Ontology as neuronal-space manifold: Towards symbolic and numerical artificial embedding

Chloé Mercier, Hugo Chateau-Laurent, Frédéric Alexandre, Thierry Viéville

Ontology as neuronal-space manifold: Towards symbolic and numerical artificial embedding *

Chloé Mercier[1] , **Hugo Chateau-Laurent**[1] , **Frédéric Alexandre**[1] , **Thierry Viéville**[1]
[1]Mnemosyne Team, Inria Bordeaux, LaBRI and IMN
firstname.lastname@inria.fr

Abstract

Some human cognitive tasks may involve tightly interleaved logical and numerical computations. On the one hand, ontologies allow us to describe symbolic structured knowledge and perform logical inference, providing a rather natural representation of human reasoning as modeled in cognitive psychology. On the other hand, spiking neural networks are a biologically plausible implementation of processing in brain circuits, yet they process numeric vectors rather than symbolic data. Unifying these symbolic and sub-symbolic approaches is still a wide and open question, and the Semantic Pointer Architecture (SPA) based on the Vector Symbolic Architecture (VSA) provides a way to manipulate symbols embedded as numeric vectors that carry semantic information.

In this paper, as a step towards filling the symbolic/numerical gap, we propose to map an ontology onto a SPA-based architecture with a preliminary partial implementation into spiking neural networks. More specifically, we focus on ontology standards used in the semantic web such as Resource Description Framework [Schema] (RDF[S]) and the Web Ontology Language (OWL). We provide a detailed implementation example in the case of specific RDFS entailments based on predicate chaining. To that end, we used the neural simulator Nengo with two associative memories in interaction, the first one storing assertions and the second one storing entailment rules. Reporting interesting formal results, our embedding enjoys intrinsic properties allowing semantic reasoning through distributed numerical computing. This original preliminary work thus combines symbolic and numerical approaches for cognitive modeling, which might be useful to model some complex human tasks such as ill-defined problem-solving, involving neuronal knowledge manipulation.

Keywords: Ontology, Resource Description Framework, Vector Symbolic Architecture, Semantic Pointer Architecture, Neural Engineering Framework, Neurosymbolism.

1 Introduction

Artificial Intelligence (AI) has recently made significant strides, both in numerical approaches and symbolic approaches; the former is based on numerical computation and includes, among others, neural networks or Bayesian inference, while the latter manipulates knowledge bases such as used in expert systems or the semantic web. Accordingly, large AI systems with impressive capabilities have been proposed. Nonetheless, the AI field is still lacking unified systems integrating both approaches (Sun and Alexandre 2013), although the last decade has seen the emergence of a new "neurosymbolic" wave aiming to make neural networks perform logical reasoning (Garcez and Lamb 2020).

It is generally considered that human cognitive capabilities are not within the reach of either purely symbolic or purely numerical artificial intelligence because of the limitations of each approach. In hybrid systems, both symbolic and numerical components are involved (Lallement, Hilario, and Alexandre 1995), but their contributions are usually kept separate in distinct aspects of the system. This is the case for example with ontology-based deep learning (e.g., (Phan et al. 2017; Petrucci, Ghidini, and Rospocher 2016; Hohenecker and Lukasiewicz 2020)) or "black-box" cooperation between deep-networks and ontology reasoners (e.g., (Ayadi et al. 2019; Jiménez, Elizalde, and Raj 2018)). However, geometric mapping of ontologies onto Euclidean spaces or manifolds (e.g., (Eidoon, Yazdani, and Oroumchian 2008; Tous and Delgado 2006; Xiao, Huang, and Zhu 2015)) allows to perform reasoning at both a symbolic and to a limited extent numerical level, while the link between neural network representation and manifolds is well established and understood (e.g., (Chui and Mhaskar 2018; Zhu et al. 2017)).

In some cases, there is a need for a full integration of both paradigms, as allowed by a unified system, because the underlying task corresponds to a tight interaction between the properties of the paradigms, which cannot be achieved with separate modules. This is the case with the topic we consider in our group, that is human learning. In a recent development (Mercier et al. 2021), we propose to formalize a problem-solving task, where children have to manipulate and assemble objects to answer instructions given by the teacher (Romero, David, and Lille 2019). It has been reviewed in the previous reference that the underlying cognitive functions are as diverse as sensorimotor skills, exploitation of explicit and sometimes partial knowledge, hypothesis generation and creativity.

In the case of such tasks with so tightly interleaved symbolic and numerical components, some authors propose that a purely numerical, neuronal approach could implement the

*Supported by Inria, AEx AIDE https://team.inria.fr/mnemosyne/en/aide.

needed unified system, with an obvious reference to the brain as an example of a neuronal system manipulating symbolic information with units generally considered as numerical systems, especially to implement high level cognitive functions (see, e.g., (Pulvermüller 2013) for a general review). Yet it is a wide and still open issue to establish the numerical primitives required for modeling symbolic representation and manipulation (see (Alexandre 2019) for a general discussion). In the last few years, models using numerical mechanisms to process symbolic information such as logical computation (Shi et al. 2020) or reasoning (Riegel et al. 2020) have flourished.

There are numerous approaches to vectorize graph data models, such as translational distance models or random walk based methods (see (Wang, Qiu, and Wang 2021) for a recent review in link with approximate statistical reasoning, or (Sajjad, Docherty, and Tyshetskiy 2019) regarding representation learning from data) including taking into account dynamic knowledge such as (Sauerwald and Zanetti 2019), including approaches combining vector representations and inference rules such as (Guo et al. 2016). There are also many real valued first order logic (e.g., probabilistic) approaches integrated in or interfaced with machine learning algorithms (see (Garcez and Lamb 2020) for a general review and, e.g., (Cohen, Yang, and Mazaitis 2017) for an example). Our approach is complementary in the sense that we focus on both a biologically plausible neuronal framework, and a symbolic formalism based on *triples* (as defined in 2.4) that seems appropriate to model human reasoning in psychology (McClelland and Rogers 2003).

Specific questions may arise when trying to integrate for example relations between ontology specifications and partial knowledge (Tettamanzi, Zucker, and Gandon 2017), allowing one to consider not only true or false knowledge, but relative degree of truth, as taken into account here. Most of machine learning algorithms represent such "partial truth" with only probability. However, the human "level of truth" seems to be different and related to other notions such as possibility and necessity, related to a given modality, that is a context, a given time, and so on, which is also considered as representative to what is modeled in educational science and philosophy (see (Smith 1994) while (Rusawuk 2018) proposes a discussion).

As a possible entry to a unified approach, Vector Symbolic Architectures (VSA) were introduced as a way to manipulate symbolic information represented as numeric vectors (see e.g. (Levy and Gayler 2008) for an introduction). VSAs have been proven helpful to model high-level cognition and account for multiple biological features (Gayler 2003; Eliasmith 2013). More specifically, the Semantic Pointer Architecture (Eliasmith 2013) instantiates so-called semantic pointers (i.e. vectors that carry semantic information) and their manipulation in networks of spiking neurons. This approach makes a significant step towards the unification of symbolic and sub-symbolic processing in that it provides a way to translate the former into the latter. Consequently, complex knowledge representation in the form of compositional structures that are traditionally restricted to symbolic approaches can now be distilled in numerical and even neural systems (Crawford, Gingerich, and Eliasmith 2016).

On the other hand, considering knowledge representation and reasoning, the capabilities of Semantic Web modeling languages, such as RDFS (Resource Description Framework Schema) and OWL (Web Ontology Language) (see, e.g. (Allemang, Hendler, and Gandon 2020) for a recent didactic reference) is a rather accessible and very powerful way of solving modeling problem and manipulate high-level data representation. To what extent could such mechanism be biologically plausible ? In order to contribute to this issue, we show here that suitable design choices allow us to make explicit how to implement a RDFS[1] specification using the Semantic Pointer Architecture. We illustrate this on a simple example and also discuss to which extent OWL specification could benefit from the same method.

2 Basic design choices

2.1 From symbols to numbers

In order to represent symbolic information, we use RDFS to structure knowledge representation. It is based on the RDF data model, which represent knowledge as triples, as made explicit in section 2.4. More precisely, the *universe of discourse* is made of *resources*, referenced by some universal resource identifier (IRI), i.e. a fixed lexical token. To structure this universe of discourse, we consider:

(i) *individuals* that refer to real-world concrete or abstract objects, or

(ii) *literals* to characterize individuals using data attributes, i.e., numerical values, character strings, or any structured information such as dates

(iii) *concepts* and *roles* (namely *classes* and *properties*) that allow to structure the knowledge about individuals.

Before going further, we can point out that the present definition follows the RDF/RDFS framework, with the following variants:

- we conflate *name* with both IRI and blank node, since on the one hand blank node can be eliminated,[2] and on the other hand because we only process the information locally at this stage, thus avoiding considering all issues regarding distributed information between different sources;
- we do not consider (i) semantic web specific literal (e.g., `rdf:XMLLiteral`), or (ii) utility and annotation or other human-targeted properties (e.g., `rdfs:seeAlso`) at this stage;
- we will introduce both containers, i.e., ordered or unordered sequences, and collections, i.e., chained lists, later in these specifications, but in a somehow different form, adapted to the numerical representation and obvious to map on RDF representations;

[1] According to the https://www.w3.org/TR/rdf-schema specification.

[2] Using a standard process related to skolemisation.

- we do not consider all XSD data-types, but will introduce a precise notion of numerical values and will detail how to represent structured data in our framework.

At the numerical level, each resource is implemented as a randomly drawn fixed unit d-dimensional vector, $\mathbf{x} \in \mathcal{R}^d$, and the key idea is to study to what extent symbolic reasoning on resources may correspond to algebraic operations implemented via numerical computations (that we make explicit in 3.1., based on a framework introduced in (Eliasmith 2013)). Typically $d \simeq 100 \cdots 1000$ and we expect to manipulate $k \simeq 100 \cdots 10000$ resources.

A similarity measure is now introduced in order to semantically compare two vectors.

2.2 Semantic similarity

Classically, the cosine similarity (i.e., normalized dot product, denoted $\cdot$) is used to compute the semantic similarity between two unit vectors:

$$\mathbf{x} \cdot \mathbf{y} \stackrel{\text{def}}{=} \mathbf{x}^\top \mathbf{y}$$

where $\mathbf{x}^\top$ denotes the transpose of $\mathbf{x}$.

The key property is that, provided that the space dimension d is large enough, two randomly chosen different vectors will be approximately orthogonal. More precisely,

$$\mathbf{x} \cdot \mathbf{y} \sim \mathcal{N}(0, O(1/d)),$$

i.e., follows a centered normal distribution (Schlegel, Neubert, and Protzel 2020), while by construction $\mathbf{x} \cdot \mathbf{x} = 1$.

Based on this, most VSA approaches consider that 2 vectors **x** and **y** are semantically equivalent when this similarity τ equals to 1, but with different ways to interpret the result:
- *Closed-world reasoning*: Anything, that cannot be stated as true is false, thus $\tau \in \{0, 1\}$, often obtained by projection and rectification (to avoid negative values). This is the most common interpretation in VSAs.
- *Open-world reasoning*: Anything might be true unless it can be proven false, prompting a need to characterize unknown statements; furthermore the notion of negation is either not defined (as in the RDFS model yielding monotonic reasoning only), or defined at a higher level (as in OWL and more generally in description logics) that we also consider as a perspective of this work. Here we enrich the notion of being either false or true, by a numeric representation of partial knowledge, as illustrated in Fig 1. The true value corresponds to 1 (fully possible and fully necessary), the false value to -1 (neither possible nor necessary, i.e., impossible) and the unknown value to 0, which corresponds to a fully possible but absolutely not necessary value. This representation has been designed to be compatible with the ternary Kleene logic, beside being also coherent with respect to the possibility theory[3] (not developed here, please refer to (Denœux, Dubois, and Prade 2020) for a general introduction). This deterministic representation of partial knowledge can be generalized in order to also include a probabilistic representation (using a 2D representation), although we will not develop this aspect any further.

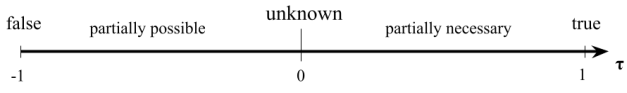

Figure 1: Representation of partial truth $\tau \in [-1, 1]$, in link with necessity and possibility.

2.3 Classes and approximate Boolean properties

The first kind of concept to structure the knowledge is the hierarchical notion of *class*, which is equivalent to the notion of Boolean property, defining the class of all individuals enjoying (or not) this property, and defining the property that an individual belongs (or not) to a given class.

In RDFS this translates into, given a scoring τ as introduced previously: `x rdf:type c` $\triangleright \tau$

We assume in the following that classes and individuals are in disjoint semantic sets. Although this is not necessarily the case in RDFS, this assumption is compatible with description logics (i.e. the OWL-DL level of specification) we target to use in future developments, and ensures to avoid self-reference paradoxes.

At the numeric level, they both correspond to unit vectors, with the following interpretation:
- For a given vector $\mathbf{x}$ encoding an individual, the vector $\mathbf{c} = \mathbf{x}$ corresponds to the singleton class $\mathcal{C} = \{\mathbf{x}\}$.
- For a given class vector $\mathbf{c}$ the individual vector $\mathbf{x} = \mathbf{c}$ can be interpreted as a "prototype" for this class.

At a geometric level, this can be interpreted as covariant/contravariant duality, and the similarity between a class covariant vector $\mathbf{c}$ and an individual contravariant vector $\mathbf{x}$ may be interpreted as the fact this individual approximately belongs, or not, to the class.

2.4 Statements: facts and rules

In RDF, the knowledge about the universe of discourse is structured into statements of the form `subject predicate object`, called *triples*, where `subject` and `object` are two resources linked by a relationship (property) explicitized by the `predicate`. We introduce *weighted triples* of the form:

`subject predicate object` $\triangleright \tau$

with a value τ as discussed previously, which generalizes usual RDF statements, introducing a scoring value (refer to (Tettamanzi, Zucker, and Gandon 2017) for an introduction and a recent literature review). This is easily stated in the RDF language itself, using reification, while in our case, it is an intrinsic feature of our design. More precisely, when considering RDFS where only true (but not false) assertions can be stated, we have $\tau \in [0, 1]$, while when generalizing to the OWL language where negation can be stated we will use

[3] To make the link explicit, given necessity ν and possibility π, with $\nu \leq \pi$ by construction, while $\nu > 0 \Rightarrow \pi = 1$ and $\pi < 1 \Rightarrow \nu = 0$, we have the one to one correspondence with our representation using $\tau \in [-1, 1]$:

$$\tau \stackrel{\text{def}}{=} \nu + \pi - 1 \text{ with } \left\{ \begin{array}{lcl} \pi & = & H(-\tau)(1-\tau) \\ \nu & = & H(\tau)\,\tau, \end{array} \right.$$

$\tau \in [-1, 1]$, this setting being compatible with both levels of specification.

It is worth noting that such statements are of three kinds, although this is not explicit in the model

`entity data-property literal` $\triangleright \tau$

defines *attributes* allowing to specify some entity attributes;

`entity object-property entity` $\triangleright \tau$

defines *relations* between entities;

`concept predicate resource` $\triangleright \tau$

defines *meta-properties* about classes or properties.

3 Ontology numerical mapping

3.1 A Semantic Pointer Triplestore

Let us suppose we need to represent the following weighted statements:

$\texttt{subject}_1\ \texttt{predicate}_1\ \texttt{object}_1 \triangleright \tau_1$
$\texttt{subject}_2\ \texttt{predicate}_2\ \texttt{object}_2 \triangleright \tau_2$
$\texttt{subject}_2\ \texttt{predicate}_3\ \texttt{object}_3 \triangleright \tau_3$

(note that the same subject $\texttt{subject}_2$ is used in the two last statements).

The first step towards a Semantic Pointer representation is to encode each resource by a randomly sampled vector on the unit hypersphere of our d-dimensional vector space: let us denote $\mathbf{s_i}, \mathbf{p_i}, \mathbf{o_i}$ the respective vector representations of the resources $\texttt{subject}_i, \texttt{predicate}_i, \texttt{object}_i$.

Next, we may store each of these statements into an associative memory (Stewart, Tang, and Eliasmith 2011), similar to a hash table, where each "key" would be a resource and the corresponding value would express the statements for which this resource is a subject:[4]

$$\mathbf{s_1} \rightarrow \tau_1 \mathcal{B}(\mathbf{o_1}, \mathbf{p_1})$$

$$\mathbf{s_2} \rightarrow \tau_2 \mathcal{B}(\mathbf{o_2}, \mathbf{p_2}) + \tau_3 \mathcal{B}(\mathbf{o_3}, \mathbf{p_3})$$

for which we need to introduce the following operations:

1. A scalar multiplication $\mathbf{a} = \tau \mathbf{b}$ that scales a vector $\mathbf{b}$ by a factor of τ and preserves its direction.
2. A vector superposition (e.g. element-wise addition) $\mathbf{a} = \mathbf{b} + \mathbf{c}$ that results in a vector $\mathbf{a}$ with $\mathbf{a} \cdot \mathbf{b} = 1$ and $\mathbf{a} \cdot \mathbf{c} = 1$, assuming $\mathbf{b}$ and $\mathbf{c}$ are orthonormal.
3. A binding operation $\mathbf{a} = \mathcal{B}(\mathbf{b}, \mathbf{c})$ that outputs a vector $\mathbf{a}$ that is not collinear with either $\mathbf{b}$ or $\mathbf{c}$.

Let us discuss in detail this last ingredient. The Semantic Pointer Architecture (SPA) developed by Eliasmith et al(Eliasmith 2013) implements such operations. This cognitive architecture builds upon a particular case of Vector Symbolic Architecture (VSA) and the Neural Engineering Framework (NEF) (Eliasmith and Anderson 2002). The NEF provides a set of principles for representing vectors into biologically plausible networks of neurons, and implementing the desired transformations through synaptic connections (see (Eliasmith and Anderson 2002) for technical details). This framework (including both NEF and SPA) is already implemented into a simulator called Nengo (Bekolay et al. 2014).

[4] In RDF/RDFS, this kind of database is known as a triplestore, as it stores statements also referred to as triples.

Several choices are available for the binding operation. The most classical one is the circular convolution $\mathbf{a} = \mathbf{b} \circledast \mathbf{c}$, which is used in Holographic Reduced Representations (HRR) (Plate 1995). This operation is commutative, associative and distributive. In order to retrieve an element vector $\mathbf{b}$ from the resulting vector $\mathbf{a}$, the circular convolution can be used with the approximate convolutive inverse of the other element ($\mathbf{c}^{-1}$): $\mathbf{b} \approx \mathbf{a} \circledast \mathbf{c}^{-1}$.

However, commutativity and associativity could lead to serious misunderstandings. Let us illustrate this point with an example:

"Luigi eats this Pizza" and "this Pizza has a topping of Mozzarella", encoded as:

Luigi $\rightarrow$ eats $\circledast$ Pizza (1)

Pizza $\rightarrow$ hasTopping $\circledast$ Mozzarella (2)

By injecting the Pizza subject from the statement (2) into the Pizza object in the statement (1), we can infer the new statement:

Luigi $\rightarrow$ eats $\circledast$ (Pizza $\circledast$ (hasTopping $\circledast$ Mozzarella) (3)

Under this form, (3) may be interpretated as "Luigi eats this Pizza which has a topping of Mozzarella", but the right member can be rewritten as:

(3r) = eats $\circledast$ Pizza $\circledast$ hasTopping $\circledast$ Mozzarella *(associativity)*

= hasTopping $\circledast$ Mozzarella $\circledast$ eats $\circledast$ Pizza *(commutativity)*

interpreted as "Luigi has a topping of Mozzarella which eats a Pizza". Note that circular references are easily managed with this setup, thanks to the binding mechanism generating always an almost orthogonal combined vector.

Instead, we will use another binding operation offered by the Vector-derived Transformation Binding (VTB) algebra, described by Gosmann and Eliasmith in (Gosmann and Eliasmith 2019), which is also implemented in Nengo and is neither commutative nor associative, but distributive and bilinear.

3.2 Using the VTB algebra

The binding operation

With d the dimensionality of the vector space, which the VTB requires to be square, and $d'^2 = d$, the binding operation is defined, following (Gosmann and Eliasmith 2019):

$$\mathcal{B}(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \mathbf{B_y}\, \mathbf{x}$$

where $\mathbf{B_y}$ is block-diagonal matrix defined as

$$\mathbf{B_y} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{B'_y} & 0 & \dots & 0 \\ 0 & \mathbf{B'_y} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{B'_y} \end{bmatrix}$$

where

$$\mathbf{B'_y} = d^{\frac{1}{4}} \begin{bmatrix} y_1 & y_2 & \dots & y_{d'} \\ y_{d'+1} & y_{d'+2} & \dots & y_{2d'} \\ \vdots & \vdots & \ddots & \vdots \\ y_{d-d'+1} & y_{d-d'+2} & \dots & y_d \end{bmatrix}$$

With these notations, the first relation above becomes

$$\mathbf{s_1} \to \tau_1 \, \mathbf{B_{p_1}} \, \mathbf{o_1}$$

This design choice enjoys the following interesting properties:
- The relation is not commutative in $\mathbf{x}$ and $\mathbf{y}$, in the general case, as required.
- The relation is bilinear in $\mathbf{x}$ and $\mathbf{y}$ (this is obvious for the former, and easily verified on the matrix form for the latter).
- The relation is approximately left and right invertible as discussed now.

The right unbinding operation

In order to retrieve an element vector from the bound vector, we need to bind it to the inverse of the other element vector. Unlike HRR and the circular convolution, the VTB algebra does not provide two-side inverses: there is no left inverse for VTB.

The right approximate inverse $\mathbf{y}^\sim$ must enjoy the following property:

$$\forall \mathbf{x},\ \mathcal{B}(\mathcal{B}(\mathbf{x}, \mathbf{y}), \mathbf{y}^\sim)) = \mathbf{B_{y}}_\sim \, \mathbf{B_y} \, \mathbf{x} = \mathbf{x}.$$

The key point here is that since the coefficients of $\mathbf{y}$ are chosen randomly and independently identically distributed, the matrix $\mathbf{B_y}$ is almost orthogonal, thus:

$$\mathbf{B_y}^\top \simeq \mathbf{B}_{\mathbf{y}^\sim},$$

which corresponds to simply permuting the elements of $\mathbf{y}$.

Consequently, the right identity vector $\mathbf{i_B}$ such that $\mathbf{B_{i_B}} = \mathbf{I}$, writes explicitly:

$$[\mathbf{i_B}]_i = \begin{cases} d^{-\frac{1}{4}} & \text{if } i = (k-1)\,d' + k, 0 < k \leq d' \\ 0 & \text{otherwise.} \end{cases}$$

where $[\mathbf{i_B}]_i$ stands for the i-th coordinate of the vector. We get i_B by "unfolding" the identity matrix $I_{d'}$ line by line and concatenating it d' times (and adjusting the resulting vector by a $d^{-\frac{1}{4}}$ factor).

This allows us to check if a property is a predicate for a given resource and, if so, retrieve the corresponding object (thus performing *right unbinding*):

$$\mathbf{x} \to \mathbf{z} = \mathcal{B}(\mathbf{y}, \mathbf{p}) = \mathbf{B_p} \, \mathbf{y}$$

if and only if $\mathbf{B_p}^\top \mathbf{z} = \mathbf{B_p}^\top \mathbf{B_p} \, \mathbf{y} = \mathbf{y}$ and $\mathbf{y}$ belongs to the vocabulary. Specifically, "$\mathbf{y}$ belongs to the vocabulary" means that there exists a vector $\mathbf{v}$ in the vocabulary such that $\mathbf{v} \cdot \mathbf{y} \simeq 1$. Otherwise the result is simply undefined, leading to the conclusion that the predicate is undefined for the given resource.

The left unbinding operation

A step further, we may also want to check if two resources are linked to each other by a predicate and, if so, retrieve the corresponding property (referred to as *left unbinding*). In order to do that, we first need to "flip" the order of the terms involved in the binding. This is achievable by performing the following operation:

$$\mathcal{B}(\mathbf{a}, \mathbf{b}) = \mathbf{B}_\leftrightarrow \, \mathcal{B}(\mathbf{b}, \mathbf{a})$$

considering the matrix $\mathbf{B}_\leftrightarrow$ defined as:

$$[\mathbf{B}_\leftrightarrow]_{ij} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } j = 1 + \left\lfloor \dfrac{i-1}{d'} \right\rfloor + d'[(i-1) \mod d'] \\ 0 & \text{otherwise.} \end{cases}$$

Thanks to this, we obtain:

$$\mathbf{x} \to \mathbf{z} = \mathcal{B}(\mathbf{y}, \mathbf{p}) = \mathbf{B_p} \, \mathbf{y}$$

if and only if $\mathbf{B_y}^\top \mathbf{B}_\leftrightarrow \mathbf{z} = \mathbf{B_y}^\top \mathbf{B_y} \, \mathbf{p} = \mathbf{p}$ and $\mathbf{p}$ belongs to the vocabulary.

4 Relationship and membership composition

4.1 Hierarchical representations

RDF/RDFS provides two preset properties to account for hierarchical representations between individuals and classes:

- `x rdf:type c`, expresses that the individual `x` belongs to the class `c`.
- `c rdfs:subClassOf c'` expresses that the class `c` is a subclass of the class `c'`, or in other words, that the concept `c` is included in the class `c'`. This means that all individuals that belong to `c` automatically belong to `c'`.

Consequently, some entailment rules [5] can be used to disclose implicit memberships:

- inheritance (referred to as *rdfs9* by the W3C):

`x rdf:type c` $\wedge$ `c rdfs:subClassOf c'`
$\Longrightarrow$ `x rdf:type c'`

- transitivity of `rdfs:subClassOf` (*rdfs11*):

`c rdfs:subClassOf c'` $\wedge$ `c' rdfs:subClassOf c''`
$\Longrightarrow$ `c rdfs:subClassOf c''`

The Semantic Pointer representation that we proposed earlier, using the VTB algebra, allows us to implement such rules.

For example, the inheritance rule can be expressed as follows: $\mathbf{x} \to \mathbf{B_{type}} \, \mathbf{c}$ and $\mathbf{c} \to \mathbf{B_{subClassOf}} \, \mathbf{c'}$ implies

$$\mathbf{x} \to \mathbf{B_{type}} \, \mathbf{c} + \mathbf{B_{type}} \, \mathbf{B_{subClassOf}} \, \mathbf{c'} + \mathbf{B_{type}} \, \mathbf{c'}$$

A sufficient condition for this implication to be true would be $\mathbf{B_{type}} \, \mathbf{B_{subClassOf}} = \mathbf{B_{type}}$, which would mean $\mathbf{subClassOf} = \mathbf{i_B}$; that may not be desirable as it would lead to many wrongly inherited memberships.

Instead, we prefer using a second associative memory to encode entailment rules such as $\mathbf{B_{type}} \, \mathbf{B_{subClassOf}} \to \mathbf{B_{type}}$, as schematized in Fig.3. However, we cannot store matrices in the associative memory, only vectors. But we can observe that both of these matrices are block-diagonal, and of the form

$$\mathbf{B_z} = \begin{bmatrix} \mathbf{B_z}' & 0 & 0 \\ 0 & \mathbf{B_z}' & 0 \\ 0 & 0 & \mathbf{B_z}' \end{bmatrix},$$

thus correspond to a given vector $\mathbf{z}$. We thus may simply store the vectors $\mathbf{z}$ into the associative memory. In this particular case, we would therefore store the association:

$$\mathbf{type} \oslash \mathbf{subClassOf} \to \mathbf{type}$$

where we introduce a *vector composition* operator $\oslash$ defined

[5] A recapitulative table of RDFS entailment rules as defined by the W3C can be found at https://www.w3.org/TR/2004/REC-rdf-mt-20040210

as: $z = a \oslash b$ such that:

$$\mathbf{B_z}' = d^{\frac{1}{4}}\, \mathbf{B_a}'\, \mathbf{B_b}',$$

so that $\mathbf{B_z} = \mathbf{B_a}\, \mathbf{B_b}$.

Similarly, we can express the transitivity of `c rdfs:subClassOf c'`:

$$\mathbf{c} \rightarrow \mathbf{B}_{\textbf{subClassOf}}\, \mathbf{c}'$$

and

$$\mathbf{c}' \rightarrow \mathbf{B}_{\textbf{subClassOf}}\, \mathbf{c}''$$

implies

$$\mathbf{c} \rightarrow \mathbf{B}_{\textbf{subClassOf}}\, \mathbf{c}' + \mathbf{B}_{\textbf{subClassOf}}\, \mathbf{B}_{\textbf{subClassOf}}\, \mathbf{c}'' + \mathbf{B}_{\textbf{subClassOf}}\, \mathbf{c}''$$

As a consequence, the inference rule will be stored as:

$$\textbf{subClassOf} \oslash \textbf{subClassOf} \rightarrow \textbf{subClassOf}$$

Then, by iterating on such inference rule, we easily obtain the transitive closure of properties inferred from `rdf:type` and `rdfs:subClassOf` statements. At the distributed implementation level, this means using a loop between the second associative memory encoding inference rules and the first associative memory encoding the input statements, as schematized in Fig.3, on the last page. This general principle allows the system to compute the closure of the reasoning rules. i.e., fixed point iteration, adding any new statement derived from the application of the rules, until a fixed point is reached.

Let us now discuss, to which extents we can generalize the present formalism, to any property.

4.2 Relational representations

The previous mechanism allows us to express any entailment of the form:

$$(\texttt{x p y}) \wedge (\texttt{y p' z}) \implies (\texttt{x p'' z})$$

by storing the rule

$$\textbf{p} \oslash \textbf{p'} \rightarrow \textbf{p''}$$

thus implemented by an operator product, for some predicate **p**, **p'**, **p''**. In other words, we may express any entailment based on **predicate chaining**.

A step further, we need to implement property restrictions, i.e. at the RDFS level, property range `rdfs:range` and domain `rdfs:domain`, e.g., inference of the form:

$$(\texttt{x p y}) \wedge (\texttt{p rdfs:domain c}) \implies (\texttt{x rdf:type c})$$

and

$$(\texttt{x p y}) \wedge (\texttt{p rdfs:range c}) \implies (\texttt{y rdf:type c})$$

and subproperty hierarchy of the form:

$$(\texttt{x p y}) \wedge (\texttt{p rdfs:subPropertyOf q}) \implies (\texttt{x q y}).$$

for the *rdfs2*, *rdfs3* and *rdfs7* entailment rules, respectively.

These rules belong to Description Logic Programming (DLP), roughly speaking at the intersection of OWL and Logic Programming with Horn clauses (Grosof et al. 2003), allowing to implement the complete RDFS language as targeted here and beyond an interesting part of OWL axioms (Levesque 1986).

In order to derive such generalization, we need non trivial algebra. Let us rewrite, for a given triple:

$$(\texttt{x p y}) \text{ i.e., } \mathbf{x} = \mathbf{B}_p\, \mathbf{y}$$

using tensorial notations:

$$\mathbf{x} = {}^{po}_{s}\mathbf{B}\, \mathbf{p}\, \mathbf{y}, \quad \mathbf{p} \simeq {}^{so}_{p}\mathbf{B}\, \mathbf{x}\, \mathbf{y}, \quad \mathbf{y} \simeq {}^{sp}_{o}\mathbf{B}\, \mathbf{x}\, \mathbf{p},$$
$$\mathbf{x} = {}^{op}_{s}\mathbf{B}\, \mathbf{y}\mathbf{p}, \quad \mathbf{p} \simeq {}^{os}_{p}\mathbf{B}\, \mathbf{y}\, \mathbf{x}, \quad \mathbf{y} \simeq {}^{ps}_{o}\mathbf{B}\, \mathbf{p}\, \mathbf{x},$$

these notations being well defined because, as stated in the previous section, since (i) the operator is bi-linear, (ii) the $\mathbf{B}_{\leftrightarrow}$ operator allows to swap property and subject, (iii) the operator is approximately invertible, while we also can define:

$$\mathbf{x} = {}^{p}_{s}\mathbf{B}\, \mathbf{p}, \quad \mathbf{p} \simeq {}^{s}_{p}\mathbf{B}\, \mathbf{x}, \quad \mathbf{y} \simeq {}^{s}_{o}\mathbf{B}\, \mathbf{x},$$
$$\mathbf{x} = {}^{o}_{s}\mathbf{B}\, \mathbf{y}, \quad \mathbf{p} \simeq {}^{o}_{p}\mathbf{B}\, \mathbf{y}, \quad \mathbf{y} \simeq {}^{p}_{o}\mathbf{B}\, \mathbf{p},$$

introducing projection (e.g., $\mathbf{x} = {}^{p}_{s}\mathbf{B}\, \mathbf{p}$ simply states that the subject $\mathbf{x}$ has a property $\mathbf{p}$ for some unknown object value, and so on). Explicitly deriving these expressions is a perspective of the present preliminary work.

If we are able to explicitly compute the previous forms of the $\mathbf{B}$ which exist, thanks to the stated algebraic properties, it would allow us to translate, at least the three previous inference rules at the numerical level. However, in this case, at the implement level, we do not set a single value to accumulate all possible derived results. Given this precision, since:

$$\mathbf{x} = {}^{p}_{s}\mathbf{B}\, \mathbf{p}, \ \mathbf{p} = {}^{po}_{s}\mathbf{B}\, \mathbf{domain}\, \mathbf{c}$$

we may implement

$$\mathbf{x} = {}^{p}_{s}\mathbf{B}\, {}^{po}_{s}\mathbf{B}\, \mathbf{domain}\, \mathbf{c}$$

while $\mathbf{x} = {}^{po}_{s}\mathbf{B}\, \mathbf{type}\, \mathbf{c}$, for any $\mathbf{x}$ and any $\mathbf{c}$, we derive:

$${}^{p}_{s}\mathbf{B}\, {}^{po}_{s}\mathbf{B}\, \mathbf{domain} = {}^{po}_{s}\mathbf{B}\, \mathbf{type}$$

and similarly:

$${}^{p}_{o}\mathbf{B}\, {}^{po}_{s}\mathbf{B}\, \mathbf{range} = {}^{po}_{s}\mathbf{B}\, \mathbf{type}$$

while for the last inference rule, we obtain:

$${}^{po}_{s}\mathbf{B}\, {}^{po}_{s}\mathbf{B}\, \mathbf{subPropertyOf} = {}^{po}_{s}\mathbf{B},$$

using the same kind of algebra.

Since our design choice leads to a multi-linear relation between the three statement elements, by construction we can also easily define reification, i.e., define the statement **s** itself, via a relation of the form:

$$\mathbf{s} = {}^{spo}\mathbf{B}\, \mathbf{x}\, \mathbf{p}\, \mathbf{y}$$

which is to be explicitized, and then obtain the `rdf:subject`, `rdf:predicate`, or `rdf:object` from the calculated statement, by further algebraic combinations.

4.3 Other intrinsic OWL properties

Beyond our objective of integrating RDFS axioms, it appears that the properties of the chosen algebraic VTB structure allows us to directly implement some of the OWL features. We already mentioned in 4.2 the possibility of inferring any predicate chaining, allowing to implement OWL2 property chains (`owl:ObjectPropertyChain`, i.e., property defined by appending two other properties) which generalizes this mechanism. Further OWL features include:

- $-\mathbf{c}$ the complement of a class $\mathbf{c}$ (`owl:complementOf`)
- $\mathbf{c_1} + \mathbf{c_2}$ the union of two classes $\mathbf{c_1}$ and $\mathbf{c_2}$, this

definition may be extended to any number of classes (`owl:unionOf`)

- the inverse $\mathbf{p}^{\sim}$ of a property $\mathbf{p}$ (`owl:inverseOf`)
- if a property p is encoded as a vector $\mathbf{p}$ such that the matrix $\mathbf{B_p}$ is symmetric, we can directly conclude that this property is symmetric. Indeed, the transpose of a symmetric matrix is equal to itself; since the matrices are considered orthogonal, the transpose is also the approximate inverse, therefore the approximate inverse of a symmetric matrix is equal to itself (`owl:SymmetricProperty`)

Further investigating how other intrinsic OWL properties can be integrated is a perspective of the present work.

5 Introducing data and data structure

5.1 Representing literals

A step further, we not only define relationships between entities (i.e., "ObjectProperty") but also relate an entity to a literal (i.e., "DataProperty"), i.e., a quantitative or qualitative value.

Regarding numerical values, it appears that we can easily define multidimensional numerical values of the form:

$$\mathbf{v} = \mathbf{e}_1^{k_1} + \mathbf{e}_2^{k_2} + \cdots$$

where $\mathbf{e}_i$ is a identifier representing the k_i-th component and $\mathbf{e}_i^{k_i}$ stands for the i-th iterate of the binding operator (Komer et al. 2019). This allows to define integer values, and even more.

If the binding is performed using a convolution operator, it is easily shown that this generalizes to complex numbers.[6]

In our case, we propose an alternative and consider only "physical" numerical values r, i.e., bounded in $[\min, \max]$ and with a finite precision ε, which is practice, the case for any physical value. We must thus write:

$$r \stackrel{\text{def}}{=} k\,\frac{\varepsilon}{\max - \min} + \min, k \in \{0 \cdots \lfloor \frac{\max - \min}{\varepsilon} \rfloor\}$$

considering that two values v_1, v_2 with $|v_1 - v_2| < \varepsilon$ are indistinguishable. Such strong specification is particularly useful for numerical calculus (normalized estimations, spurious value detection, estimation precision threshold, ...) and in our case, it allows to consider only the related positive integer value k.

Regarding string literals or qualitative values (e.g,. Boolean "true" or "false" value), at this stage we simply propose to define one identifier (i.e. a string literal and a random unit vector) for each value.

5.2 Data container

In RFD and in any knowledge specification, we need to define data containers (i.e., `rdfs:Container`), mainly either unordered (i.e., `rdf:Bag` or "set") or ordered (i.e., `rdf:Seq` or "list").

More precisely, RDF/RDFS provides several classes to represent collections:

(i) `rdf:Container`, which includes the following subclasses:

(i.i) `rdf:Seq`, which is an ordered container,

(i.ii) `rdf:Bag`, which is an unordered container,

(i.iii) `rdf:Alt`, which contains a set of "alternatives", the first element of which being the default, and the other elements constituting an unordered collection

(ii) `rdf:List` completed by:

- the properties `rdf:first` and `rdf:rest`
- the instance `rdf:nil` corresponding to the empty list

In our representation, ordered sets (`rdf:List` or `rdf:Seq`) could all be represented as lists in the sense of RDF, in a recursive manner.[7]

Furthermore, unordered sets of value are obviously represented by simple addition, i.e.:

$$\text{bag} = \mathbf{x}_1 + \mathbf{x_2} + \cdots$$

allowing to easily implement membership property, element addition and deletion.

A step ahead, "Alternative" containers (i.e., `rdf:Alt` in the RDFS sense) are simply implemented by a combination of the two previous representations, where `rdf:first` points to an addition. Similarly, we could define vectors like indexed containers like in (Eliasmith 2013).

6 Effective neuronal implementation: an illustrative example

In order to illustrate these developments and validate the fact the proposed formalism is effectively compatible with a biologically inspired approach, we have considered a very small example of ontology (schematized in Fig. 2) inspired by the so-called "pizza" tutorial ontology [8] (Horridge 2011). To that end, we used the Nengo simulator (Bekolay et al. 2014) considering two associative memories as schematized in Fig. 3. [9]

The Nengo platform provides an effective implementation of the Neural Engineering Framework (NEF) and the Semantic Pointer Architecture (SPA). It enables building large-scale bio-inspired models of spiking neurons by connecting together reusable modular components (see (Bekolay et al. 2014) for more details). Therefore, instead of directly manipulating the neurons, we can implement our architecture

[6] In a nutshell, because it can be related to a numerical exponentiation via a Fourier transform (Komer et al. 2019).

[7] This is done by storing the following associations:

$$\begin{aligned}
\mathbf{l} \rightarrow & \mathcal{B}(\texttt{rdf:List}, \texttt{rdf:type}) \\
& + \mathcal{B}(\mathbf{l_1}, \texttt{rdf:first}) \\
& + \mathcal{B}(\mathbf{l}', \texttt{rdf:rest}) \\
\mathbf{l}' \rightarrow & \mathcal{B}(\texttt{rdf:List}, \texttt{rdf:type}) \\
& + \mathcal{B}(\mathbf{l_2}, \texttt{rdf:first}) \\
& + \mathcal{B}(\mathbf{l}'', \texttt{rdf:rest}) \\
\cdots & \\
\mathbf{l^{(n-1)}} \rightarrow & \mathcal{B}(\texttt{rdf:List}, \texttt{rdf:type}) \\
& + \mathcal{B}(\mathbf{l_n}, \texttt{rdf:first}) \\
& + \mathcal{B}(\texttt{rdf:nil}, \texttt{rdf:rest})
\end{aligned}$$

[8] https://github.com/owlcs/pizza-ontology

[9] The code is openly shared at https://gitlab.inria.fr/line/aide-group/onto2spa

using only those modular neural networks and the available connecting operations. Among other features, it encompasses binding and unbinding operations defined within the VTB algebra, as well as associative memories which allow us to store and recall patterns organized into SPA vocabularies. Theoretical details underlying the implementation of such associative memories are available in (Stewart, Tang, and Eliasmith 2011).

To illustrate the use of these features with an example, we created a Nengo vocabulary containing all the resources of our ontology encoded as vectors, and we stored asserted memberships and relationships between these resources into a first associative memory (AM1). In a second associative memory (AM2), we stored the RDFS entailment rule referred to in our paper as the class inheritance (*rdfs9* described in 4.1). Such an architecture could store more rules based on predicate chaining but, for the sake of simplicity, we restricted our example to this one only.

The idea is to query the knowledge base (stored into AM1) against the entailment rules (stored into AM2) according to some specific cues: our network can therefore be seen as a question answering system, where the question to answer is raised by these cues. For instance, in our example, the system receives a subject cue (thisPizza) and a predicate cue (type), which induce the question "what is the type of thisPizza". A third cue indicates which rule to use: in this case, the class inheritance entailment (in practice, instead of a third cue, the network could keep a buffer of several rules to test successively). In order to visualize the different steps of the data processing, we plotted the similarities of the module outputs at several points of the architecture, compared against the symbols in our vocabulary (Fig 4).

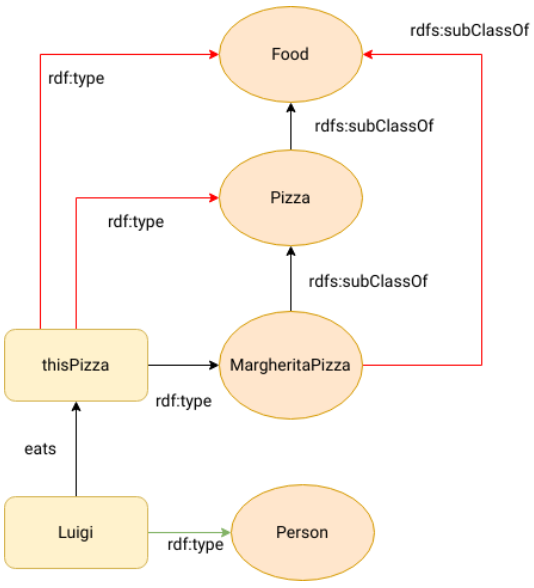

Figure 2: An example of a very simple ontology with two individuals, black arrows correspond to factual statements input in the data base and red arrows to inferred statements. Rectangular boxes stand for individuals, round boxes for classes and arrows are labeled by properties.

7 Discussion and conclusion

What has been presented here is a proof of concept of mapping between RDFS (with some OWL extensions) ontology specification and a biologically plausible numerical implementation based on VSA, with a preliminary partial numerical experimentation. This mainly aims at opening new research perspectives on the unification of symbolic and numerical approaches for cognitive modeling and Artificial Intelligence.

Here the key point is to demonstrate that any symbolic knowledge, as possibly expressed in languages such as RDFS and OWL, can be also represented and manipulated with a numerical neuronal formalism of associative memory. Regarding the genericity of this demonstration, it is worth mentioning that (Mandler 2011) proposes that human (neuronal) memory can represent three classes of structures (in short, associations, sequences and relational structures), the three of them being discussed here. This is especially interesting for our team, since we study human learning using this kind of symbolic approaches (Mercier et al. 2021). Therefore the biological plausibility of ontology representation and reasoning is a key issue. By plausibility, we indeed do not claim that this is directly coded "as is" in the brain, but that what corresponds to symbolic processing in the brain may be represented by such processing, exactly as discussed by (Eliasmith 2013). Beyond that, this kind of representation allows us to inject prior knowledge into learning systems, and opens new perspectives on symbol emergence.

Of course, several alternative implementations of ontologies in VSA are possible. While we chose a non-commutative, non-associative algebra to differentiate the functions (subject, predicate, object) of the resources in a statement, we could also achieve that in a commutative and associative algebra such as HRR, thus using a slightly different representation similar to what has been done in (Crawford, Gingerich, and Eliasmith 2016) to represent the Wordnet database. [10] However, this formalism is quite heavy and we would miss some of the interesting properties induced by binding directly the predicate to the object.

This preliminary study is to be completed at different levels. At a technical level, the tensorial generalization in section 4.2 has been stated at an abstract level, and has yet to be effectively implemented in order to target all RDFS mechanisms. The fact that we rely on Description Logic Programming (DLP) restrains the possibility to take into account the whole decidable part OWL2 specification and we aim to surpass this limit by further studying the intrinsic algebraic properties of our representation. We will also, as in human cognitive processes, consider "approximate" rea-

[10] In that case, the "work-around" consists in adding new vectors $\mathbf{predicate}$ and $\mathbf{object}$ accounting for the functions in the triple, as well as an index t_i (represented by a vector $\mathbf{t_i}$) to each statement and store the triples as:

$$\mathbf{s_1} \rightarrow \tau_1\, \mathbf{t_1} \circledast (\mathbf{p_1} \circledast \mathbf{predicate} + \mathbf{o_1} \circledast \mathbf{object})$$

$$\mathbf{s_1} \rightarrow \tau_2\, \mathbf{t_2} \circledast (\mathbf{p_2} \circledast \mathbf{predicate} + \mathbf{o_2} \circledast \mathbf{object}) + \tau_3\, \mathbf{t_3} \circledast (\mathbf{p_3} \circledast \mathbf{predicate} + \mathbf{o_3} \circledast \mathbf{object})$$

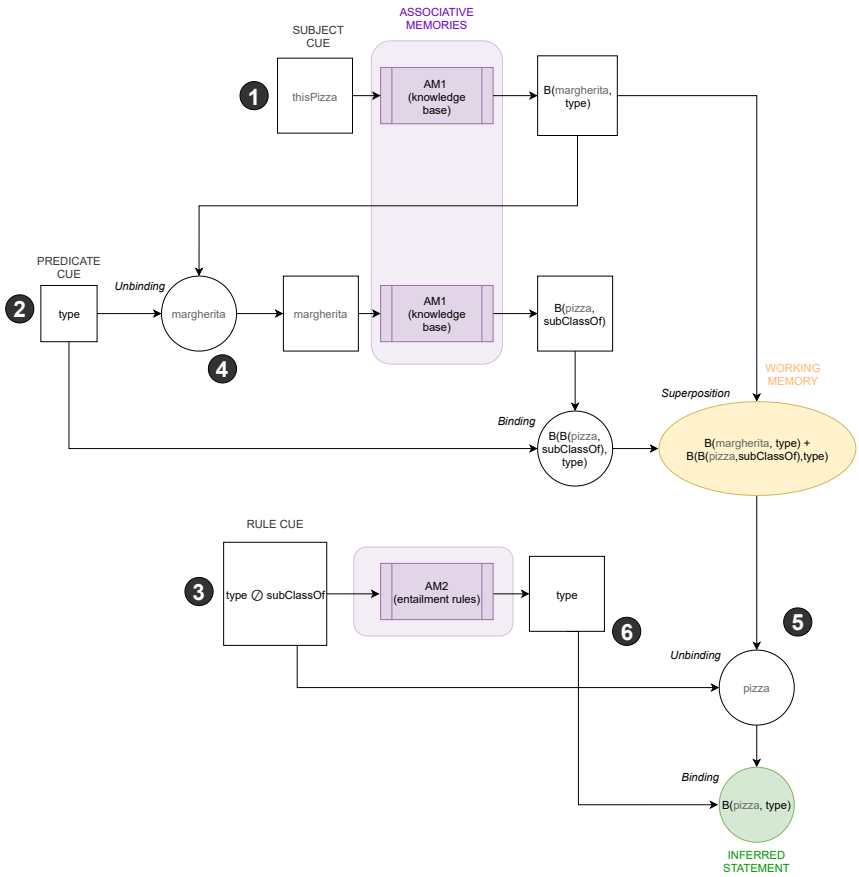

Figure 3: A proposition of architecture to implement our ontology and inference rules in a neuronal system, using the modules provided by the Nengo simulator. A subject cue (thisPizza) and a predicate cue (type) prompt the question "what is the type of thisPizza"; a third cue indicates the rule to use (class inheritance entailment). The network retrieves information in each associative memory and combines them to infer that thisPizza is not only a MargheritaPizza but also, through class inheritance, a Pizza. (Data is in square boxes and their processing in round boxes. Rectangular purple boxes account for associative retrievals. The circled numbers are markers locating which signals are plotted on Fig. 4.)

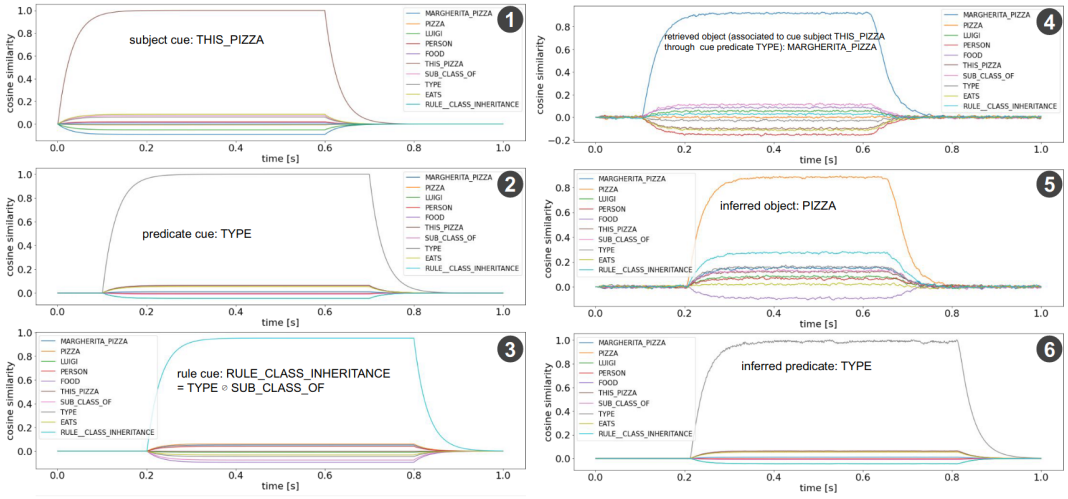

Figure 4: The numerical simulation results. Each signal is visualized through its cosine similarity against the vocabulary. The cues guiding the reasoning (under the form of square inputs) are plotted on the left, and processing steps are on the right (refer to Fig. 3 for the location of each signal, marked by the circled numbers). We can observe the correctness of the inferred object and predicate (respectively PIZZA and TYPE, plotted on 5 and 6) bound together to characterize the subject THIS_PIZZA.

soning about undecidable facts, by better considering the semantic offered by the possibility/necessity score used here. Finally, we made explicit how literal representation is easily possible, but a lot of work is still to be done to have this part fully operational.

We are also aware that, in our representation, the ontology and inference rules are "hard-coded" within the associative memories; the long-term perspective would be to learn those associations. As new observations come along, we wish to store the newly observed relationships between individuals. A step further, logical inference as we proposed is a way to learn new associations, both memberships and relationships, within the ontology itself: for now, the newly inferred statements are only kept into the working memory, but we could store them for future use. Last but not least, we also wish to learn the inference rules themselves, by spotting some patterns that are repeated within the ontology. We presume that this vector formalism will facilitate learning, as it is the representation used by most learning algorithms, although a concrete implementation for this feature is still an open question. In order to remember inferred relationships, the naive approach would be to change the connections in the associative memory or even add new ensembles by hand. However, the neuroscience and machine learning fields provide more sophisticated and biologically plausible solutions to this problem. For example, (Voelker, Crawford, and Eliasmith 2014) used a combination of unsupervised and supervised learning to learn new key-value associations in spiking neurons online.

Acknowledgments. Margarida Romero is highly thanked for powerful ideas regarding the use of this formalism to model human learning which is at the origin of this technical work. We also had a great pleasure discussing with Terrence Stewart whom we deeply thank for his insightful feedback.

References

Alexandre, F. 2019. De quelles façons l'intelligence artificielle se sert-elle des neurosciences ? *The Conversation*.

Allemang, D.; Hendler, J.; and Gandon, F. 2020. *Semantic Web for the Working Ontologist*. ACM.

Ayadi, A.; Samet, A.; de Beuvron, F. d. B.; and Zanni-Merk, C. 2019. Ontology population with deep learning-based NLP: a case study on the Biomolecular Network Ontology. *Procedia Computer Science* 159:572–581.

Bekolay, T.; Bergstra, J.; Hunsberger, E.; DeWolf, T.; Stewart, T. C.; Rasmussen, D.; Choo, X.; Voelker, A. R.; and Eliasmith, C. 2014. Nengo: a Python tool for building large-scale functional brain models. *Front. Neuroinform.* 7.

Chui, C. K., and Mhaskar, H. N. 2018. Deep Nets for Local Manifold Learning. *Front. Appl. Math. Stat.* 4. Publisher: Frontiers.

Cohen, W. W.; Yang, F.; and Mazaitis, K. R. 2017. TensorLog: Deep Learning Meets Probabilistic DBs. *arXiv:1707.05390 [cs]*. arXiv: 1707.05390.

Crawford, E.; Gingerich, M.; and Eliasmith, C. 2016. Biologically Plausible, Human-Scale Knowledge Representation. *Cogn Sci* 40(4):782–821.

Denœux, T.; Dubois, D.; and Prade, H. 2020. Representations of Uncertainty in AI: Probability and Possibility. In Marquis, P.; Papini, O.; and Prade, H., eds., *A Guided Tour of Artificial Intelligence Research: Volume I: Knowledge Representation, Reasoning and Learning*. Cham: Springer International Publishing. 69–117.

Eidoon, Z.; Yazdani, N.; and Oroumchian, F. 2008. Ontology Matching Using Vector Space. In Macdonald, C.; Ounis, I.; Plachouras, V.; Ruthven, I.; and White, R. W., eds., *Advances in Information Retrieval*, Lecture Notes in Computer Science, 472–481. Berlin, Heidelberg: Springer.

Eliasmith, C., and Anderson, C. H. 2002. *Neural Engineering:Computation, Representation, and Dynamics in Neurobiological Systems*. A Bradford Book. The MIT Press. Publisher: The MIT Press.

Eliasmith, C. 2013. *How to Build a Brain: A Neural Architecture for Biological Cognition*. OUP USA. Google-Books-ID: BK0YRJPmuzgC.

Garcez, A. d., and Lamb, L. C. 2020. Neurosymbolic AI: The 3rd Wave. *arXiv:2012.05876 [cs]*. arXiv: 2012.05876.

Gayler, R. 2003. Vector Symbolic Architectures answer Jackendoff's challenges for cognitive neuroscience. In *Frontiers in Artificial Intelligence and Applications*. Journal Abbreviation: ICCS/ASCS International Conference on Cognitive Science Publication Title: ICCS/ASCS International Conference on Cognitive Science.

Gosmann, J., and Eliasmith, C. 2019. Vector-Derived Transformation Binding: An Improved Binding Operation for Deep Symbol-Like Processing in Neural Networks. *Neural Computation* 31(5):849–869. Publisher: MIT Press.

Grosof, B. N.; Horrocks, I.; Volz, R.; and Decker, S. 2003. Description logic programs: combining logic programs with description logic. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, 48–57. New York, NY, USA: Association for Computing Machinery.

Guo, S.; Wang, Q.; Wang, L.; Wang, B.; and Guo, L. 2016. Jointly Embedding Knowledge Graphs and Logical Rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 192–202. Austin, Texas: Association for Computational Linguistics.

Hohenecker, P., and Lukasiewicz, T. 2020. Ontology Reasoning with Deep Neural Networks. *jair* 68. arXiv: 1808.07980.

Horridge, M. 2011. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3.

Jiménez, A.; Elizalde, B.; and Raj, B. 2018. Sound event classification using ontology-based neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, 9.

Komer, B.; Stewart, T. C.; Voelker, A. R.; and Eliasmith, C. 2019. A neural representation of continuous space using fractional binding. In *41st Annual Meeting of the Cognitive*

Science Society, 6. Montreal, Canada: Cognitive Science Society.

Lallement, Y.; Hilario, M.; and Alexandre, F. 1995. Neurosymbolic Integration: Cognitive Grounds and Computational Strategies. In *Proceedings World Conference on the Fundamentals of Artificial Intelligence*, 12.

Levesque, H. J. 1986. Knowledge Representation and Reasoning. *Annu. Rev. Comput. Sci.* 1(1):255–287. Publisher: Annual Reviews.

Levy, S. D., and Gayler, R. 2008. Vector Symbolic Architectures: A New Building Material for Artificial General Intelligence. In *Frontiers in Artificial Intelligence and Applications*, 6.

Mandler, G. 2011. From Association to Organization. *Curr Dir Psychol Sci* 20(4):232–235. Publisher: SAGE Publications Inc.

McClelland, J. L., and Rogers, T. T. 2003. The parallel distributed processing approach to semantic cognition. *Nat Rev Neurosci* 4(4):310–322.

Mercier, C.; Roux, L.; Romero, M.; Alexandre, F.; and Viéville, T. 2021. Formalizing Problem Solving in Computational Thinking : an Ontology approach. In *2021 IEEE International Conference on Development and Learning (ICDL)*, 1–8.

Petrucci, G.; Ghidini, C.; and Rospocher, M. 2016. Ontology Learning in the Deep. In Blomqvist, E.; Ciancarini, P.; Poggi, F.; and Vitali, F., eds., *Knowledge Engineering and Knowledge Management*, Lecture Notes in Computer Science, 480–495. Cham: Springer International Publishing.

Phan, N.; Dou, D.; Wang, H.; Kil, D.; and Piniewski, B. 2017. Ontology-based deep learning for human behavior prediction with explanations in health social networks. *Information Sciences* 384:298–313.

Plate, T. 1995. Holographic reduced representations. *IEEE Trans. Neural Netw.* 6(3):623–641.

Pulvermüller, F. 2013. How neurons make meaning: brain mechanisms for embodied and abstract-symbolic semantics. *Trends in Cognitive Sciences* 17(9):458–470.

Riegel, R.; Gray, A.; Luus, F.; Khan, N.; Makondo, N.; Akhalwaya, I. Y.; Qian, H.; Fagin, R.; Barahona, F.; Sharma, U.; Ikbal, S.; Karanam, H.; Neelam, S.; Likhyani, A.; and Srivastava, S. 2020. Logical Neural Networks. *arXiv:2006.13155 [cs]*. arXiv: 2006.13155.

Romero, M.; David, D.; and Lille, B. 2019. CreaCube, a Playful Activity with Modular Robotics. In Gentile, M.; Allegra, M.; and Söbke, H., eds., *Games and Learning Alliance*, volume 11385. Cham: Springer International Publishing. 397–405. Series Title: Lecture Notes in Computer Science.

Rusawuk, A. L. 2018. Possibility and Necessity: An Introduction to Modality.

Sajjad, H. P.; Docherty, A.; and Tyshetskiy, Y. 2019. Efficient Representation Learning Using Random Walks for Dynamic Graphs. *arXiv:1901.01346 [cs, stat]*. arXiv: 1901.01346.

Sauerwald, T., and Zanetti, L. 2019. Random Walks on Dynamic Graphs: Mixing Times, HittingTimes, and Return Probabilities. *arXiv:1903.01342 [cs, math]*. arXiv: 1903.01342.

Schlegel, K.; Neubert, P.; and Protzel, P. 2020. A comparison of Vector Symbolic Architectures. *arXiv:2001.11797 [cs]*. arXiv: 2001.11797.

Shi, S.; Chen, H.; Ma, W.; Mao, J.; Zhang, M.; and Zhang, Y. 2020. Neural Logic Reasoning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, 1365–1374. New York, NY, USA: Association for Computing Machinery.

Smith, L. 1994. The development of modal understanding: Piaget's possibility and necessity. *New Ideas in Psychology* 12(1):73–87.

Stewart, T. C.; Tang, Y.; and Eliasmith, C. 2011. A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research* 12(2):84–92.

Sun, R., and Alexandre, F. 2013. *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*. Taylor & Francis Group, 3rd edition edition.

Tettamanzi, A.; Zucker, C. F.; and Gandon, F. 2017. Possibilistic testing of OWL axioms against RDF data. *International Journal of Approximate Reasoning*.

Tous, R., and Delgado, J. 2006. A Vector Space Model for Semantic Similarity Calculation and OWL Ontology Alignment. In Bressan, S.; Küng, J.; and Wagner, R., eds., *Database and Expert Systems Applications*, Lecture Notes in Computer Science, 307–316. Berlin, Heidelberg: Springer.

Voelker, A.; Crawford, E.; and Eliasmith, C. 2014. Learning large-scale heteroassociative memories in spiking neurons.

Wang, M.; Qiu, L.; and Wang, X. 2021. A Survey on Knowledge Graph Embeddings for Link Prediction. *Symmetry* 13(3):485. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

Xiao, H.; Huang, M.; and Zhu, X. 2015. From one point to a manifold: Knowledge graph embedding for precise link prediction. *arXiv preprint arXiv:1512.04792*.

Zhu, W.; Qiu, Q.; Huang, J.; Calderbank, R.; Sapiro, G.; and Daubechies, I. 2017. LDMNet: Low Dimensional Manifold Regularized Neural Networks. *arXiv:1711.06246 [cs]*. arXiv: 1711.06246 version: 1.