Overview

Embedding-based approaches:

- LTNs: Logic Tensor Networks
- RRNs: Recursive Reasoning Networks (a.k.a. Deep Ontological Networks)
- NeuPSL: Neural Probabilistic Soft Logic

LTNs: Introduction

- Logic Tensor Networks (LTN) is a recently-developed NSR framework based on **fuzzy differentiable logic**.
- Supports tasks based on manipulating **data** and **knowledge**.
- It has been shown to effectively tackle many tasks that are central to intelligent systems:
  - multi-label classification,
  - relational learning,
  - data clustering,
  - semi-supervised learning,
  - regression,
  - embedding learning, and
  - query answering under uncertainty.

LTNs: Overview

- Real Logic
  - Tensors
  - Groundings
  - Fuzzy operators
- Logic Tensor Networks
  - From Real Logic to LTN
  - LTN Tasks
  - Use Cases
- Bonus: Differentiable Fuzzy Logic

Real Logic: Introduction

LTN uses an infinitely-valued fuzzy logical language called **Real Logic** as the underlying formalism:

- Domains are interpreted concretely by **tensors** in the Real field.
- Recall that tensors are **algebraic objects** that include:
  - Scalars: 0-dimensional,
  - Vectors: 1-dimensional,
  - Matrices: 2-dimensional,
  - as well as higher-dimension structures.
- To emphasize this, the authors use the term “**grounding**”, denoted with the letter ⍰, instead of “interpretation” (the usual name for this concept in logic).

Source: https://github.com/logictensornetworks/

Real Logic: Introduction (cont.)

LTN uses an infinitely-valued fuzzy logical language called **Real Logic** as the underlying formalism:

- Grounding ⍰ maps:
  - Terms to **tensors** of real numbers, and
  - **Formulas** to **real numbers** in the interval [0,1].
- We commonly use "tensor" to abbreviate the expression "tensor in the Real field".
- As usual, the language allows for logical **connectives** and **quantifiers**.

Source: https://github.com/logictensornetworks/

Real Logic: Language

Constants:

- Denote **individuals** from a space of tensors $\bigcup_{n_1 \dots n_d \in \mathbb{N}^*} \mathbb{R}^{n_1 \times \dots \times n_d}$ (from now on, we write “tensor of any rank” to denote this expression).
- The individual can be **pre-defined** (data point) or **learnable** (embedding).
- **Intuition**: Each dimension corresponds to a **feature**, and the number corresponds to the value of that feature for that individual.

Variables denote sequences of individuals:

- Sequences represent the **possible values** that the variable can take.
- They can contain more than one instance of the same value.

Source: https://github.com/logictensornetworks/

Real Logic: Language (cont.)

Functions:

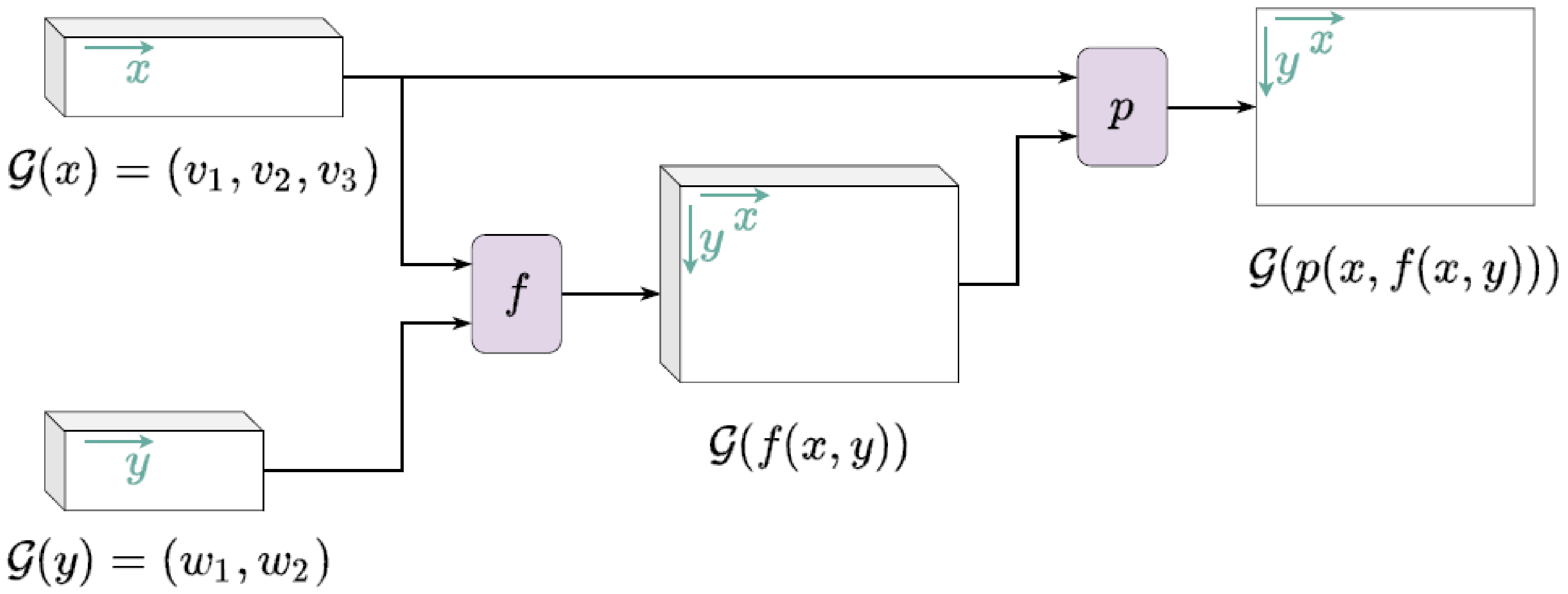
- Can be any mathematical function (either pre-defined or learnable).
- Examples of functions are distance functions, regressors, etc.

Predicates:

- Represented as mathematical functions that map an n-ary domain of individuals to a real in [0,1], interpreted as a **truth degree**.
- Examples of predicates: similarity measures, classifiers, etc.

Source: https://github.com/logictensornetworks/

Real Logic: Grounding of Functions and Predicates



x
$\mathcal{G}(x) = (v_1, v_2, v_3)$
y
$\mathcal{G}(y) = (w_1, w_2)$
f
y
x
$\mathcal{G}(f(x,y))$
p
y
x
$\mathcal{G}(p(x, f(x,y)))$

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Real Logic: Language (cont.)

Connectives are modeled using fuzzy semantics:

- Conjunction (∧): t-norm T
- Disjunction (∨): t-conorm S
- Implication (⍰): fuzzy implication I
- Negation (¬): fuzzy negation N

$\text{FuzzyOp} \in \{T, S, I, N\}$

We will come back to these operators later when discussing differentiability.

Quantifiers are defined using **aggregators** (symmetric and continuous operators)

- Existential (∃): Generalization of existential quantification in FOL
- Universal (∀): Generalization of universal quantification in FOL

Source: https://github.com/logictensornetworks/

Real Logic: Common Fuzzy Operators

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

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

Source: https://github.com/logictensornetworks/

Real Logic: Common Fuzzy Operators

Some possible **aggregators** for fuzzy **universal** quantification:

$$A_{T_M}(x_1,\ldots,x_n) = \min(x_1,\ldots,x_n)$$ (Minimum)

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

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

Some possible **aggregators** for fuzzy **existential** quantification:

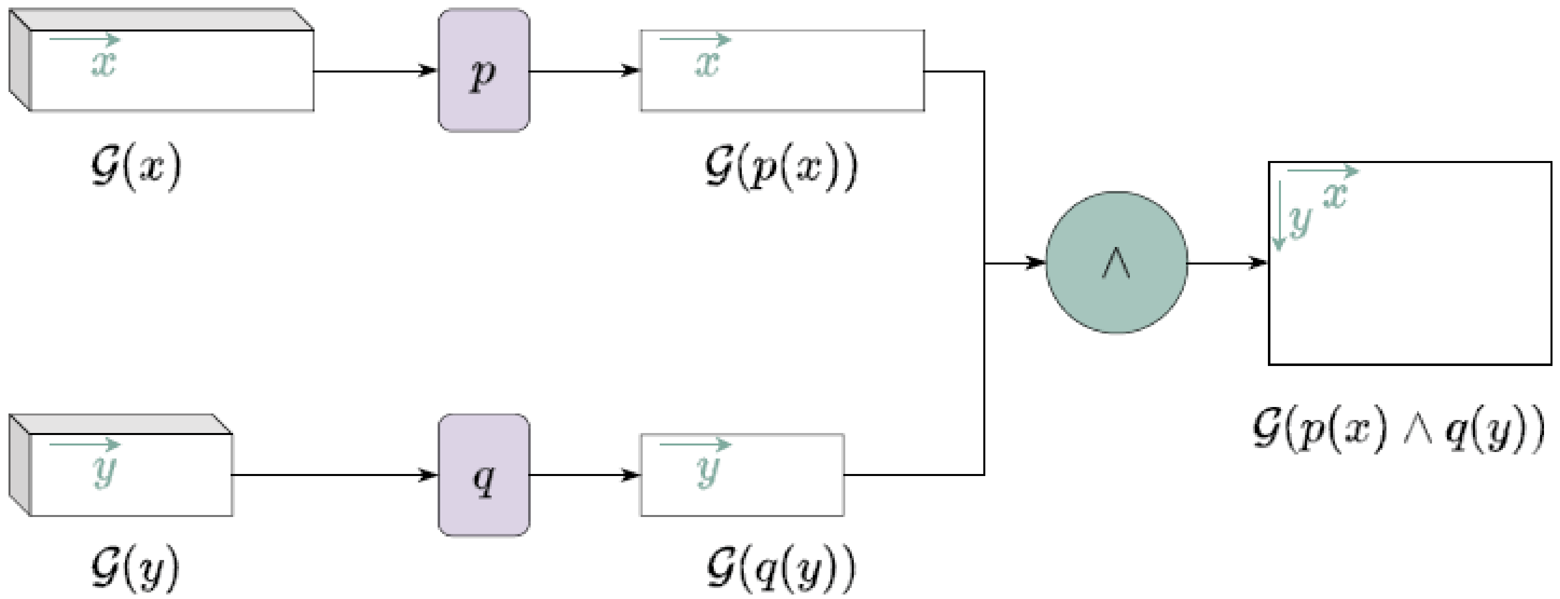
$$A_{S_M}(x_1,\ldots,x_n) = \max(x_1,\ldots,x_n)$$ (Maximum)

$$A_{S_P}(x_1,\ldots,x_n) = 1 - \prod_{i=1}^{n}(1 - x_i)$$ (Probabilistic Sum)

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

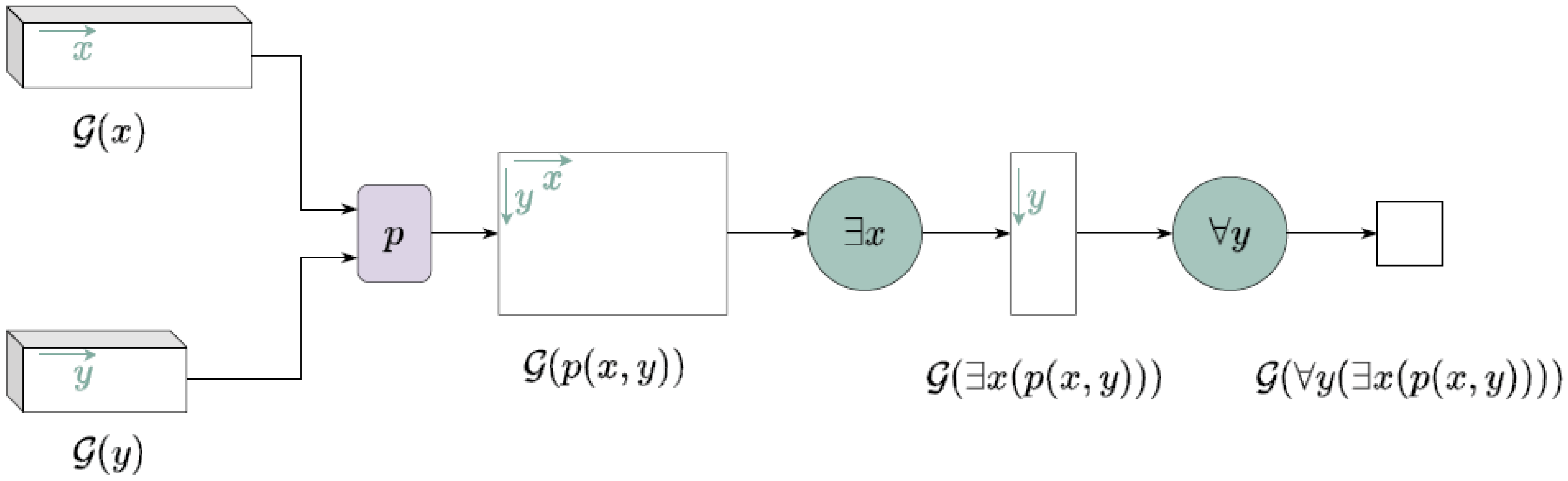
Source: https://github.com/logictensornetworks/

Real Logic: Conjunction



Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Real Logic: Quantification



Note: Depending on the properties that the aggregation operators enjoy, the semantics may or may not adequately generalize that of FOL.

(For instance, commutativity of quantifiers may not be guaranteed.)

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Real Logic: Language (cont.)

Diagonal quantification:

- Diag(x_1, ..., x_h) quantifies over **specific tuples** s.t. the *i*-th tuple contains the *i*-th instance of each of the variables in the argument of Diag.
- Assumes that all variables in the argument are grounded onto sequences with the same number of instances.

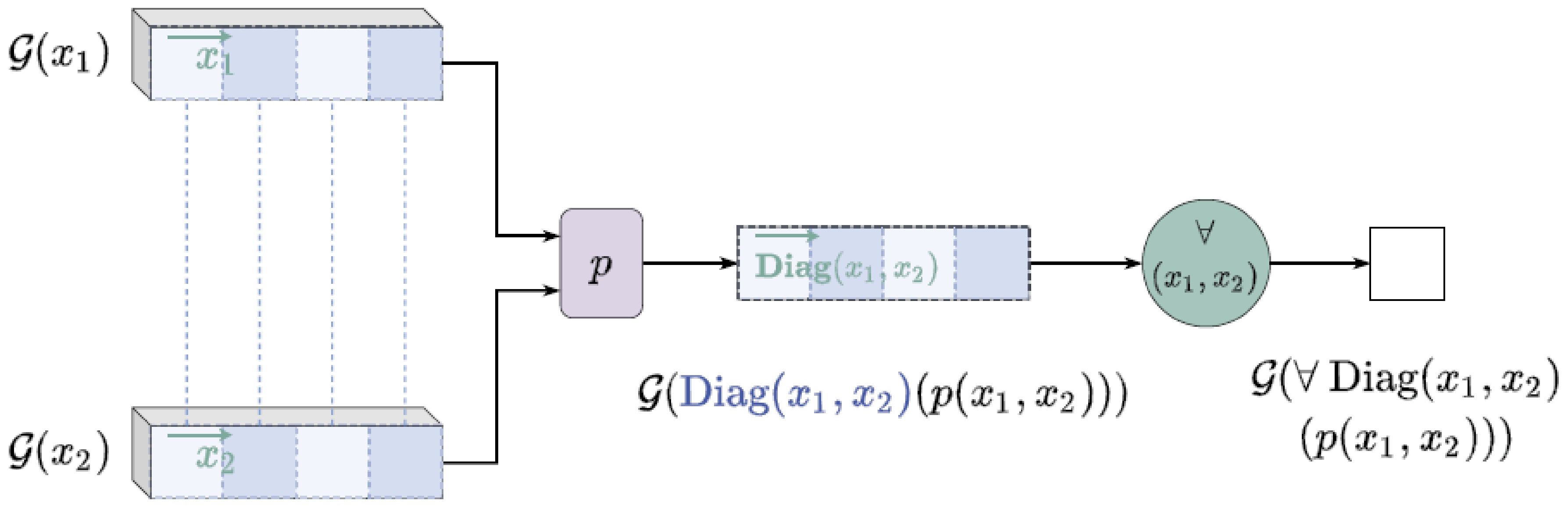
Guarded quantification:

- Quantifies over a **subset** of variables that satisfy a **condition** m (mask)
- Definition:

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

Source: https://github.com/logictensornetworks/

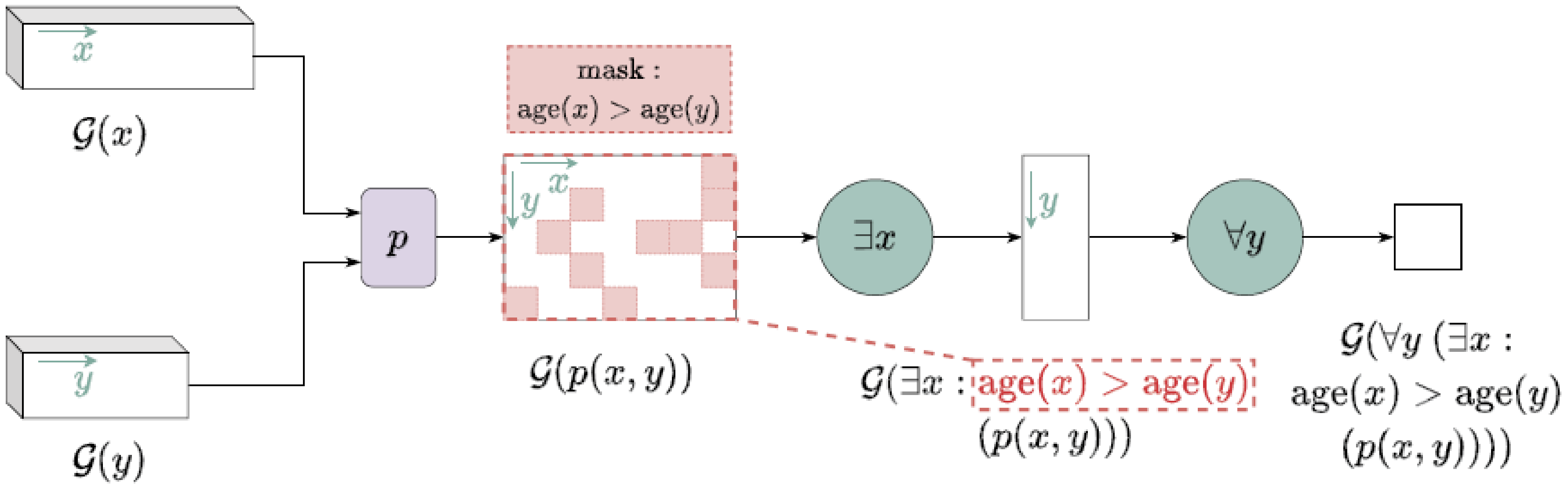
Real Logic: Diagonal Quantification



$\mathcal{G}(x_1)$
x_1
$\mathcal{G}(x_2)$
x_2
p
$\mathrm{Diag}(x_1, x_2)$
$\forall$
(x_1, x_2)
$\mathcal{G}(\mathrm{Diag}(x_1, x_2)(p(x_1, x_2)))$
$\mathcal{G}(\forall\ \mathrm{Diag}(x_1, x_2)$
$(p(x_1, x_2)))$

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Real Logic: Guarded Quantification



x
$\mathcal{G}(x)$
y
$\mathcal{G}(y)$
p
mask : $\text{age}(x) > \text{age}(y)$
y x
$\mathcal{G}(p(x,y))$
$\exists x$
y
$\forall y$
$\mathcal{G}(\exists x : \text{age}(x) > \text{age}(y) (p(x,y)))$
$\mathcal{G}(\forall y\ (\exists x : \text{age}(x) > \text{age}(y) (p(x,y))))$

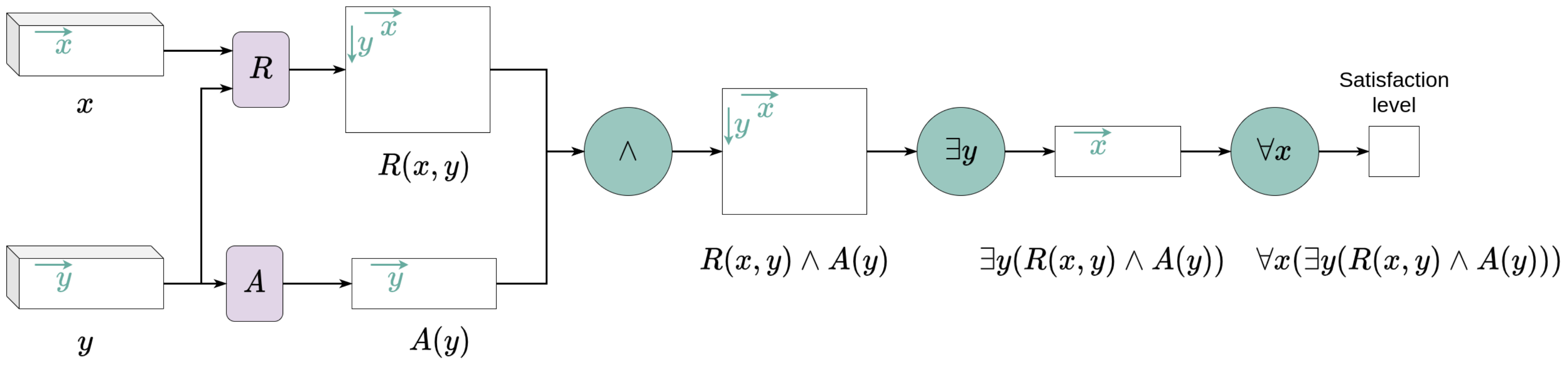
Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

From Real Logic to Logic Tensor Networks

- Up to now, we have presented a kind of fuzzy logic – where do we go from here?
- Towards a machine learning setup:
  - **Objects** are represented by points in a feature space.
  - Functions and predicates are **learnable**.
- Let’s illustrate this with some examples…

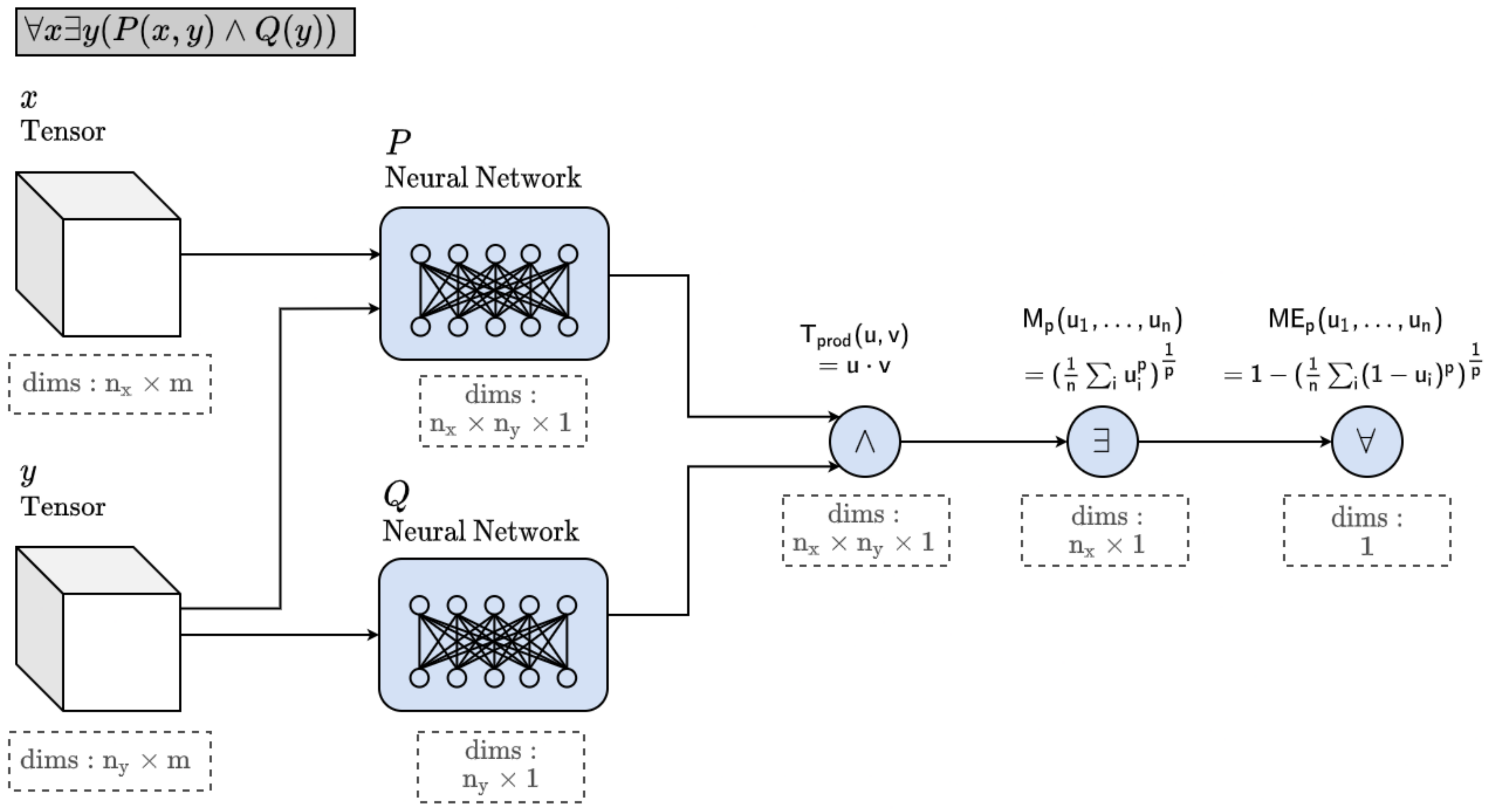
From Real Logic to Logic Tensor Networks

If R denotes the predicate $Friends$, and A denotes the predicate $Italian$, the following computational graph translates the sentence "*everybody has a friend who is Italian*":



Source: https://github.com/logictensornetworks

From Real Logic to Logic Tensor Networks



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

Source: https://github.com/logictensornetworks

From Real Logic to Logic Tensor Networks

- This is powerful because the NN can be used to learn the **membership** function for the corresponding **concept**.
- The underlying feature space can be based on features **extracted** from training data.
- In summary, the **symbolic-subsymbolic** connection is:
  - Subsymbolic: Weights in a neural network that classifies objects, parameters in regressors, etc.
  - Symbolic: information in rules such as “smoking causes cancer”.

From Real Logic to Logic Tensor Networks

Intuitions:

- When we observe a **new object**, we can use the NNs to classify it, and then reason using the formulas.
- Furthermore, rules can be leveraged for learning NN parameters; for instance:
  - First optimize the weights of the “smoker” network so that it correctly classifies individuals w.r.t. their smoker status.
  - But also take into account fuzzy formula *smokes(x)* → *cancer(x)* so that it is true for all points *x* in the feature space.
  - Rules thus provide additional **constraints**.

LTN: Tasks

In Real Logic, one can define the tasks of:

- **Learning:** The task of making generalizations from specific observations obtained from data (often called inductive inference)
- **Reasoning:** The task of deriving what knowledge follows from the facts that are currently known.
- **Query answering (QA):** The task of evaluating the truth value of a certain logical expression (query), or finding the set of objects in the data that evaluate a certain expression to *true*.

To discuss these tasks, we first need to discuss which types of **knowledge** can be represented in Real Logic.

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Representing Knowledge with Real Logic

- **Groundings** are an integral part of the knowledge represented by Real Logic.
- The **connection** between the symbols and the domain is represented explicitly in the language by a grounding $\mathcal{G}$
- An RL **knowledge base** is thus defined by formulas of the logical language and knowledge about the domain in the form of groundings obtained from data.
- There are several **types** of knowledge that can be represented in Real Logic.

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Knowledge through symbol groundings

- Knowledge through symbol groundings:
  - **Boundaries** for domain grounding
  - Explicit definition of grounding for **symbols**
  - **Parametric** definition of grounding for symbols
- Knowledge through formulas:
  - Factual propositions
  - Generalized propositions
- Knowledge through fuzzy semantics

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

LTN Satisfiability

- A Real Logic theory ⍰ = (⍰, ⍰(. | ⍰), ⍰) has three components:
  - Knowledge about the **grounding of symbols** (domains, constants, variables, functions, and predicate symbols);
  - a set of closed logical **formulas** describing factual propositions and general knowledge;
  - **operators** and the **hyperparameters** used to evaluate each formula.
- Learning and reasoning in a Real Logic theory are both associated with searching for and applying the set of values of parameters ⍰ from the hypothesis space ⍰ that **maximize** the **satisfaction** of the formulas in ⍰.

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

LTN Satisfiability

- We use the term **grounded theory**, denoted by $\langle \mathcal{T}, \mathcal{G}_{\theta} \rangle$, to refer to a Real Logic theory with a specific set of **learned** parameter values.
- To define this optimization problem, we aggregate the truth values of all the formulas in $\mathcal{T}$ by selecting a **formula aggregating operator**:

$$SatAgg: [0,1]^{*} \rightarrow [0,1]$$

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

LTN Learning

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

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

- With this general formulation, one can learn the grounding of constants, functions, and predicates:
  - Learning grounding of **constants** corresponds to learning of **embeddings**.
  - Learning grounding of **functions** corresponds to learning **generative models** or a **regression** task.
  - Learning of the grounding of **predicates** corresponds to a **classification** task.

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

LTN Querying

- Given a grounded theory, QA allows one to **check** if a certain **fact** is true (rather, by *how much* it is true).
- Various types of queries can be asked:
  - **Truth queries:** What is the truth value of a formula in the language? If the formula is closed, we get a scalar, if it has n free variables, we get a tensor of order n.
  - **Value queries:** What is the value of a term in the language? Analogous to truth queries.
  - **Generalization truth queries:** What is the truth value associated to a formula evaluated over unseen data?
  - **Generalization value queries:** Analogous

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

LTN Reasoning

- **Reasoning** is the task of verifying if a formula is a logical consequence of a set of formulas.
- A formula φ is a fuzzy **logical consequence** of a finite set of formulas ⍰ iff every model of ⍰ is a model of φ.
- In Real Logic, this is **generalized** by defining an interval $[q, 1]$ with $0.5 < q < 1$ and assuming that a formula is true iff its truth-value is in the interval $[q, 1]$.
- Unfortunately, this definition is only useful in theory since it requires inspecting potentially **infinite** sets of groundings.

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

LTN Approximate Reasoning

Option 1: Querying after learning

- Consider only the grounded theories that maximally satisfy the given theory.
- A (likely incomplete) set of such grounded theories can be found via **multiple optimization runs**.
- This is a kind of **brave reasoning**.

Option 2: Proof by refutation

- Search for a **counterexample** to the consequence.
- If no such example is found, the consequence is **assumed** to hold.
- The general formulation cannot be used as an objective function due to null derivatives – see paper for a soft constraint.

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Use Case: Smokers, Friends, Cancer

Let’s consider the classic example introduced in the Markov Logic Networks paper (Domingos et al., 2006):

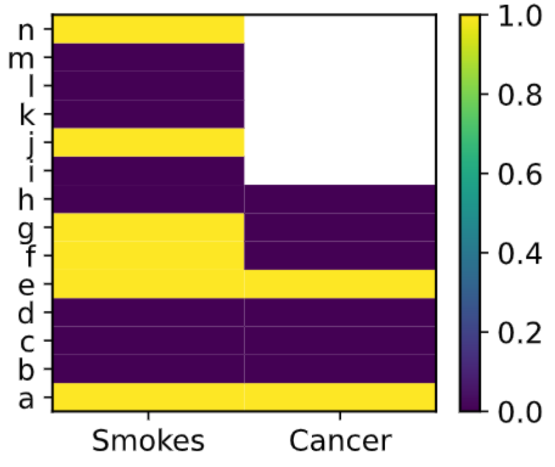
- 14 people divided into two groups: $\{a, b, \dots, h\}$ and $\{i, j, \dots, n\}$.
- Within each group, there **is complete knowledge** about **smoking** habits.
- In the first group, there is **complete knowledge** about who has and who does not have **cancer**.
- Knowledge about the **friendship** relation is complete within each group only if symmetry is assumed (i.e., $\forall x, y\ friends(x, y) \rightarrow friends(y, x)$). Otherwise, knowledge about friendship is **incomplete**.
- Finally, general knowledge about smoking, friendship, and cancer:
  - smoking causes cancer,
  - friendship is normally symmetric and anti-reflexive,
  - everyone has a friend, and
  - smoking propagates (actively or passively) among friends.

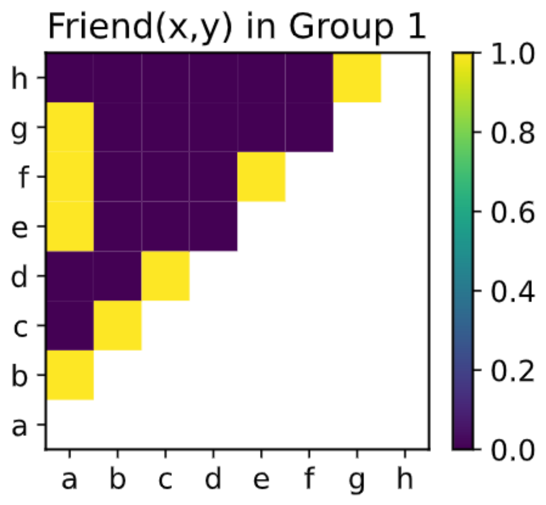
Source: https://github.com/logictensornetworks

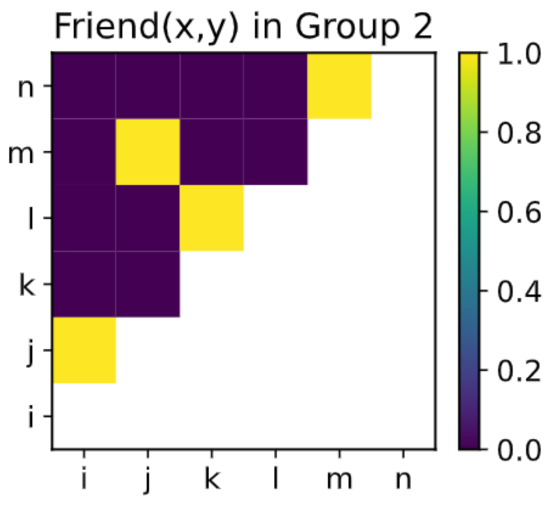
Use Case: Smokers, Friends, Cancer

Language:

- LTN **constants** are used to denote the individuals. Each is grounded as a trainable embedding.
- Smokes, Friends, Cancer **predicates** are grounded as simple MLPs.
- All **rules + facts** are formulated in the knowledgebase.
- **Inconsistency:** for example, person f smokes, doesn't have cancer.
- **Incompleteness:** for example, inter-group friendship, cancer in group 2.





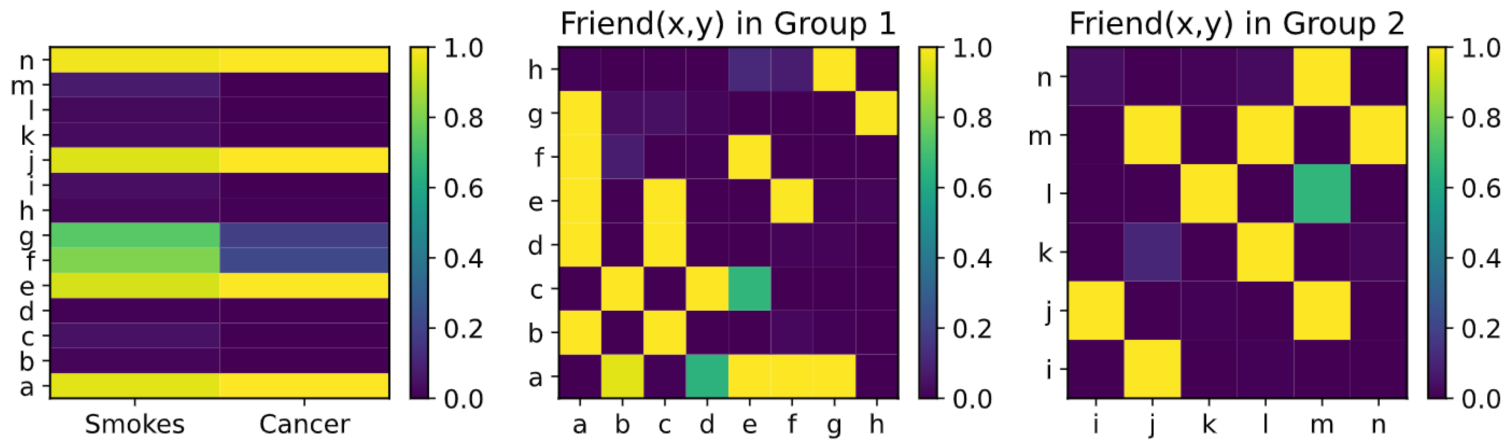


Source: https://github.com/logictensornetworks

Use Case: Smokers, Friends, Cancer

After training, we can:

- **Test satisfiability** of the axioms:



- **Issue queries** with new formulas:

forall p: Cancer(p) -> Smokes(p): 0.96

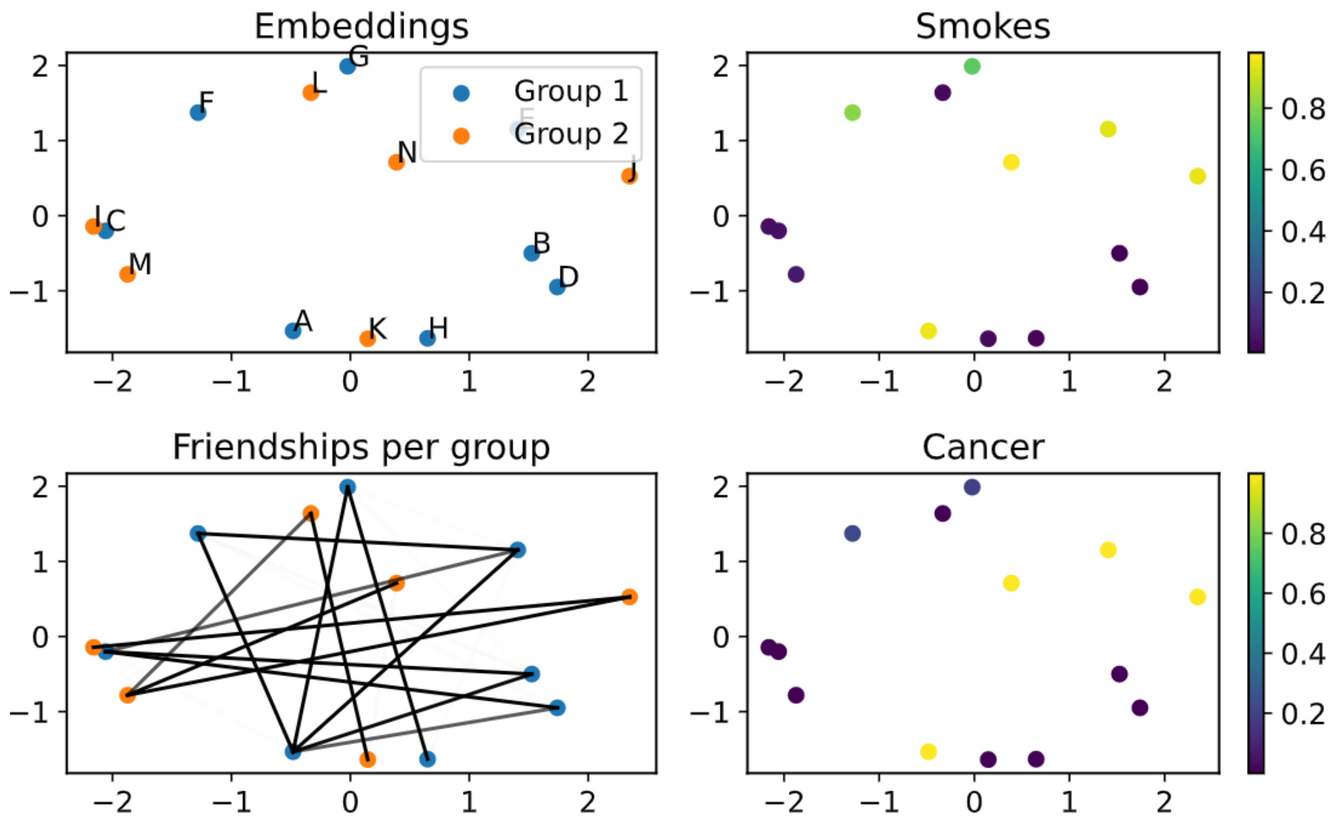
forall p,q: (Cancer(p) or Cancer(q)) -> Friends(p,q): 0.22

Source: https://github.com/logictensornetworks

Use Case: Smokers, Friends, Cancer

After training, we can (cont.):

- **Visualize** the embeddings:



Source: https://github.com/logictensornetworks

RRNs / Deep Ontological Reasoning: Overview

- Introduction and Motivation
- Recursive Reasoning Networks (RRNs)
  - Problem Statement
  - Intuitions
  - Deeper Dive
- Experimental Evaluation

Introduction and Motivation

"Human-like reasoning" is one of the major goals of AI:

- Machine learning approaches typically "*entertain a quite informal notion of reasoning, which is often simply identified with a particular kind of prediction task.*"
- KR&R typically approaches reasoning as the application of some kind of mathematical proof theory.
- It is well known that each approach has pros and cons:
  - **ML:** Good results on specific domains, handles noise well, scalable, but brittle in general.
  - **KR&R:** Expressive, correct, flexible representations, but not scalable in general, and noise causes major issues.
- **Recursive Reasoning Networks (RRN)** as an NSR model.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Introduction and Motivation

Why take an ML approach to reasoning?

- **Combining** ML and KR&R is commonly regarded as the way towards further progress in AI.
- Expected to allow:
    - Leveraging background knowledge to address the **brittleness** problem.
    - Knowledge transfer
    - Learning from **smaller amounts of data**
    - **Explainable** symbolic inference of learned neural models

Goal: Employ SOTA techniques from DL to “manage the balancing act” and obtain the best of both worlds.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Remarks

- Ontological KBs can be seen as a way to formalize information in terms of **individuals**, **classes**, and **relations**.
- **Knowledge graphs** are special cases of such KBs:
  - Individuals correspond to nodes
  - Binary relations correspond to labeled directed edges
  - Classes correspond to unary vertex labels
- Facts in KGs are typically stated in the form of **triples**:

  <subject, predicate, object>
- It is common to assume a **fixed vocabulary**, so all classes and relations are fixed a priori.

Towards a Problem Statement

- The authors consider a language based on **Datalog**:
  - **Relations** have arity 1 or 2.
  - No functional symbols.
  - Rules with “□” in the head are allowed and represent **constraints**.
- **Rules** are of the form: $□_1$ □ … □ $□_n$ □ □
- **Negative constraints** are of the form: $□_1$ □ … □ $□_n$ □ □

  In both cases, □ and $□_1$, …, $□_n$ are atoms and n □ 0.
- **Facts** are rules without bodies, and **literals** are either facts □ or negated facts □□.
- **Databases** are finite sets of facts.

Datalog Ontologies: Example

Ontology:	$\texttt{human(X)} \leftarrow \texttt{holds(X,\_)}$	Only human beings can hold things.
	$\texttt{object(Y)} \leftarrow \texttt{holds(\_,Y)}$	Only objects can be held.
	$\bot \leftarrow \texttt{human(X)} \wedge \texttt{object(X)}$	Objects are not human beings and vice versa.
	$\texttt{isAt(Y,Z)} \leftarrow \texttt{holds(X,Y)} \wedge \texttt{isAt(X,Z)}$	Objects are at the same location as the one holding them.
	$\bot \leftarrow \texttt{isAt(X,Y)} \wedge \texttt{isAt(X,Z)} \wedge \texttt{Y} \neq \texttt{Z}$	Nobody/nothing can be at two locations at the same time.
Facts:	`holds(mary,apple)`	Mary holds the apple.
	`isAt(mary,kitchen)`	Mary is in the kitchen.
Queries:	`?human(apple)`	Is the apple a human being? (Evaluates to **false**.)
	`?isAt(apple,kitchen)`	Is the apple in the kitchen? (Evaluates to **true**.)
	`?isAt(mary,bedroom)`	Is Mary in the bedroom? (Evaluates to **false**.)

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Towards a Problem Statement

- Language **semantics** is straightforward and defined as usual:
  - Herbrand interpretations
  - Satisfaction of rules by interpretations
  - Sets of rules satisfiable by interpretations (models)
  - Logical entailment of facts by sets of rules
  - Unique least satisfying interpretations (minimal models)
  - Negated facts derived via the (Local) Closed World Assumption defined as usual: *R* □ □□ iff *R* □ □ (local variant considers “close” atoms)
  - Logical entailment is denoted with the symbol “ □”
- **Assume that:** *D* is variable and of size *k*, □ is fixed, □ is variable.
- **Problem:** Given a database *D*, a program □, and a literal □, decide whether *D* □ □ □□.

Problem Statement

- **Problem:** Given a database *D*, a program �, and a literal �, <u>decide</u> whether *D* � � ��.
- **Solution (goal):** Derive a neural network *N*[�,*k*] with binary output that, given:
  - an **arbitrary** *D* of size at most *k*, and
  - an **arbitrary** literal �

  is such that *N*[�,*k*] = 1 if and only if *D* � � ��
- **Recursive Reasoning Networks (RRNs):** Designed specifically for this class of problems.

RRN Model: Intuitions

Formal ontology reasoning is **replaced** with computing a learned deep neural network called an RRN:

- Every RRN is trained relative to a particular **ontology** – like its counterpart, it is **independent** of specific facts.
- The **vocabulary** of classes and relations in the ontology determines the **structure** of the recursive layers available in the RRN.
- In contrast, the rules are not provided, they must be **learned** from the training **data**.
- Application of trained model on specific facts is done in two stages:
  - Generate **vector representations** (embeddings) for individuals in the data.
  - Compute **predictions for queries** solely based on such embeddings.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

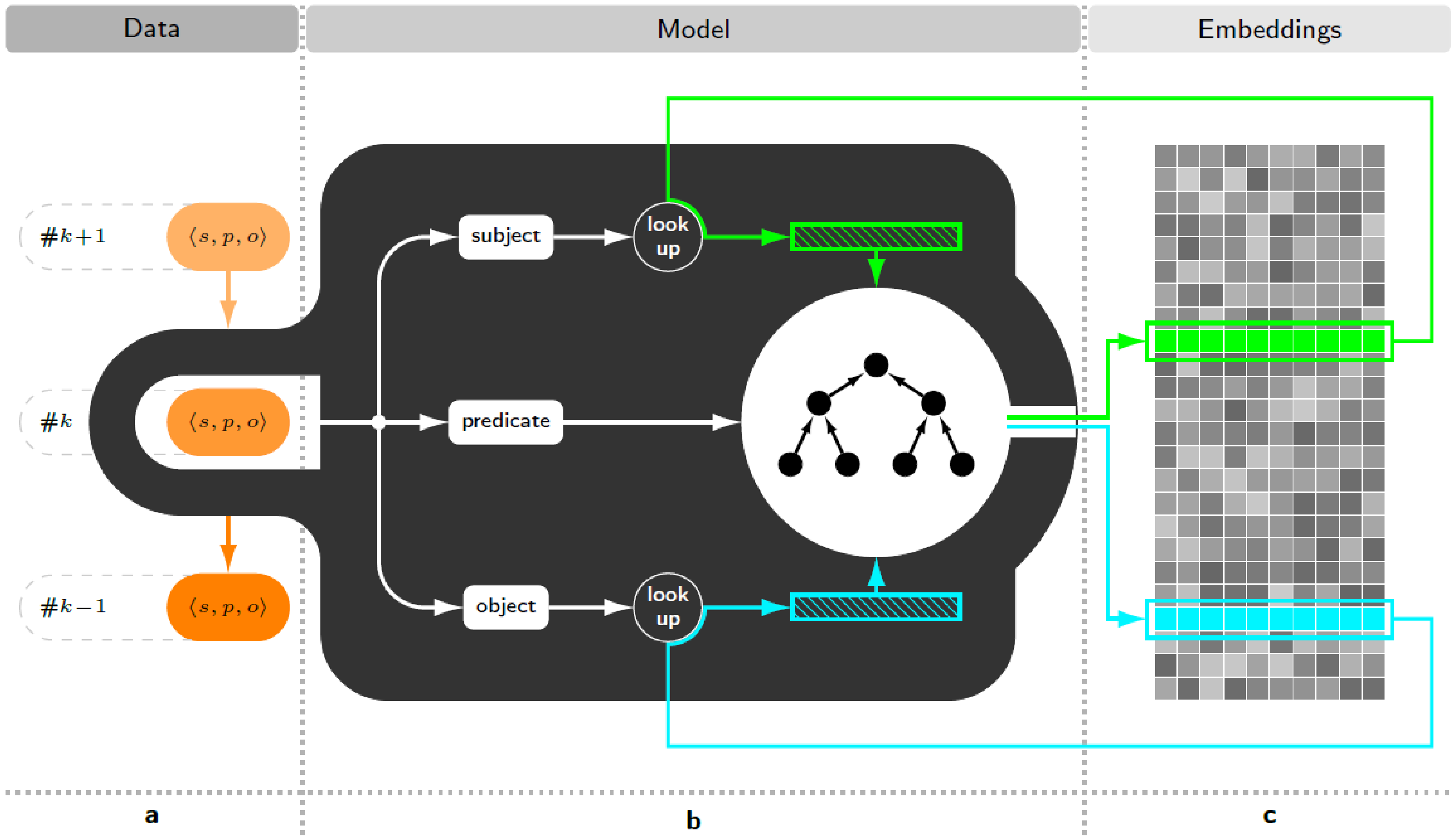
RRN Learning: Intuitions

Based on the idea that all the information on an individual – both **specified** and **inferable** – can be encoded in its **embedding**:

- Start by **randomly generating** initial embeddings for all individuals appearing in the data.
- **Iterate** over all input triples and **update** the embeddings of individuals involved.
- This considers both the individuals' embeddings and the possible **inferences** afforded by the provided facts.
- Clearly, the process involves going through all the data multiple times – this is essential for encoding **reasoning chains**.
- The process is the same as that for learning classical RNNs.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

RRN Learning: Overview



Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

RRN Prediction: Intuition

- Learned embeddings can be used to answer **atomic queries** about the data they are obtained from.
- For this purpose, the model provides multi-layer perceptrons (MLPs) for both:
  - **Relations** between two individuals
  - **Class memberships** of a single individual
- This second step only “uncovers knowledge” that was **encoded** during the learning process.
- The model thus allows to unravel complex relationships and considers relations and classes simultaneously.
- **Note:** *Triples are not treated as text* – RRNs are thus agnostic to individual names used in the database.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

A Deeper Dive: Model

Embeddings:

- **Individuals** are represented as unit vectors in $\mathbb{R}^d$.
- Hyperparameter *d* to be chosen based on the ontology: expressiveness, vocabulary size, and number of individuals.

Update Layers:

- Need to define two kinds: **relations** and **classes**.
- Relations are in general **not symmetric**, so in total 4 update layers (subject/object and positive/negative).
- The RRN’s recursive layers define a **gated network** (a kind of RNN): Gates control how candidate update steps are applied.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

A Deeper Dive: Model based on GRUs

One-sided updates for relations:

Updating subject's embedding (the one for the object's is analogous)

Logistic function

Gate controlling how much of the update is applied

Model parameters

Calculation of candidate update steps

$$\mathbf{g}^{\triangleleft P}(s,o) = \sigma\left(\mathbf{V}_1^{\triangleleft P}\mathbf{e}_s + \mathbf{V}_2^{\triangleleft P}\mathbf{e}_o\right),$$

$$\hat{\mathbf{e}}_s^{(1)} = ReLU\left(\mathbf{W}_1^{\triangleleft P}\mathbf{e}_s + \mathbf{W}_2^{\triangleleft P}\mathbf{e}_o + \mathbf{e}_s{\mathbf{e}_o}^T\mathbf{w}^{\triangleleft P}\right),$$

$$\hat{\mathbf{e}}_s^{(2)} = \mathbf{e}_s + \hat{\mathbf{e}}_s^{(1)} \circ \mathbf{g}^{\triangleleft P}(s,o),$$

$$\mathbf{e}_s = Update^{\triangleleft P}(s,o) = \frac{\hat{\mathbf{e}}_s^{(2)}}{\|\hat{\mathbf{e}}_s^{(2)}\|_2}.$$

Simultaneous update operation:

$$\langle \mathbf{e}_s, \mathbf{e}_o \rangle = Update(\langle s, P, o \rangle) = \left\langle Update^{\triangleleft P}(s,o),\ Update^{P\triangleright}(s,o) \right\rangle$$

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

A Deeper Dive: Model based on GRUs

- **Classes** are one-sided and we don't need a positive and negative version, so we simply have:

$$\mathbf{g}(i) = \sigma\left(\underset{\uparrow}{\mathbf{V}} \cdot [\mathbf{e}_i : \mathbb{1}_{KB}(i)]\right),$$

Model parameters

$$\hat{\mathbf{e}}_i^{(1)} = ReLU(\underset{\uparrow}{\mathbf{W}} \cdot [\mathbf{e}_i : \mathbb{1}_{KB}(i)]),$$

$$\hat{\mathbf{e}}_i^{(2)} = \mathbf{e}_i + \hat{\mathbf{e}}_i^{(1)} \circ \mathbf{g}(i),$$

$$\mathbf{e}_i = Update(i) = \frac{\hat{\mathbf{e}}_i^{(2)}}{\|\hat{\mathbf{e}}_i^{(2)}\|_2}.$$

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

A Deeper Dive: Prediction

MLPs for computing **predictions**:

- A **single** MLP for all classes:

  $MLP^{(\text{classes})}$ – expects a **single** embedding as input and returns probabilities for the corresponding individual’s membership in each class.

- One for each **relation** R in the vocabulary:

  - $MLP^{(R)}$ – expects a **pair** of embeddings and provides a probability for the relation to hold between the corresponding individuals.

  - The **negation** of the relation is simply computed as 1 ⍰ $MLP^{(R)}(\mathbf{e}_i, \mathbf{e}_j)$

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

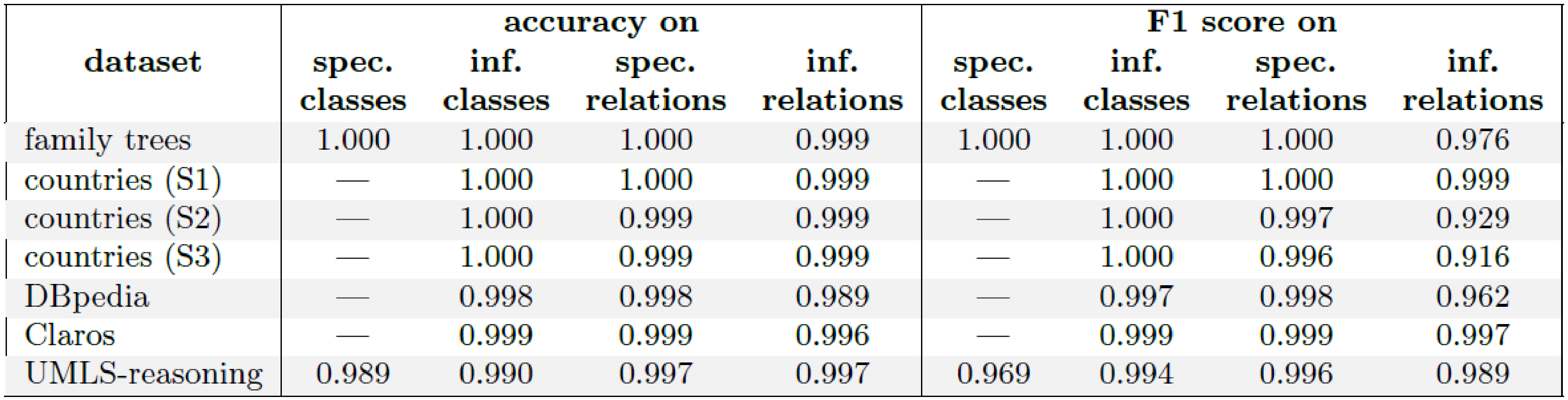
Experiments: Synthetic and Real Datasets

	dataset	# sample KBs		avg. ind.	vocabulary size	
		train	test	per sample	# class types	# relation types
Synth.	family trees	5,000	500	23	2	29
	countries (S1)	5,000	20	240	3	2
	countries (S2)	5,000	20	240	3	2
	countries (S3)	5,000	20	240	3	2
Real	DBpedia	5,000	500	200	101	518
	Claros	5,000	500	200	33	77
	UMLS-reasoning	5,000	500	60	127	53

dataset	avg. classes per sample		avg. relations per sample	
	specified (pos./neg.)	inferable (pos./neg.)	specified (pos./neg.)	inferable (pos./neg.)
family trees	23 / —	— / 23	28 / —	240 / 16,160
countries (S1)	— / —	238 / 478	820 / —	20 / 49,261
countries (S2)	— / —	238 / 478	782 / —	39 / 49,279
countries (S3)	— / —	238 / 478	757 / —	68 / 49,275
DBpedia	— / —	183 / —	642 / —	156 / —
Claros	— / —	1,499 / —	518 / —	17,072 / —
UMLS-reasoning	43 / 39	1,194 / —	28 / 44	59 / 345

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Results



dataset	accuracy on				F1 score on			
	spec. classes	inf. classes	spec. relations	inf. relations	spec. classes	inf. classes	spec. relations	inf. relations
family trees	1.000	1.000	1.000	0.999	1.000	1.000	1.000	0.976
countries (S1)	—	1.000	1.000	0.999	—	1.000	1.000	0.999
countries (S2)	—	1.000	0.999	0.999	—	1.000	0.997	0.929
countries (S3)	—	1.000	0.999	0.999	—	1.000	0.996	0.916
DBpedia	—	0.998	0.998	0.989	—	0.997	0.998	0.962
Claros	—	0.999	0.999	0.996	—	0.999	0.999	0.997
UMLS-reasoning	0.989	0.990	0.997	0.997	0.969	0.994	0.996	0.989

Table 2: This table summarizes our experimental results. Accuracy and F1 score are reported separately for class memberships and relations, and within each group, for those triples that describe specified knowledge, that is, facts, and those that represent inferable information.

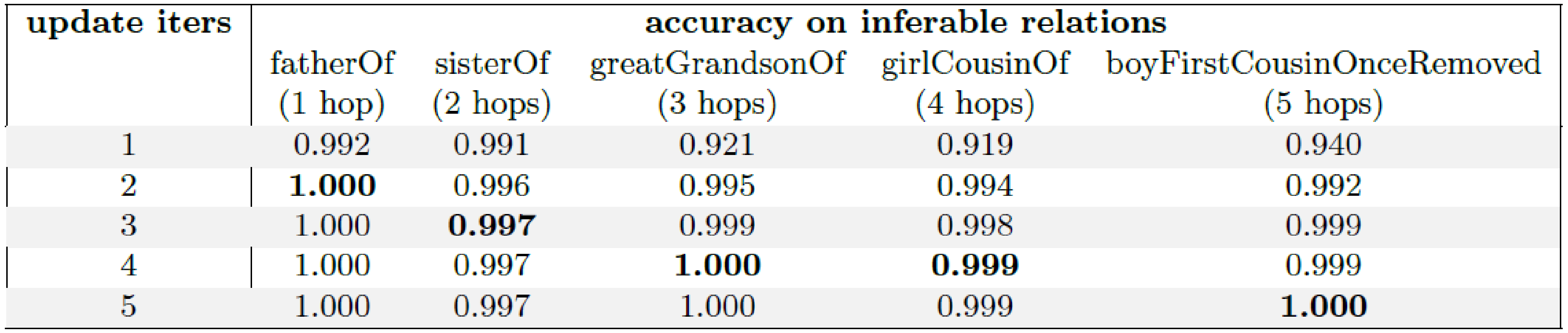
Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Further experiments: Tolerance to Noise

- **Corrupted** the test data:
  - **Missing information:** Randomly removed one fact that could not be inferred.
  - **Inconsistency:** Randomly chose one fact in each test sample and added a negated version of the same as another fact.
- **Results:**
  - Reconstruction of missing information:
    - DBpedia: 33.8%
    - Claros: 38.4%
  - Inconsistency:
    - DBpedia: 88.4%
    - Claros: 96.2%

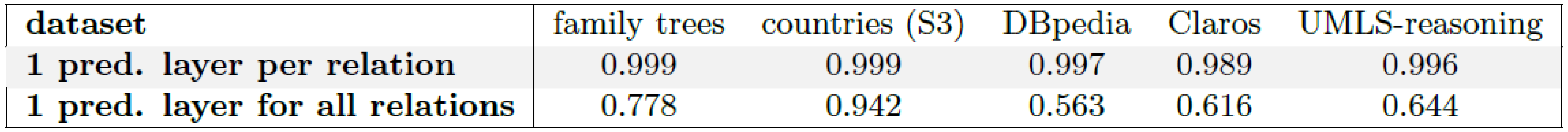
Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Further Results: #Updates and #Layers



update iters	**accuracy on inferable relations**				
	fatherOf (1 hop)	sisterOf (2 hops)	greatGrandsonOf (3 hops)	girlCousinOf (4 hops)	boyFirstCousinOnceRemoved (5 hops)
1	0.992	0.991	0.921	0.919	0.940
2	**1.000**	0.996	0.995	0.994	0.992
3	1.000	**0.997**	0.999	0.998	0.999
4	1.000	0.997	**1.000**	**0.999**	0.999
5	1.000	0.997	1.000	0.999	**1.000**

Table 4: The prediction accuracies for inferable relations with different predicates in the family trees dataset after different numbers of update iterations. Bold-faced values mark the lowest number of update iterations that allow for achieving the best possible accuracy for a relation type.



dataset	family trees	countries (S3)	DBpedia	Claros	UMLS-reasoning
1 pred. layer per relation	0.999	0.999	0.997	0.989	0.996
1 pred. layer for all relations	0.778	0.942	0.563	0.616	0.644

Table 5: The accuracies for inferable relations that were achieved with one prediction layer per relation type on the one hand and a single prediction layer for all relations on the other.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

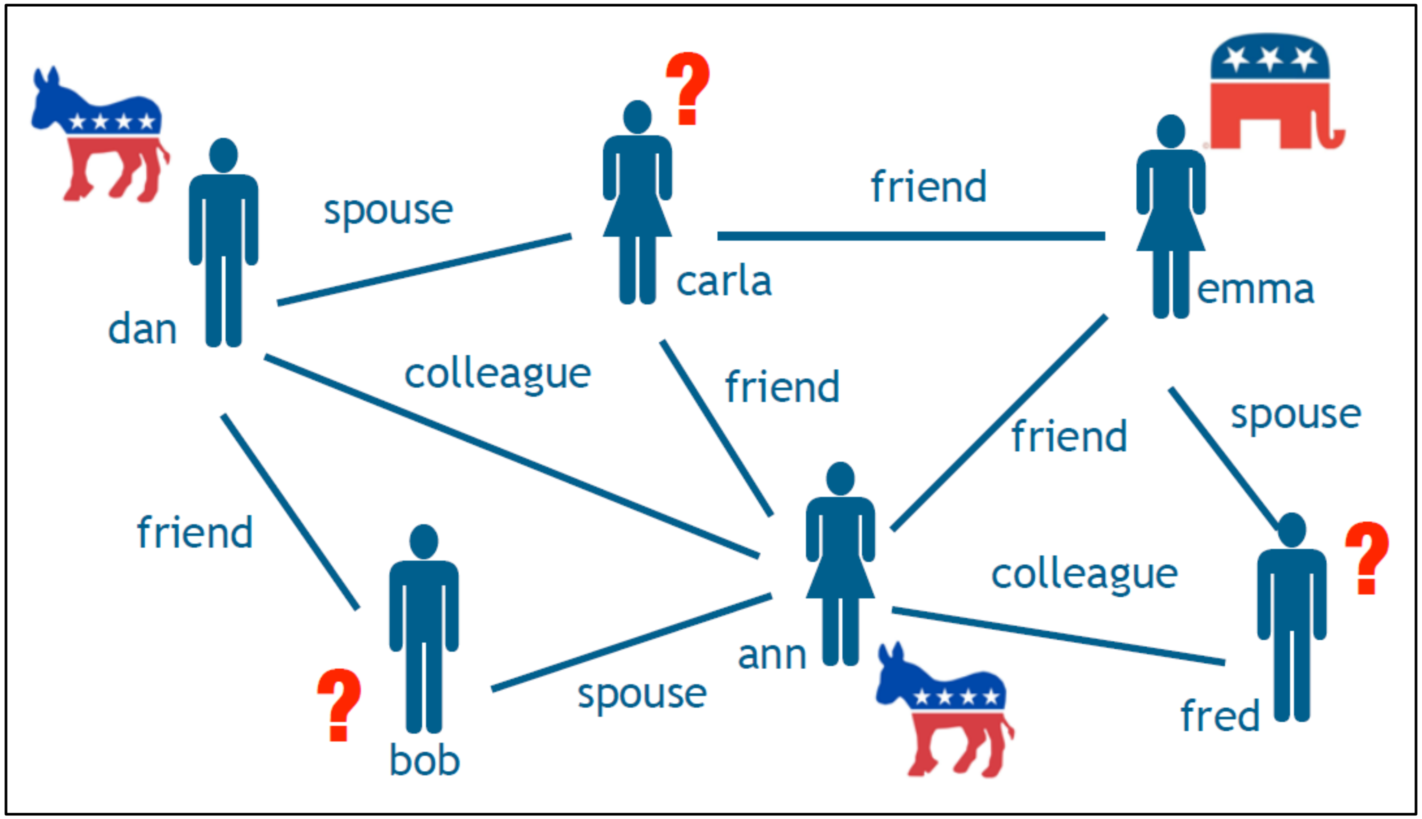
Probabilistic Soft Logic (PSL)

- In the same spirit as MLNs, PSL is a declarative language to **specify** PGMs.
- Main features:
  - Logical atoms with **soft truth values** in [0,1] – this means that **continuous** models are required.
  - Dependencies encoded via weighted first order rules
  - Support for similarity functions and aggregation
  - Linear (in)equality constraints
  - Efficient MPE inference via continuous **convex optimization**

This section is based on: Kimmig, Bach, Broecheler, Huang, Getoor (2012): “A Short Introduction to Probabilistic Soft Logic”. NIPS Workshop on Probabilistic Programming.

PSL Application: Voter Opinion Modeling

Given a social network labeled with different relationships between nodes and how **some** of them voted, can we infer anything about how the **others** voted?



Source: [Kimmig et al., 2012]

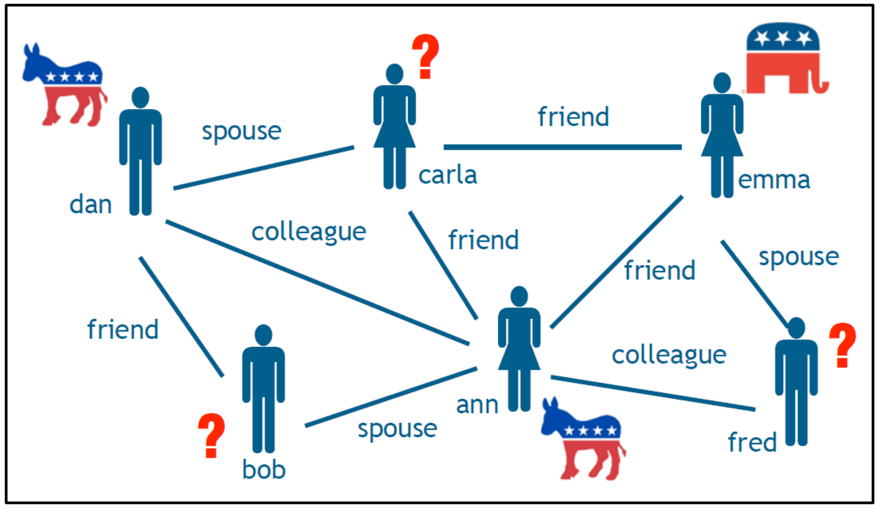
PSL Programs

Ground atoms correspond to random variables:

- friend(carla, emma),
- friend(bob, dan),
- spouse(carla, dan), …

Soft truth value assignments:

- friend(carla, emma) = 0.9
- friend(bob, dan) = 0.4



Source: [Kimmig et al., 2012]

Weighted Rules:

- Local rule: 0.3: lives(A, S) ∧ majority(S, P) → prefers(A, P)
- Propagation rule: 0.8: spouse(B, A) ∧ prefers(B, P) → prefers(A, P)
- Similarity rule: similarAge(B, A) ∧ prefers(B, P) → prefers(A, P)

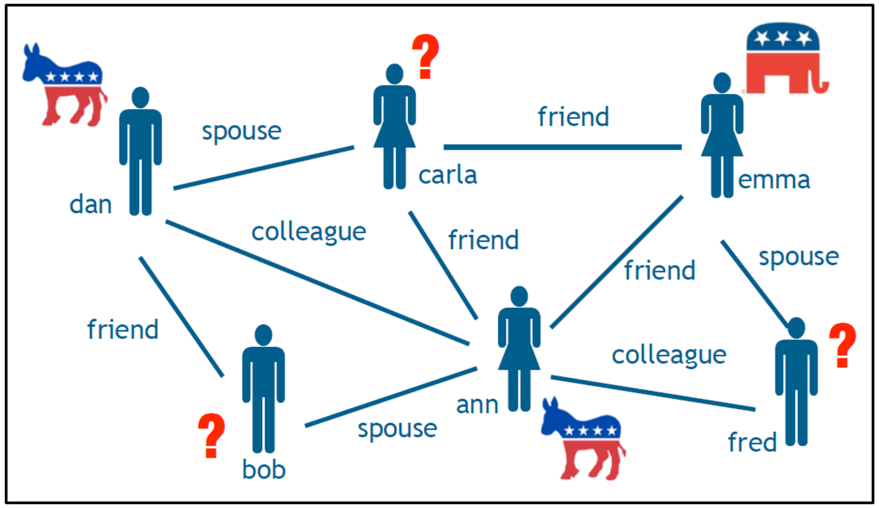
PSL Programs

Partial functions:

- prefers(A, dem) + prefers(A, rep) ≤ 1.0

Sets:

- 0.4: prefers(A, P) → prefersAvg({A.friend}, P)
  - **A.friend**: All X s.t. friend(A,X)
  - Truth value of **prefersAvg**: average truth value of all atoms of the form prefers(X,P)



Source: [Kimmig et al., 2012]

PSL: Probabilistic Model

Ground rule's distance from satisfaction given I

∈{1,2}

$$f(I) = \frac{1}{Z} \exp\left(- \sum_{r \in P} \sum_{g \in G(r)} w_r (d_g(I))^k \right)$$

Interpretation

Rule's weight

Set of rule groundings

Normalization constant

$$Z = \int_{J \in \mathcal{I}} \exp\left(- \sum_{r \in P} \sum_{g \in G(r)} w_r (d_g(J))^k \right)$$

Source: [Kimmig et al., 2012]

PSL: Probabilistic Model

Distance to satisfaction:

$$d_r(I) = \max\{0, I(body) - I(head)\}$$

- Intuitively, "*if body then head*" is satisfied iff the truth value of the body is less than or equal to the truth value of the head.
- This is a **generalization** of classical logical implication.

In order to compute values of conjunctions and disjunctions, PSL uses the Lukasiewicz **infinite value** logic operators:

$$I(v_1 \wedge v_2) = \max\{0, I(v_1) + I(v_2) - 1\}$$

$$I(v_1 \vee v_2) = \min\{1, I(v_1) + I(v_2)\}$$

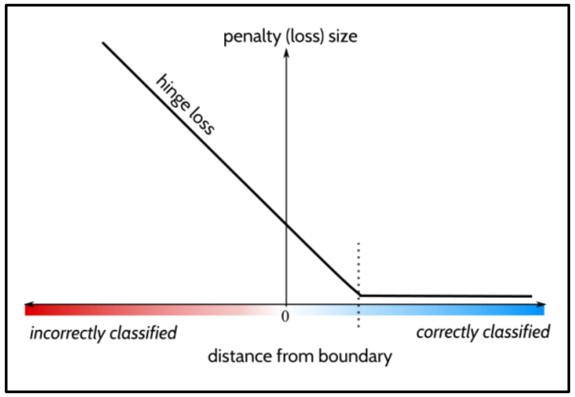
$$I(\neg v) = 1 - I(v)$$

PSL: Probabilistic Model

PSL programs ground out to special kinds of MRFs called **Hinge-loss MRFs**.

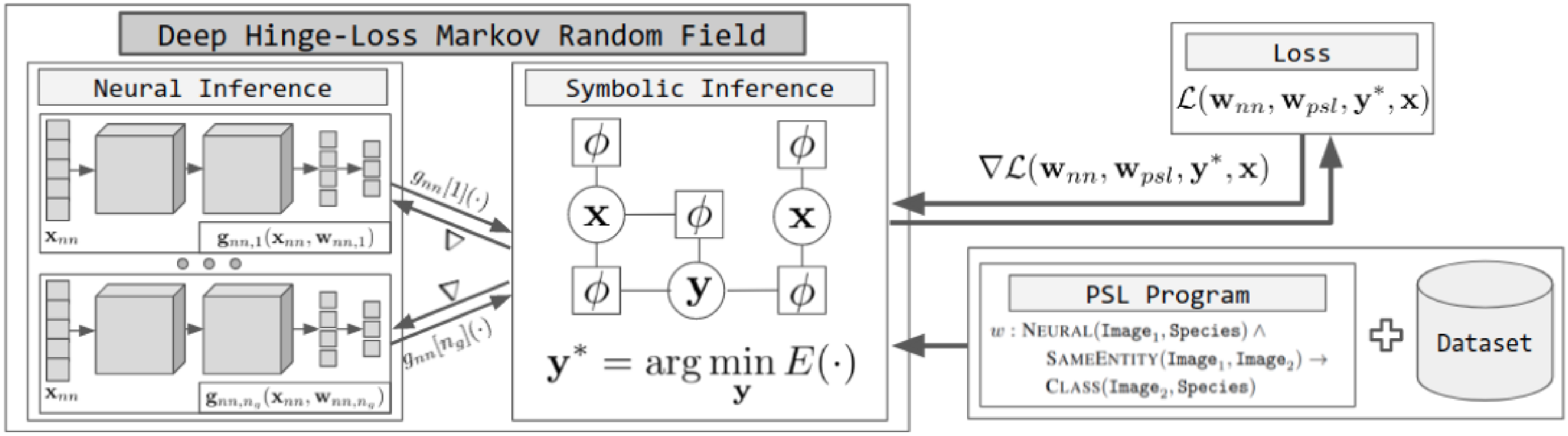
- Nodes are **continuous** variables in [0,1]
- **Potentials** are hinge-loss functions
- **Log-concave**: This means that a best interpretation can be found tractably.
- Details are out of scope; interested students are referred to:

  Bach, S. H., Broecheler, M., Huang, B., & Getoor, L. (2017): "*Hinge-loss Markov Random Fields and Probabilistic Soft Logic*"



Source: https://math.stackexchange.com/questions/782586/how-do-you-minimize-hinge-loss

From PSL to NeuPSL



Deep Hinge-Loss Markov Random Field
Neural Inference
$\mathbf{x}_{nn}$
$\mathbf{g}_{nn,1}(\mathbf{x}_{nn}, \mathbf{w}_{nn,1})$
$\mathbf{g}_{nn,n_g}(\mathbf{x}_{nn}, \mathbf{w}_{nn,n_g})$
$g_{nn}[1](\cdot)$
$g_{nn}[n_g](\cdot)$
Symbolic Inference
ϕ
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{y}^* = \arg\min_{\mathbf{y}} E(\cdot)$
Loss
$\mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{psl}, \mathbf{y}^*, \mathbf{x})$
$\nabla\mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{psl}, \mathbf{y}^*, \mathbf{x})$
PSL Program
w : NEURAL(Image1, Species) ∧ SAMEENTITY(Image1, Image2) → CLASS(Image2, Species)
Dataset

- NeuPSL leverages NNs for low-level perception, integrating their outputs into a set of symbolic potentials created by a PSL program.
- The symbolic potentials and neural networks together define a Deep-HL-MRF that supports scalable convex joint inference.
- NeuPSL and LTNs are examples of **NeSy Energy-based Models**.

Source: Pryor, Dickens, Augustine, Albalak, Wang, Getoor (2022). NeuPSL: Neural Probabilistic Soft Logic. arXiv preprint arXiv:2205.14268.

Questions

Additional Material

LTNs: Differentiable Fuzzy Logic

Differentiable Fuzzy Logic

- In presenting Real Logic and LTN, we did not give much thought to the role of **fuzzy operators** in the main **tasks**.
- Gradient descent requires that operators be **differentiable** so that it can smoothly traverse the universe of values.
- Three types of **gradient problems** commonly arise:
  - **Single-Passing:** The derivatives of some operators are non-null for only one argument. The gradients propagate to only one input at a time.
  - **Vanishing Gradients:** Gradients vanish on some part of the domain. Learning does not update inputs that are in the vanishing domain.
  - **Exploding Gradients:** Large error gradients accumulate and result in unstable updates.

Sources:

van Krieken et al. (2022). Analyzing differentiable fuzzy logic operators. Artif. Intell., 302, 103602

Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

https://github.com/logictensornetworks

Problems: Single-Passing Gradients

Some operators have gradients propagating to only one input at a time, meaning that all other inputs will not benefit from learning at this step.

e.g. in $\min(u_1, \ldots, u_n)$.

```
xs = tf.constant([1.,1.,1.,0.5,0.3,0.2,0.2,0.1])

with tf.GradientTape() as tape:
    tape.watch(xs)
    y = forall_min(xs)
res = y.numpy()
gradients = tape.gradient(y,xs).numpy()
print(res)
print(gradients)
```

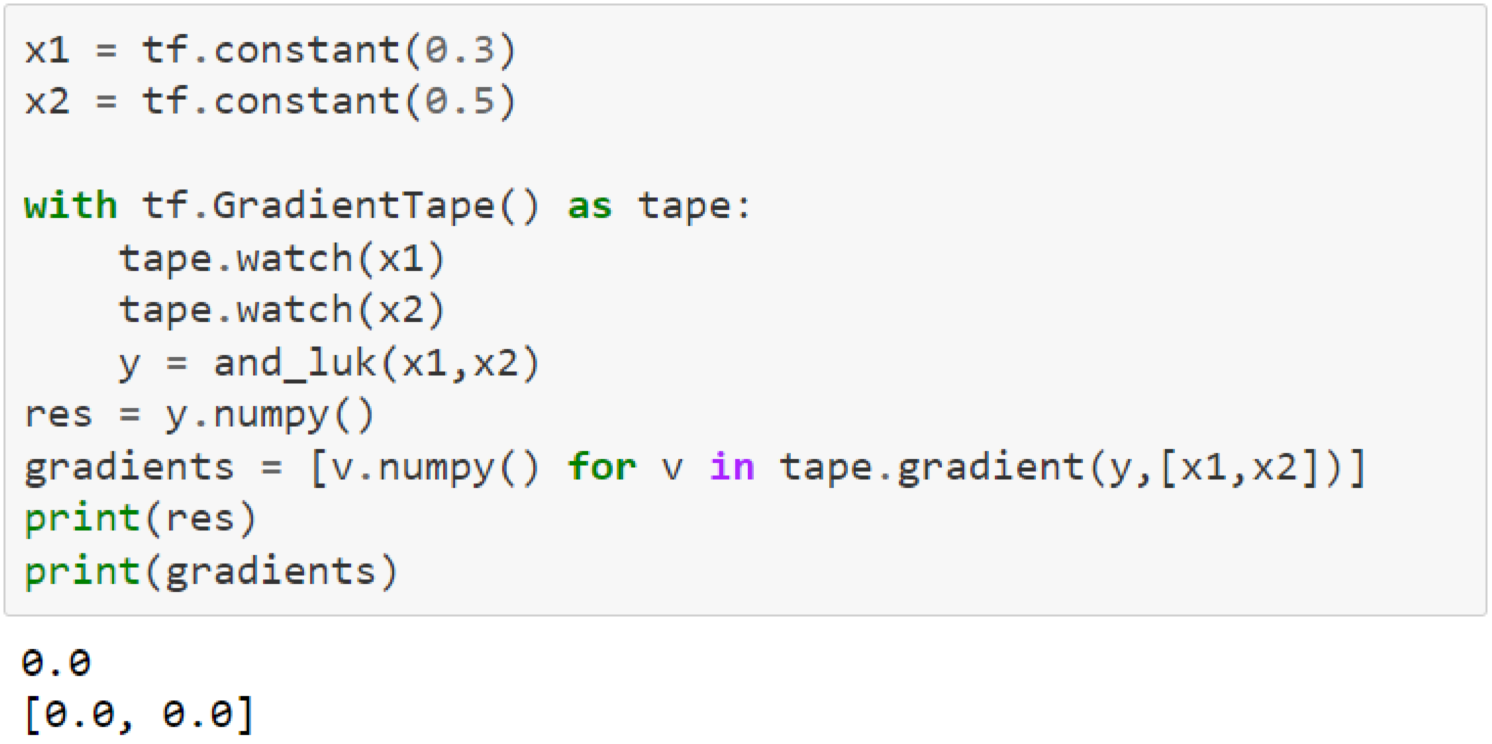
```
0.1
[0. 0. 0. 0. 0. 0. 0. 1.]
```

Source: https://github.com/logictensornetworks

Problems: Vanishing Gradients

Some operators have vanishing gradients on some part of their domains.

e.g. in $u \wedge_{\text{luk}} v = \max(u + v - 1, 0)$, if $u + v - 1 < 0$, the gradients vanish.



```python
x1 = tf.constant(0.3)
x2 = tf.constant(0.5)

with tf.GradientTape() as tape:
    tape.watch(x1)
    tape.watch(x2)
    y = and_luk(x1,x2)
res = y.numpy()
gradients = [v.numpy() for v in tape.gradient(y,[x1,x2])]
print(res)
print(gradients)
```

```
0.0
[0.0, 0.0]
```

Source: https://github.com/logictensornetworks

Problems: Exploding Gradients

Some operators have exploding gradients on some part of their domains.

e.g. in $\mathrm{pME}(u_1, \ldots, u_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - u_i)^p \right)^{\frac{1}{p}}$, on the edge case where all inputs are 1.0.

```python
xs = tf.constant([1.,1.,1.])

with tf.GradientTape() as tape:
    tape.watch(xs)
    y = forall_pME(xs,p=4)
res = y.numpy()
gradients = tape.gradient(y,xs).numpy()
print(res)
print(gradients)
```

```
1.0
[nan nan nan]
```

Source: https://github.com/logictensornetworks

Gradient Problems: Binary Connectives

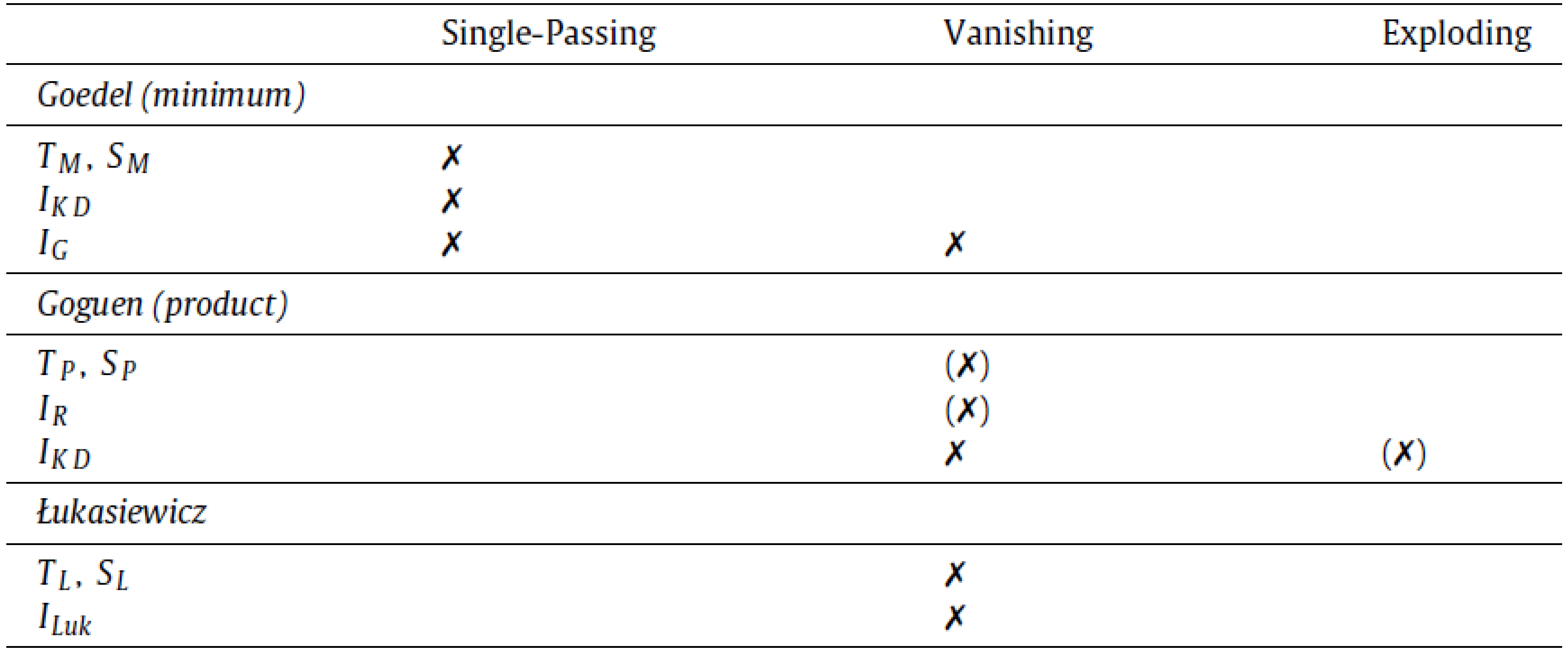


Table C.7
Gradient problems for some binary connectives. (✗) means that the problem only appears on an edge case.

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

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Gradient Problems: Aggregators

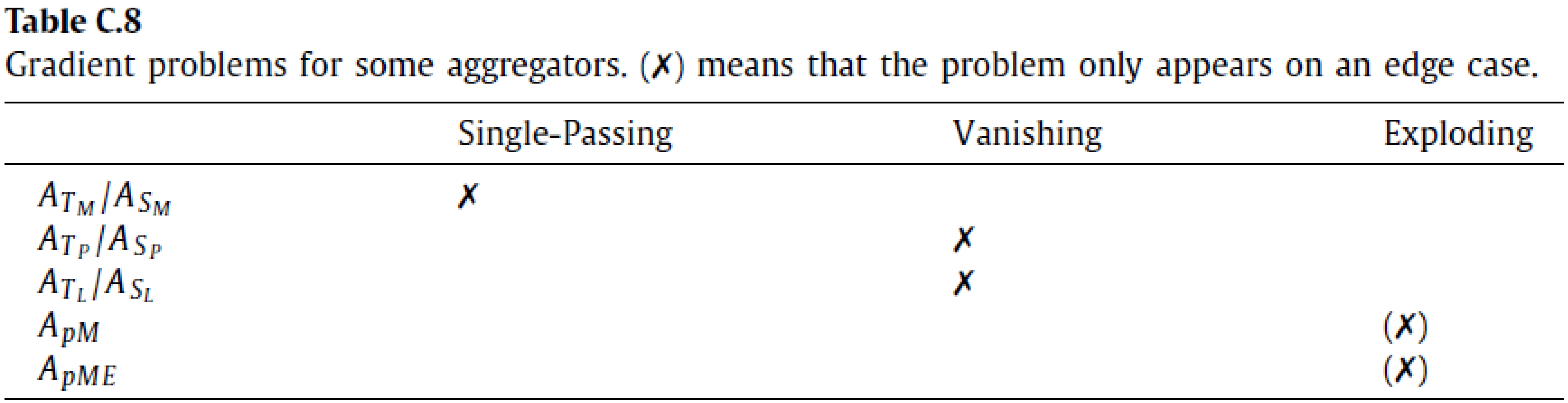


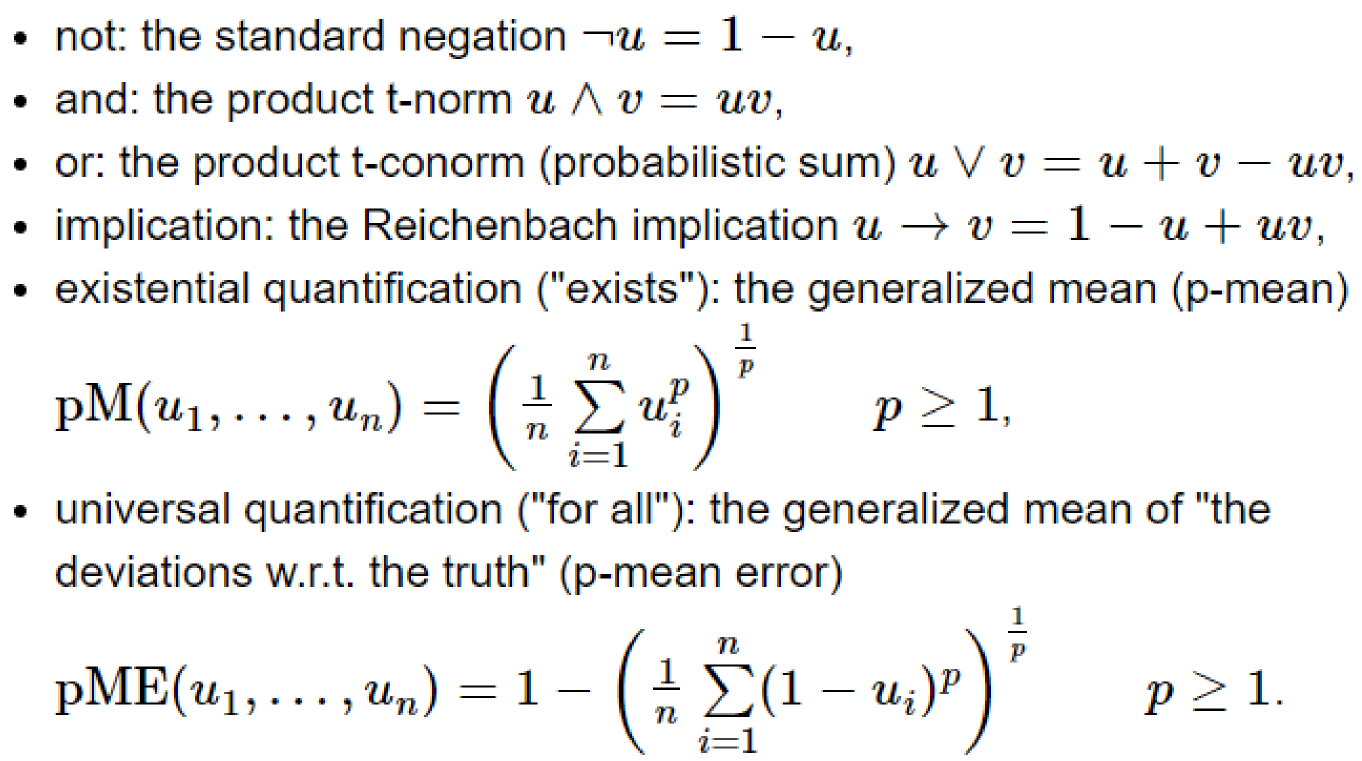
Table C.8
Gradient problems for some aggregators. (✗) means that the problem only appears on an edge case.

	Single-Passing	Vanishing	Exploding
A_{T_M}/A_{S_M}	✗		
A_{T_P}/A_{S_P}		✗	
A_{T_L}/A_{S_L}		✗	
A_{pM}			(✗)
A_{pME}			(✗)

Source: Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

Stable Configuration of Operators

The following is proposed as a stable configuration by Badreddine et al. (2022):



- not: the standard negation $\neg u = 1 - u$,
- and: the product t-norm $u \wedge v = uv$,
- or: the product t-conorm (probabilistic sum) $u \vee v = u + v - uv$,
- implication: the Reichenbach implication $u \rightarrow v = 1 - u + uv$,
- existential quantification ("exists"): the generalized mean (p-mean)

$$\mathrm{pM}(u_1, \ldots, u_n) = \left(\frac{1}{n} \sum_{i=1}^{n} u_i^p \right)^{\frac{1}{p}} \qquad p \geq 1,$$

- universal quantification ("for all"): the generalized mean of "the deviations w.r.t. the truth" (p-mean error)

$$\mathrm{pME}(u_1, \ldots, u_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - u_i)^p \right)^{\frac{1}{p}} \qquad p \geq 1.$$

Sources:

Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

https://github.com/logictensornetworks

Stable Configuration of Operators

Some caveats:

- The product t-norm has vanishing gradients on the edge case $u = v = 0$.
- The product t-conorm has vanishing gradients on the edge case $u = v = 1$.
- The Reichenbach implication has vanishing gradients on the edge case $u = 0, v = 1$.
- p-mean has exploding gradients on the edge case $u_1 = \cdots = un = 0$.
- p-mean error has exploding gradients on the edge case $u_1 = \cdots = un = 1$.

These issues happen on **edge cases** and can be fixed using the following "trick":

- if the edge case happens when an input u is 0, we modify every input with
$$u' = (1 - \epsilon)u + \epsilon$$
- if the edge case happens when an input u is 1, we modify every input with
$$u' = (1 - \epsilon)u$$

where ϵ is a small positive value (e.g., $1\mathrm{e}{-5}$).

Sources:

Badreddine, S., Garcez, A. D. A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. Artif. Intelli., 303, 103649.

https://github.com/logictensornetworks/LTNtorch/blob/main/tutorials/2b-operators-and-gradients.ipynb

Additional Material

Deep Ontological Networks: Deeper Dive

A Deeper Dive: Setup

Recall some **assumptions**:

- **Fixed vocabulary** (everything but the data, including individuals).
- Relations are either **unary** or **binary**.
- Databases can be seen as sets of **triples**:
  - $R(i, j)$ as $\langle i, R, j \rangle$
  - $C(i)$ as $\langle i,$ `MemberOf`$, C \rangle$
- **Negated** facts are simply encoded with a different relation symbol: “$\neg R$” and “$\neg$`MemberOf`”.

A Deeper Dive: Setup

- For every KB, we have an **indicator function**:

$$\mathbb{1}_{KB} : individuals(KB) \to \{-1, 0, 1\}^{|classes(KB)|}$$

$$[\mathbb{1}_{KB}(i)]_\ell = \begin{cases} 1 & \text{if } \langle i, \texttt{MemberOf}, C_\ell \rangle \in D, \\ -1 & \text{if } \langle i, \neg\texttt{MemberOf}, C_\ell \rangle \in D, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

- These vectors summarize all the information about an individual’s class memberships given **explicitly** in the facts.
- The **trained model** over ontology ⍰ is of the form:

$$RRN_\Sigma(D, T) = \mathbb{P}\{T \text{ is true} \mid \langle \Sigma, D \rangle\}$$

where *T* = ⍰*i, R, j*⍰ is an **arbitrary query**.

A Deeper Dive: Setup

The probability distribution on the RHS expresses the model’s **belief** regarding the **truth** of a query:

- Allows for training via cross-entropy error.
- During evaluation, queries are predicted to be true when the probability is 50% or more.
- **Truth:** The authors underscore the use of this term – as opposed to “*D* [illegible]” – since the RRN provides a prediction even if the query is **not provably entailed** by the ontology.

Note: Training can be done using an **actual** DB or, if only the rules are available, by **generating** a DB based on those rules.

A Deeper Dive: Model

Embeddings:

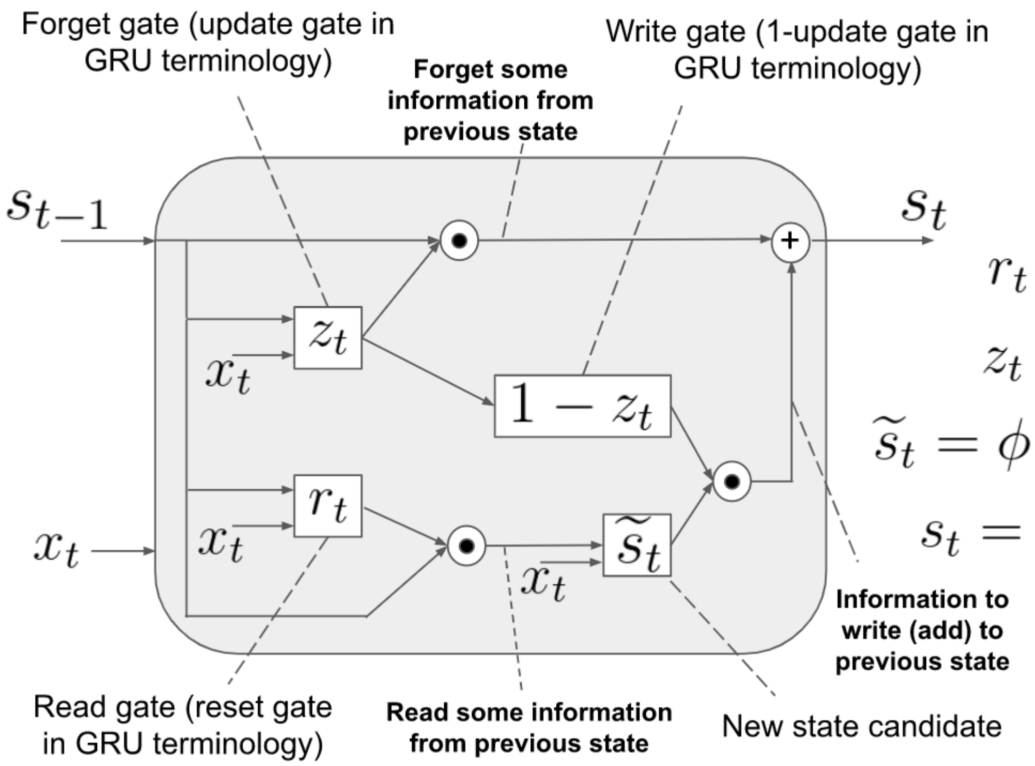
- **Individuals** are represented as unit vectors in $\mathbb{R}^{d}$.
- Hyperparameter *d* to be chosen based on the ontology: expressiveness, vocabulary size, and number of individuals.

Update Layers:

- Need to define two kinds: **relations** and **classes**.
- Relations are in general **not symmetric**, so in total 4 update layers (subject/object and positive/negative).
- The RRN’s recursive layers define a **gated network** (a kind of RNN): Gates control how candidate update steps are applied.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Gated Recurrent Units (GRUs)


Forget gate (update gate in GRU terminology)
Forget some information from previous state
Write gate (1-update gate in GRU terminology)
s_{t-1}
s_t
z_t
x_t
$1 - z_t$
r_t
x_t
$\widetilde{s}_t$
Information to write (add) to previous state
Read gate (reset gate in GRU terminology)
Read some information from previous state
New state candidate

GRU equations	In GRU terminology
$r_t = \sigma(W_r s_{t-1} + U_r x_t + b_r)$	Reset gate
$z_t = \sigma(W_z s_{t-1} + U_z x_t + b_z)$	Update gate
$\widetilde{s}_t = \phi(W\ (r_t \circ s_{t-1}) + U x_t + b)$	State candidate
$s_t = z_t \circ s_{t-1} + (1 - z_t) \circ \widetilde{s}_t$	Current State

deepsystems.ai

Source: https://goo.gl/XodLUU

A Deeper Dive: Model

One-sided updates for relations:

Updating subject's embedding (the one for the object's is analogous)

Logistic function

Gate controlling how much of the update is applied

Model parameters

Calculation of candidate update steps

$$\mathbf{g}^{\triangleleft P}(s,o) = \sigma\Big(\mathbf{V}_1^{\triangleleft P}\mathbf{e}_s + \mathbf{V}_2^{\triangleleft P}\mathbf{e}_o\Big),$$

$$\hat{\mathbf{e}}_s^{(1)} = ReLU\Big(\mathbf{W}_1^{\triangleleft P}\mathbf{e}_s + \mathbf{W}_2^{\triangleleft P}\mathbf{e}_o + \mathbf{e}_s\mathbf{e}_o{}^T\mathbf{w}^{\triangleleft P}\Big),$$

$$\hat{\mathbf{e}}_s^{(2)} = \mathbf{e}_s + \hat{\mathbf{e}}_s^{(1)} \circ \mathbf{g}^{\triangleleft P}(s,o),$$

$$\mathbf{e}_s = Update^{\triangleleft P}(s,o) = \frac{\hat{\mathbf{e}}_s^{(2)}}{\|\hat{\mathbf{e}}_s^{(2)}\|_2}.$$

Simultaneous update operation:

$$\langle \mathbf{e}_s, \mathbf{e}_o \rangle = Update(\langle s, P, o\rangle) = \Big\langle Update^{\triangleleft P}(s,o),\, Update^{P\triangleright}(s,o)\Big\rangle$$

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

A Deeper Dive: Model

- **Classes** are one-sided and we don't need a positive and negative version, so we simply have:

$$\mathbf{g}(i) = \sigma\left(\underset{\uparrow}{\mathbf{V}} \cdot [\mathbf{e}_i : \mathbb{1}_{KB}(i)]\right),$$

Model parameters

$$\hat{\mathbf{e}}_i^{(1)} = ReLU(\underset{\uparrow}{\mathbf{W}} \cdot [\mathbf{e}_i : \mathbb{1}_{KB}(i)]),$$

$$\hat{\mathbf{e}}_i^{(2)} = \mathbf{e}_i + \hat{\mathbf{e}}_i^{(1)} \circ \mathbf{g}(i),$$

$$\mathbf{e}_i = Update(i) = \frac{\hat{\mathbf{e}}_i^{(2)}}{\|\hat{\mathbf{e}}_i^{(2)}\|_2}.$$

Algorithm 1: Generating individual embeddings

Input: an ontological knowledge base $KB = \langle \Sigma, D \rangle$ with $individuals(KB) = \{i_1, i_2, \dots, i_M\}$, a number of update iterations N, and (optionally) a matrix of initial embeddings $\mathbf{E}$.

Output: the generated embeddings $\mathbf{E}$.

```
if $no\ embedding\ matrix\ \mathbf{E}\ was\ provided$ then
    Randomly initialize $\mathbf{E} = [\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \dots, \mathbf{e}_{i_M}]^T$;
end
for $iter = 1, \dots, N$ do
    foreach $i \in individuals(KB)$ do
        $\mathbf{e}_i = Update(i)$;
    end
    foreach $\langle s, P, o \rangle \in F$ $with\ P\ being\ a\ (possibly\ negated)\ relation\ type$ do
        $\mathbf{e}_s, \mathbf{e}_o = Update(\langle s, P, o \rangle)$;
    end
end
return $\mathbf{E}$.
```

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

A Deeper Dive: Prediction

MLPs for computing **predictions**:

- A **single** MLP for all classes:

  $MLP^{(\text{classes})}$ – expects a **single** embedding as input and returns probabilities for the corresponding individual’s membership in each class.

- One for each **relation** R in the vocabulary:
  - $MLP^{(R)}$ – expects a **pair** of embeddings and provides a probability for the relation to hold between the corresponding individuals.
  - The **negation** of the relation is simply computed as 1 ⍰ $MLP^{(R)}(\mathbf{e}_i, \mathbf{e}_j)$

A Deeper Dive: Training

- **Training** is done via samples of databases (sets of facts) that are **associated** with an ontology.
- Recall that only DBs are used – not rules – in training.
- **Sampling** can thus be used to generate DBs for training:
    - Allows for a balanced use of rules
    - Training can be done on facts that only appear as inferences and not as facts in the DB.
- Though a single training example is considered at a time, since samples are DBs, this corresponds to **minibatches**.
- The training procedure itself is straightforward – an overview is included in the following slide.

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.

Algorithm 2: RRN Training

```
Input: a sequence of training samples $T = \langle KB_1, KB_2, \dots, KB_M \rangle$, where $KB_\ell = \langle \Sigma, D_\ell \rangle$, and
       a number of update iterations $N$.
while evaluation error has not converged do
    Randomly shuffle $T$;
    for $KB_\ell \in T$ do
        Generate embedding matrix $\mathbf{E}$ for $KB_\ell$;
        Compute cross-entropy loss for predicting triples in $KB_\ell$ from $\mathbf{E}$ (both facts and
          inferences);
        Update model parameters to minimize the loss;
    end
end
```

Source: Hohenecker, P., & Lukasiewicz, T. (2020). Ontology reasoning with deep neural networks. JAIR, 68, 503-540.