering, 2019.

[5] I. Donadello, L. Serafini, Compensating Supervision Incompleteness with Prior Knowledge in Semantic Image Interpretation, arXiv:1910.00462 [cs, stat] (2019). URL: http://arxiv.org/abs/1910.00462, arXiv: 1910.00462.

[6] E. van Krieken, E. Acar, F. van Harmelen, Semi-Supervised Learning using Differentiable Reasoning, arXiv:1908.04700 [cs] (2019). URL: http://arxiv.org/abs/1908.04700, arXiv: 1908.04700.

[7] L. Hella, L. Libkin, J. Nurmonen, L. Wong, Logics with aggregate operators, Journal of the ACM 48 (2001) 880–907. URL: https://doi.org/10.1145/502090.502100. doi:10.1145/502090.502100.

[8] W. Faber, G. Pfeifer, N. Leone, T. Dell'Armi, G. Ielpa, Design and Implementation of Aggregate Functions in the DLV System, arXiv:0802.3137 [cs] (2008). URL: http://arxiv.org/abs/0802.3137, arXiv: 0802.3137.

[9] M. Grabisch, J.-L. Marichal, R. Mesiar, E. Pap, Aggregation functions: Means, Information Sciences 181 (2011) 1–22. URL: https://hal.archives-ouvertes.fr/hal-00539028. doi:10.1016/

j.ins.2010.08.043, publisher: Elsevier.

[10] A. Van Gelder, The well-founded semantics of aggregation, in: Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS '92, Association for Computing Machinery, New York, NY, USA, 1992, pp. 127–138. URL: https://doi.org/10.1145/137097.137854. doi:10.1145/137097.137854.

[11] S.-M. Chen, W.-T. Jong, Fuzzy query translation for relational database systems, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 27 (1997) 714–721. doi:10.1109/3477.604117, conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics).

[12] J. Galindo, A. Urrutia, M. Piattini, Fuzzy Databases: Modeling, Design and Implementation, IGI Global, 2006. URL: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59140-324-1. doi:10.4018/978-1-59140-324-1.

[13] R. Mama, M. Machkour, A study of fuzzy query systems for relational databases, in: Proceedings of the 4th International Conference on Smart City Applications, SCA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–5. URL: https://doi.org/10.1145/3368756.3369105. doi:10.1145/3368756.3369105.

[14] D. Suciu, D. Olteanu, C. Ré, C. Koch, Probabilistic Databases, Synthesis Lectures on Data Management 3 (2011) 1–180. URL: https://www.morganclaypool.com/doi/abs/10.2200/S00362ED1V01Y201105DTM016. doi:10.2200/S00362ED1V01Y201105DTM016, publisher: Morgan & Claypool Publishers.

[15] E. van Krieken, E. Acar, F. van Harmelen, Analyzing Differentiable Fuzzy Logic Operators, arXiv:2002.06100 [cs] (2020). URL: http://arxiv.org/abs/2002.06100, arXiv: 2002.06100.

[16] E. Grädel, Y. Gurevich, Metafinite Model Theory, Information and Computation 140 (1998) 26–81. URL: https://www.sciencedirect.com/science/article/pii/S0890540197926754. doi:10.1006/inco.1997.2675.

[17] E. Arakelyan, D. Daza, P. Minervini, M. Cochez, Complex Query Answering with Neural Link Predictors, arXiv:2011.03459 [cs] (2021). URL: http://arxiv.org/abs/2011.03459, arXiv: 2011.03459.

[18] T. Friedman, G. V. d. Broeck, Symbolic Querying of Vector Spaces: Probabilistic Databases Meets Relational Embeddings, arXiv:2002.10029 [cs] (2020). URL: http://arxiv.org/abs/2002.10029, arXiv: 2002.10029.

[19] M. Wang, R. Wang, J. Liu, Y. Chen, L. Zhang, G. Qi, Towards Empty Answers in SPARQL: Approximating Querying with RDF Embedding, in: D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, E. Simperl (Eds.), The Semantic Web – ISWC 2018, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 513–529. doi:10.1007/978-3-030-00671-6_30.

A. Computational Efficiency

As LTN implements Real Logic using Tensorflow computational graphs[8], it inherits from Tensorflow's built-in optimizations. For example, on a local portable machine, we measure the computation time of two queries averaged on 10000 runs:

Q6. $\mathsf{is\_high}(\mathsf{kcal}(\mathsf{orange}), \mathsf{mean}_x \mathsf{kcal}(x), \mathsf{std}_x \mathsf{kcal}(x))$ which takes 2.58 ms to execute,

Q7. $\mathsf{is\_high}(\mathsf{kcal}(y), \mathsf{mean}_x \mathsf{kcal}(x), \mathsf{std}_x \mathsf{kcal}(x))$ which takes 3.57ms to execute.

Q6 returns a single result for orange, whereas Q7 returns results for each ingredient, that is 797 results. One could expect Q7 to be approximately 797 slower than Q6 if LTN would need to recompute the terms $\mathsf{mean}_x \mathsf{kcal}(x)$ and $\mathsf{std}_x \mathsf{kcal}(x)$ as many times. Instead, we find that Q6 is executed in 2.58ms and Q7 is executed in 3.57ms. Using Tensorflow built-in optimization, LTN can efficiently re-use common parts of the computational graph for optimal complexity.

[8] https://www.tensorflow.org/