Finding Answers to Questions: Bridging between Type-based and Computational Neuroscience Approaches

Staffan Larsson[1], Jonathan Ginzburg[2], Robin Cooper[1], Andy Lücking[3]

[1]University of Gothenburg, [2]Université Paris Cité, [3]Goethe-Universität Frankfurt

Abstract

The paper outlines an account of how the brain might process questions and answers in linguistic interaction, focusing on accessing answers in memory and combining questions and answers into propositions. To enable this, we provide an approximation of the lambda calculus implemented in the Semantic Pointer Architecture (SPA), a neural implementation of a Vector Symbolic Architecture. The account builds a bridge between the type-based accounts of propositions in memory (as in the treatments of belief by Ranta, 1994 and Cooper, 2023) and the suggestion for question answering made by Eliasmith (2013), where question answering is described in terms of transformations of structured representations in memory providing an answer. We will take such representations to correspond to beliefs of the agent. On Cooper's analysis, beliefs are considered to be types which have a record structure closely related to the structure which Eliasmith codes in vector representations (Larsson et al., 2023). Thus the act of answering a question can be seen to have a neural base in a vector transformation translatable in Eliasmith's system to activity of spiking neurons and to correspond to using an item in memory (a belief) to provide an answer to the question.

1 Introduction

Understanding how semantic representations are instantiated in biological neural networks remains a fundamental challenge in cognitive science. The Semantic Pointer Architecture (SPA) has been used to build what is currently the world's largest functional brain model (Spaun; Eliasmith, 2013; Eliasmith et al., 2012; Voelker and Eliasmith, 2023), which includes perception, decision making, and motor control systems integrated in a cognitive model that is implemented in spiking neurons and captures detailed anatomical and physiological characteristics of the mammalian brain. The SPA's structured representations are a neural implementation of a Vector Symbolic Architecture (VSA; Gayler, 2003; Schlegel et al., 2022). The basic strategy of the SPA, as we explain below, is to combine a VSA's algebraic structure on a vector space, with coding and decoding operations into ensembles of neurons. In this way, one can retain compositional analyses of natural language in a transparent way that contrasts with LLM approaches, while retaining the robustness and the continuity vectors provide in semantic space. In this paper, we begin to address this challenge by focussing on the processing of questions and answers within a VSA approach, using a VSA approximation of the lambda calculus, and how it can be imlemented in neural network simulations.

Plate (2003, §3.4) and Eliasmith (2013, §4.4) discuss how structured representations encoded as vectors can be manipulated to support reasoning. In particular, on pp. 135ff Eliasmith gives a simple suggestion of how some aspects of question answering could work. In terms of vectors it involves a convolution of a proposition expressed as superpositions of role–filler pairs with another vector corresponding to the question in order to obtain a vector which approximately encodes the answer. However, as Eliasmith (p. 139 2013) himself notes, this model "does not have a solid linguistic justification". Accordingly, VSA approaches to date lack a semantically motivated representation of question and question answering (QA). In particular, some aspects of question and answer processing are still missing, specifically (1) how a proposition containing an answer to a question can be found in memory and (2) how a question and an (elliptical) answer can be combined into a proposition upon hearing the answer. Here we attempt to fill this gap by bridging between a type-based semantic theory of questions and the computational neuroscience VSA approach of the SPA.

The memory which is being probed for answers

can be thought of as a collection of proposition encodings – in some cases long term memory (e.g., (1a)), in others working memory, at times a combination thereof (e.g., (1b)).

(1) a. What is the capital of Togo?

b. Are you aware of the wasp on your nose?

Of course, we may not have precisely the proposition we need in memory but may need to reason from a proposition we have to the proposition we need to answer the question.

In this paper we will first explain the formal and conceptual backgrounds that are synthesized in this paper (section 2), including SPA and Type Theory with Records, the semantic framework whose entities underpin our discussion here. We will then explain the previous work on question answering in terms of flat role–filler structures (section 3). Finally, we propose a more comprehensive account of questions and answers using our type-based approach, where we use a SPA approximation of the lambda calculus to handle both question answering and semantic ellipsis resolution (section 4). Section 5 presents a toy model implementation that illustrates and evaluates some features of the SPA-TTR hybrid approach to QA. We conclude in section 6.

2 Background

In this section we describe vector symbolic architectures that mediate between symbolic and distributive representations, neural networks that implement such representations, and finally type-theoretic semantics, specifically the framework Type Theory with Records. In the following, we briefly discuss each of these backgrounds.

2.1 Holographic Reduced Representations (HRR)

Holographic Reduced Representations (HRR; Plate, 2003) are a particular implementation of compressed representations, that is, (higher-order) semantic representations that are obtained by "compressing" (lower-level) semantic representations (Hinton, 1990). HRR achieve this by *circular convolution*: a multiplication operation that binds high-dimensional vectors of dimension d into a new vector of dimension d. Thus, HRR is a true-to-dimension instance of a *Vector Symbolic Architecture* (VSA; Gayler, 2003), in contrast to, for instance, tensor products (Smolensky, 1990).

The vector algebra of HRR includes the following operators ($\mathbf{a}$, $\mathbf{b}$, ... are vectors, i.e., lists of numbers of length d, the dimension of the vector; in the following we usually assume normalized unit vectors, i.e. vectors whose length is 1)[1]:

- $+$: $\mathbf{a}+\mathbf{b} = [\mathbf{a}_0+\mathbf{b}_0, \mathbf{a}_1+\mathbf{b}_1, \ldots, \mathbf{a}_{d-1}+\mathbf{b}_{d-1}]$
- $-$: $\mathbf{a}-\mathbf{b} = [\mathbf{a}_0-\mathbf{b}_0, \mathbf{a}_1-\mathbf{b}_1, \ldots, \mathbf{a}_{d-1}-\mathbf{b}_{d-1}]$
- $\circledast$: $\mathbf{c} = \mathbf{a} \circledast \mathbf{b} : \mathbf{c}_j = \sum_{k=0}^{d-1} \mathbf{a}_k \mathbf{b}_{j-k \pmod d}$
- inverse: $\mathbf{a}' = [\mathbf{a}_0, \mathbf{a}_{d-1}, \mathbf{a}_{d-2}, \mathbf{a}_{d-3}, \ldots, \mathbf{a}_1]$

Basic properties of HRR vector manipulations (Plate, 2003): Circular convolution can be regarded as a multiplication operator for vectors. It has many properties in common with both scalar and matrix multiplication. It is commutative, associative, and bilinear. There is an identity vector and a zero vector and each vector has an approximate inverse ('involution'). Involution distributes over addition and convolution, and is its own inverse. It is noteworthy that the true-to-dimension HRR vector manipualtions are lossy: in particular the inverse $\mathbf{a}'$ of a vector $\mathbf{a}$ is not the exact inverse but approximates it. This "lossiness" introduces the need for clean-up memories when using it in cognitive VSA architectures like the SPA (see below).

2.2 Neural Engineering Framework (NEF)

The *Neural Engineering Framework* (NEF; Eliasmith and Anderson, 2003) implements vectorial representations and manipulations in neural simulations.[2] The basic idea is that high-dimensional vectors figure as the currents that are processed (encoded and decoded) by ensembles of neurons in real time.

2.3 Semantic Pointer Architecture (SPA)

Semantic pointers (Eliasmith, 2013) are structured representations (they can be "dereferenced" to access the more extensive information folded into them) in a (high-dimensional) vector space that function as symbols in cognitive processing and are processed as activity patterns in neural networks. In implementational terms, a semantic pointer can be conceived as a "vector with a name" (it can be addressed); the collection of semantic pointers in this sense make up a "dictionary". In the *Semantic Pointer Architecture* (SPA; Eliasmith, 2013)

[1] The Euclidian length, $\|\cdot\|$, of a vector is the square root of the sum of the squares of its dimension: $\|\mathbf{a}\| = \sqrt{\mathbf{a}_0^2 + \ldots + \mathbf{a}_{d-1}^2}$.

[2] See https://www.nengo.ai/.

of cognitive functions, questions and answers are modeled as semantic pointers (see section 3). SPA uses HRR as default operation for binding (denoted by "$\circledast$") and unbinding (i.e., binding with the inverse vector, below notated with a prime) vectors (see section 2.1), though other algebras can be used in the SPA as well.

Since HRR is lossy (see above), processing with circular convolution "degrade[s] gracefully in the presence of noise" (Plate, 2003, p. 141). To distinguish random noise from "allowable representations", the high-dimensional vectors that are obtained from vector manipulations (carried out by ensembles of neurons in NEF, see section 2.2) are compared to "valid representations" from the vocabularies of semantic pointers (Eliasmith, 2013, §4.6). This validation is derived from inclusion in clean-up memory: the noisy processed vector is validated against the semantic pointers (now conceived as vectors) in the vocabulary of semantic pointers. Technically this is spelled out as the dot product (the standard measure of vector similarity) of the processed vector and the named vectors in the semantic pointer vocabulary.

Clean-up memories are a natural component of lossy VSAs. While the need for long-term clean-up memories is uncontroversial, it is only used sparingly in biological systems because its maintenance is neurologically costly (Stewart et al., 2011). A clean-up memory call replaces a noisy processing vector with its most similar semantic pointer vector from the semantic pointer vocabulary. We indicate the need (or at least the benefit) and the use of clean-up memory as $\mathbf{Clean}(\cdot)$ (or $\mathbf{Clean}_D(\cdot)$ where D indicates the domain of the cleanup function, i.e. the vocabulary which the cleanup function compares with) in the following.

Although being noisy, HRRs involve, among other things, an associative and commutative vector combinatory operation, which is not necessarily the case with other algebraic systems (e.g., Vector-Derived Transformation Binding (Gosmann and Eliasmith, 2019), which can nevertheless be neurally efficient). Futhermore, the SPA (via NEF) relates symbolic, distributional and neural levels. For this reasons, we formulate our approach in terms of the SPA and its default HRR algebra.

2.4 Type Theory with Records (TTR)

We give a brief sketch of those aspects of TTR which we will use in this paper. For more detailed accounts see Cooper (2023).

$s : T$ represents a judgement that s is of type T. Types may be either *basic* or *complex* (in the sense that they are structured objects which have types or other objects introduced in the theory as components). One basic type that we will use is *Ind*, the type of individuals; another is *Real*, the type of real numbers.

Among the complex types are *ptypes* which are constructed from a predicate and arguments of appropriate types as specified for the predicate. Examples are 'man(a)', 'see(a,b)' where $a, b : Ind$. The objects or *witnesses* of ptypes can be thought of as situations, states or events in the world which instantiate the type. Thus $s : \mathrm{man}(a)$ can be glossed as "s is a situation which shows (or proves) that a is a man".

Another kind of complex type are *record types*. In TTR *records* are modelled as a labelled set consisting of a finite set of fields. Each field is an ordered pair, $\langle \ell, o \rangle$, where ℓ is a *label* (drawn from a countably infinite stock of labels) and o is an object which is a witness of some type. No two fields of a record can contain the same label. Importantly, o can itself be a record.

A *record type* is like a record except that the fields are of the form $\langle \ell, T \rangle$ where ℓ is a label as before and T is a type. The basic intuition is that a record, r is a witness for a record type, T, just in case for each field, $\langle \ell_i, T_i \rangle$, in T there is a field, $\langle \ell_i, o_i \rangle$, in r where $o_i : T_i$. (Note that this allows for the record to have additional fields with labels not included in the fields of the record type.) The types within fields in record types may *depend* on objects which can be found in the record which is being tested as a witness for the record type. We use a graphical display to represent both records and record types where each line represents a field. Example (2) represents the type of records which can be used to model situations where a man runs.

$$(2) \quad \begin{bmatrix} \mathrm{ref} & : & Ind \\ \mathrm{c_{man}} & : & \mathrm{man(ref)} \\ \mathrm{c_{run}} & : & \mathrm{run(ref)} \end{bmatrix}$$

A record of this type would be of the form

$$(3) \quad \begin{bmatrix} \mathrm{ref} & = & a \\ \mathrm{c_{man}} & = & s \\ \mathrm{c_{run}} & = & e \\ \ldots & & \end{bmatrix}$$

where $a : Ind$, $s : \mathrm{man}(a)$ and $e : \mathrm{run}(a)$.

Detailed accounts of questions in TTR can be found in (Ginzburg, 2012; Ginzburg et al., 2014, 2022). These are based on viewing questions as

akin to propositional abstracts. This approach is the basis for the most detailed discussion of the response space of questions (Ginzburg et al., 2022) that we are aware of. Another reason for using TTR is that all complex TTR objects are constructed from labelled sets, which correspond to the representation of structured objects which Eliasmith achieves using superposition and circular convolution (Larsson et al., 2023).

2.5 Mapping TTR onto SPA

First steps towards a hybrid of formal and neural semantics by mapping TTR to the SPA have been taken by Larsson et al. (2023). The basic idea was to relate type judgments (not just types) to neural events. By this means, basic types, perceptual and cache-based judgements, singleton types, record types, meet types and merging of record types, ptypes, and subtyping have been accounted for. However, this previous work had little to say about functions, which we address in the following by example of *Wh*-questions.

3 Previous work on question answering in HRR and SPA

Modelling questions in the NEF-SPA means constructing a (biocognitive inspired) network that models a question-related task. Eliasmith (2013) illustrates such a network with question answering (QA). The idea is that the visual cortex provides statements and questions, both supplied as semantic pointers. Statement pointers (e.g., "**red** $\circledast$ **circle**") represent "the world" and are sent to a memory population of neurons (working memory). Question pointers pose questions to memory content (e.g., "$\mathbf{red}'$" $\approx$ "What is red?"). Basal ganglia monitors input and determines what kind of routing is appropriate to answer the question. The answer is sent to the clean-up memory and the memory item with highest similarity is sent to motor cortex (i.e., the answer is given). For instance, the simple statement "dog_{52} chases cat_{43}" could be represented as a flat role–filler structure as follows:

(4) $\mathbf{agent} \circledast \mathbf{dog_{52}} + \mathbf{frame} \circledast \mathbf{chase} + \mathbf{theme} \circledast \mathbf{cat_{43}}$

A fixed set of semantic role labels provides information about where to find the desired information in semantic pointers and can be used for modeling questions. For example, *Who*-questions address entities (persons, or animate beings in general) associated with a certain role in role–filler representations, where addressing is captured in terms of unbinding.

Accordingly, asking the given statement (4) "Who does dog_{52} chase?" amounts to unbinding (4) with $\mathbf{theme}'$ and thereby retrieving an answer:

(5) (4) $\circledast$ $\mathbf{theme}' \approx \mathbf{cat_{43}}$

There are obvious ways to make this QA network more complex. Firstly, more complex models need to employ more semantic roles (e.g., location, time, colour, shape, manner, ...), in particular to deal with embedded clauses.

Secondly, a long-term memory along with the working memory will be used. Routing through basal ganglia will then decide when to look into working or long-term memory to find an answer to a given question.

QA with flat role–filler structures in the SPA according to the pattern outlined above offers two insights:

1. The structures of questions and answers have to match closely: the statement pointer that is enquired by a question pointer needs to be tailored to the question asked. In this sense, the answering statement has to be given/known in advance.
2. QA is *dynamic*: regardless of the knowledge source of the answer (e.g., actual perception or long term memory retrieval), both the question and the answer are rehearsed in working memory for QA.

The first issue seems to be a limiting consequence of using the inverses of "label pointers" within flat role–filler structures as models for questions. This assumption is not shared in linguistic semantics, which instead uses *functions*. Linguistic semantics in turn ignores the dynamicity of knowledge source retrieval and QA rehearsal in working memory. Here we aim to reconcile this gap between neuro-computational and formal semantics.

4 Dealing with questions using Lambda calculus in SPA-TTR

The role-filler approach to simple question answering outlined in section 3 assumes that you have already found the proposition which provides the answer and tells you where to look to find the answer within the proposition. A natural language question has to in addition give you material to find the proposition in memory. Also, since answers

are often semantically underspecified (and syntactically elliptical) and thus need to be understood in the context of a specific question, it must be possible to combine questions and (underspecified) answers into full propositions.

An account of question and answer processing thus needs to address the following:

- How are questions and answers combined to propositional types (ptypes)?
- How can answers to a question be found in memory?

We address these questions in sections 4.1 and 4.5, respectively. Along the way, we also cover typechecking of answer candidates in section 4.2, double abstraction ("Who chases who?") in section 4.3, and as a preparation for section 4.5 we also adapt the role-filler approach to extracting an answer from a question and a ptype, in section 4.4.

We will represent questions as functions. The body of the function tells you what would be an appropriate proposition to find in memory. The abstraction in the function tells you where to look in that proposition.

4.1 Combining question and answer into a ptype

We will start by showing how a question and a semantically underspecified (elliptical) answer can be combined into a TTR ptype. For simplicity, we will assume that referents have been identified, so that we write dog_{52} (a specific dog) instead of "the dog" or "that dog". Suppose we have the following exchange:

(6) Q: "Who does dog_{52} chase?" A:"cat_{43}"

In TTR, this would be handled by applying a question q to an answer a to arrive at a ptype p:

(7) a. $q = \lambda v : \mathit{Ind}.\text{chase}(\text{dog}_{52}, v)$
b. $a = \text{cat}_{43}$
c. $p = q(a) = \text{chase}(\text{dog}_{52}, \text{cat}_{43})$

To convert this into SPA-TTR, we use the fact that in TTR, $\lambda v : \mathit{Ind}.\text{chase}(\text{dog}_{52}, v)$ is the labelled set

(8) $$\left\{ \begin{array}{l} \langle \text{lambda}, \mathit{Ind} \rangle, \\ \langle \text{body} \quad , \left\{ \begin{array}{l} \langle \text{pred}, \text{chase} \rangle \\ \langle \text{arg1}, \text{dog}_{52} \rangle \\ \langle \text{arg2}, \text{'body.arg2'} \rangle \end{array} \right\} \rangle \end{array} \right\}$$

In SPA-TTR, we modify this slightly. Firstly, we let the value of the abstracted field be **I**, the identity vector. Secondly, we add a field **lambdapath** containing a path in the question body leading to the abstracted field.

(9) $$\mathbf{Q} = \begin{pmatrix} \textbf{lambdapath} \circledast \textbf{arg2}+ \\ \textbf{lambdatype} \circledast \textbf{Ind}+ \\ \textbf{body} \circledast \begin{pmatrix} \textbf{pred} \circledast \textbf{chase}+ \\ \textbf{arg1} \circledast \textbf{dog}_{52}+ \\ \textbf{arg2} \circledast \mathbf{I} \end{pmatrix} \end{pmatrix}$$

with **I** the identity vector $\mathbf{I} = [1, 0, 0, \ldots]$, such that for any vector $\mathbf{x}$, $\mathbf{I} \circledast \mathbf{x} = \mathbf{x}$. This is a variant of de Bruijn indexing (de Bruijn, 1972), coding lambda terms without using variables but using paths to mark positions instead. (For a different variant using paths see Cooper, 2023.)

For an answer like the one in (10), we want to get the ptype in (11).

(10) $\mathbf{A} = \textbf{cat}_{43}$

(11) $$\mathbf{P} = \begin{pmatrix} \textbf{pred} \circledast \textbf{chase}+ \\ \textbf{arg1} \circledast \textbf{dog}_{52}+ \\ \textbf{arg2} \circledast \textbf{cat}_{43} \end{pmatrix}$$

In SPA-TTR, **Q** and **A** are inputs to a network for lambda function application that combines them into a proposition by realizing the SPA function below:[3]

(12) $f(Q, A) = Q \circledast \textbf{body}' - \mathit{Path} + \mathit{Path} \circledast A$
where $\mathit{Path} = Q \circledast \textbf{lambdapath}'$

For our example above, this gets us

(13) $\mathbf{P} \approx f(\mathbf{Q}, \mathbf{A}) = \mathbf{Q} \circledast \textbf{body}' - \textbf{arg2} + \textbf{arg2} \circledast \mathbf{A}$

This function outputs the body of the question with the lambda abstracted variable replaced by the argument.

For the specific $Q = \mathbf{Q}$ and $A = \mathbf{A}$ given above, $\mathit{Path} = Q \circledast \textbf{lambdapath}'$ evaluates (with some noise) to **arg2**, yielding

(14) $$\mathbf{P} = \begin{pmatrix} \textbf{pred} \circledast \textbf{chase}+ \\ \textbf{arg1} \circledast \textbf{dog}_{52}+ \\ \textbf{arg2} \circledast \mathbf{I} \end{pmatrix} - \textbf{arg2} + \textbf{arg2} \circledast \textbf{cat}_{43} \approx \begin{pmatrix} \textbf{pred} \circledast \textbf{chase}+ \\ \textbf{arg1} \circledast \textbf{dog}_{52}+ \\ \textbf{arg2} \circledast \textbf{cat}_{43} \end{pmatrix}$$

as desired. To reduce noise, we can add a **Clean**() operation over the domain of possible paths:

(15) $\mathit{Path} = \textbf{Clean}_{\textbf{Paths}}(Q \circledast \textbf{lambdapath}')$
where $\mathit{Paths} = \{\textbf{arg1}, \textbf{arg2}, \textbf{pred}\}$

[3] We would like to thank Chris Eliasmith for help with this formulation.

4.2 Typechecking

Above, we have been ignoring the typechecking specified in (7) . One strategy for including it is to prefix the function f in (12) with a term that returns the identity vector $\mathbf{I}$ if typechecking is successful and noise otherwise. This would mean that f returns noise if typechecking fails, but if it succeeds the result will be identical to using the definition in (12). To achieve this, we can use the fact that the result of binding a vector to its own (approximate) inverse is similar to the identity vector:

(16) $A' \circledast A \approx \mathbf{I}$

The idea is then to use this to compare the type of the answer to the type specified by $\mathbf{Q} \circledast \mathbf{lambdatype}'$. To get the type of the answer, we assume there is a vector $\mathbf{FT}$ which binds objects to their types, so that that $\mathbf{FT} \circledast A'$ for some object A returns the type of A (assuming for now that each object is of exactly one type in $\mathbf{FT}$):

(17) $\mathbf{FT} = \ldots + \mathbf{cat_{45}} \circledast \mathbf{Ind} + \ldots$

The typechecking needed for applying a lambda function Q to an argument A can now be handled by binding with

(18) $(\mathbf{FT} \circledast A') \circledast (Q \circledast \mathbf{lambdatype}')'$

which for (12) gets us

(19) $f(Q,A) =$
$((\mathbf{FT} \circledast A') \circledast (Q \circledast \mathbf{lambdatype}')') \circledast$
$(Q \circledast \mathbf{body}' - Path + Path \circledast A)$

For $\mathbf{Q}$ and $\mathbf{A}$ as in (9) and (10), respectivly, we have that $\mathbf{FT} \circledast \mathbf{cat}'_{45} \approx \mathbf{Ind}$ and $\mathbf{Q} \circledast \mathbf{lambdatype}' \approx \mathbf{Ind}$, so that (18) evaluates to

(20) $\mathbf{Ind} \circledast \mathbf{Ind}' \approx \mathbf{I}$

Since operations such as (18) and (19) introduce a lot of noise, it might be necessary to support them with clean-up steps. For instance, in a simple implementation (see section 5) the similarity of (20) drops to 0.39. If the two unbinding sub-steps involved in (18) are cleaned-up in a memory of basic types first, it reaches 1. In both cases, however, the type *Ind* is the most similar base type for the unbound types of question and answer. If we assume that abstracted arguments are of basic types[4] and take **BType** to be the SPA-TTR implementation of TTR basic types, we thus replace (18) with (21).

(21) $\mathbf{Clean}_{\mathbf{BType}}(\mathbf{FT} \circledast A') \circledast$
$\mathbf{Clean}_{\mathbf{BType}}(Q \circledast \mathbf{lambdatype}')'$

This provides a general method for doing typechecking for SPA-TTR lambda functions. For brevity, we will exclude typechecking below.

4.3 Double abstraction

What about questions with double abstraction, such as "Who chased who?"? In TTR, this is done as a double lambda term $\lambda v_1, v_2 : \langle Ind, Ind \rangle.\text{chase}(v_1, v_2)$. We can construct an f_{sim2} (along the lines of the definition of f above) to work directly on that:

(22) $$\mathbf{Q2} = \begin{pmatrix} \mathbf{lambdapath1} \circledast \mathbf{arg1} + \\ \mathbf{lambdatype1} \circledast \mathbf{Ind} + \\ \mathbf{lambdapath2} \circledast \mathbf{arg2} + \\ \mathbf{lambdatype2} \circledast \mathbf{Ind} + \\ \mathbf{body} \circledast \begin{pmatrix} \mathbf{pred} \circledast \mathbf{chase} + \\ \mathbf{arg1} \circledast \mathbf{I} + \\ \mathbf{arg2} \circledast \mathbf{I} \end{pmatrix} \end{pmatrix}$$

(23) $f_{sim2}(Q, A_1, A_2) = Q \circledast \mathbf{body}' - Path_1 + Path_1 \circledast A_1 - Path_2 + Path_2 \circledast A_2$
where $Path_1 = Q \circledast \mathbf{lambdapath1}'$,
$Path_2 = Q \circledast \mathbf{lambdapath2}'$

As a side note, it is also possible to abstract over the same variable more than once, as in "Who chases herself?", in TTR $\lambda v : Ind.\text{chase}(v, v)$, as shown in (24).

(24) $$\mathbf{Q3} = \begin{pmatrix} \mathbf{lambdapath} \circledast \mathbf{(arg1+arg2)} + \\ \mathbf{body} \circledast \begin{pmatrix} \mathbf{pred} \circledast \mathbf{chase} + \\ \mathbf{arg1} \circledast \mathbf{1} + \\ \mathbf{arg2} \circledast \mathbf{1} \end{pmatrix} \end{pmatrix}$$

Using our original function f in (12) to apply this question to a single argument yields an instantiated ptype as desired:

(25) $f(\mathbf{Q3}, \mathbf{cat_{45}}) =$

$$\begin{pmatrix} \mathbf{pred} \circledast \mathbf{chase} + \\ \mathbf{arg1} \circledast \mathbf{1} + \\ \mathbf{arg2} \circledast \mathbf{1} \end{pmatrix} - (\mathbf{arg1} + \mathbf{arg2}) + (\mathbf{arg1} + \mathbf{arg2}) \circledast \mathbf{cat_{45}} =$$

$$\mathbf{pred} \circledast \mathbf{chase} + \mathbf{arg1} \circledast \mathbf{cat_{45}} + \mathbf{arg2} \circledast \mathbf{cat_{45}} =$$

$$\begin{pmatrix} \mathbf{pred} \circledast \mathbf{chase} + \\ \mathbf{arg1} \circledast \mathbf{cat}_{45} + \\ \mathbf{arg2} \circledast \mathbf{cat}_{45} \end{pmatrix}$$

In a general account of functions and function application, one would also like to include recursive

[4] This assumption is not generally true, e.g. for "why"-questions. We leave such cases for future work.

function application. In SPA-TTR, a recursively applicable function would correspond to a network that can be applied twice, once for each argument. We leave the specification of recursive function application for future work.

4.4 Extracting answer from question and ptype

If we have a ptype P that we know contains the answer to a question Q, we can use a slightly modified version of Eliasmith's method outlined in section 3.

First, we note that the representation in example (4) corresponds closely to our current representation of ptypes, except for the names of the labels (**pred** instead of **frame**, **arg1** instead of **agent**, **arg2** instead of **theme**).

However, representing the whole question as **theme** does not help us in combining questions and answers into ptypes, so instead we use the more elaborate question seen above. Here, **lambdapath** in Q is bound to **arg1**, and by unbinding it we end up doing the exact same operation as suggested by Eliasmith.

(26) $f_{qp}(Q,P) = P \circledast (Q \circledast \mathbf{lambdapath}')'$

Using the **Q** and **A** from (9) and (10), this gets us

(27) $\mathbf{A} = f_{qp}(\mathbf{Q},\mathbf{P}) = \mathbf{P} \circledast (\mathbf{Q} \circledast \mathbf{lambdapath}')' = \mathbf{P} \circledast (\mathbf{arg2})' = \mathbf{cat}_{43}$

4.5 Extracting the answer from the question and memory

In the general case of question answering, however, we cannot assume we have found the ptype. The real challenge is then to find the answer to a question in LTM, and if you only represent the question as $\mathbf{theme}'$, this will not be possible. It needs to also include **chase** and **dog**. Since we also have these in our representation of the question, we can find relevant ptypes in LTM.

We should instead only assume there is a record with many different ptypes, one (or several) of which may contain an answer. Previous work (Cooper et al., 2015) developed a notion of a *judgement history* consisting of a set of Austinian propositions encoding judgements that situations s are of types T, $s : T$. Inspired by this but simplifying matters somewhat, we will here use **M** as a name for a labelled set of ptypes indexed by natural numbers:

(28) $\mathbf{M} = \mathbf{1} \circledast P1 + \mathbf{2} \circledast P2 + \ldots + n \circledast Pn$

where Pi is a SPA ptype. Given this, we can outline a procedure for finding the answer A in **M** to a question Q:

1. $B = Q \circledast \mathbf{body}'$
2. Find a P which is similar to B; since $A+B$ is similar to A and to B and to any bundle $A+C$, then if for (the SPA representation of) some natural number n we have $\mathbf{M} \circledast n' \approx B$ we can conclude that $P \approx \mathbf{M} \circledast n'$ is a subtype of B
3. Let A be the value of $Q \circledast \mathbf{lambdapath}'$ in P, i.e. $A = P \circledast (Q \circledast \mathbf{lambdapath}')'$ (unless it's noise)

As an example, assume $Q = \mathbf{Q}$ as in (10) above and $\mathbf{434} \circledast \mathbf{P}$ is in **M**. Then

(29) a. $B = \begin{pmatrix} \mathbf{pred} \circledast \mathbf{chase} + \\ \mathbf{arg1} \circledast \mathbf{dog}_{52} + \\ \mathbf{arg2} \circledast \mathbf{I} \end{pmatrix}$

b. $F = \mathbf{Q} \circledast \mathbf{lambdapath}' \approx \mathbf{arg2}$

Now, since there is an $n = \mathbf{423}$ for which $\mathbf{M} \circledast n' \approx B$, we conclude that

(30) $\mathbf{P} \approx \mathbf{M} \circledast \mathbf{423}'$
$\mathbf{A} = \mathbf{P} \circledast \mathbf{arg2}' \approx \mathbf{cat}$

However, the above is not quite sufficient since it does not specify a SPA mechanism for searching **M**. To address this, we can use the fact that SPA does not distinguish TTR labels from values, so to find the n we are looking for, we can use B as an approximation of the ptype P we are looking for;

(31) $n = \mathbf{M} \circledast B'$

or more robustly

(32) $n = \mathbf{Clean}_{\mathbf{Nat}}(\mathbf{M} \circledast B')$

so that we can define a lookup function that returns the ptype in **M** which is the most similar to some other ptype L:

(33) $SearchM(L) = \mathbf{M} \circledast (\mathbf{Clean}_{\mathbf{Nat}}(\mathbf{M} \circledast L'))'$

so that we can get an answer A:

(34) $\mathbf{A} = SearchM(B) \circledast \mathbf{arg2}'$

Based on this, we can define a function $f_{qm}(Q)$ returning an answer to a question Q from **M**:

(35) $f_{qm}(Q) \approx SearchM(Q \circledast \mathbf{body}') \circledast (Q \circledast \mathbf{lambdapath}')'$

This solution is sensitive to noise, and applying cleanup over a limited domain will help. Fortunately, our question representation provides a type constraint on the possible answers (the **lambdatype** field) that we can use to specify the cleanup domain:

(36) $f_{qm}(Q) \approx \mathbf{Clean}_{Q \circledast \mathbf{lambdatype}'}(SearchM(Q \circledast \mathbf{body}') \circledast (Q \circledast \mathbf{lambdapath}')')$

Exploring whether this retrieval mechanism works similarly to human associative memory is a topic for future research.

5 A simple proof-of-concept model

To see if λ-abstracted question answering is feasible with neurons, we implemented the key steps from Section 4.5 in Nengo (https://www.nengo.ai/).[5] The "knowledge base" **M** from which an answer is to be found consists of six role-filler semantic pointers (vectors of 128 dimensions) that correspond to the statements P1 = *dog chases cat*, P2 = *dog chases cow*, P3 = *dog sees mouse*, P4 = *mouse sees cow*, P5 = *cat likes mouse*, P6 = *dog likes cow*. The inputs to the network are three questions, posed one after the other: 1. *Who does the cat like*? 2. *Who does the dog chase*? 3. *Who likes the cow*? An answer for 1. can be found in P5 (i.e., *mouse*), an answer for 3. can be found in P6 (i.e., *dog*). 2., however, is ambiguous: possible answers are provided by P1 and P2 (i.e., *cat* or *cow*).

The networks unbinds the body and the lambdapath from the question, substracts the lambdapath, and involutes both to retrieve the fragment answer (i.e., compares the resulting vector to the semantic pointers in clean-up memory). The result is shown in the bottom row in fig. 1 ("Answer without Memory Clean-up") and shows that the model does not perform well: the semantic pointer corresponding to *mouse* is always returned as the answer, which is wrong in all but one case. Apparently, the convolution operations introduce too much noise.

To compensate for the noise, we introduced a memory clean-up step over the ptypes P1-P6 after unbinding the questions' bodies. The vector that is fed into an autoassociative clean-up memory over time is most similar to the propositions shown in the top row of fig. 1 ("Memory Input"). The clean-up step reinforces this input (see "Memory Output"). As a consequence, the network now returns vectors that are indeed most similar to the expected items (i.e., *mouse*, *cow* or *cat*, and *dog*, see "Answer with Memory Clean-up"). Note that the last answer is in close competition with the wrong fragment *mouse*, so a final cleanup may be required.

[5]The model can be obtained from https://github.com/aluecking/QA-SPA-TTR.

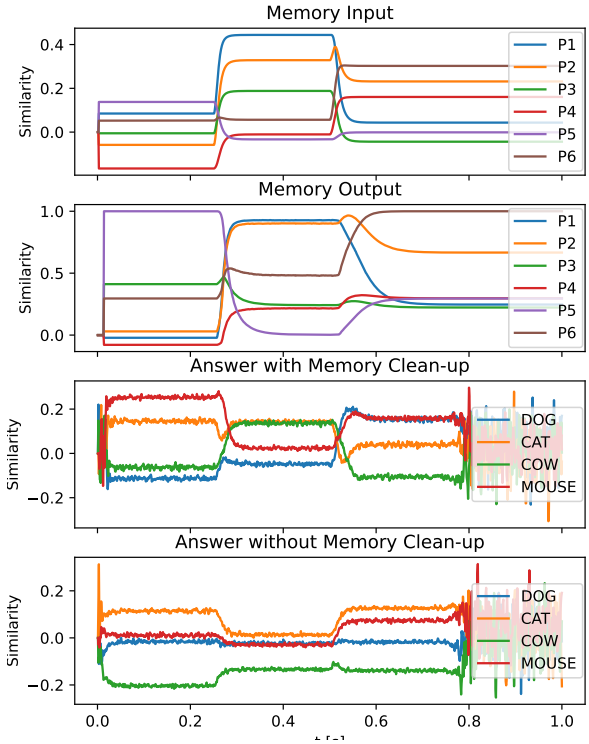

Figure 1: Retrieving a fragment answer according to section 4.5 from a neural simulation. The neural simulation is dynamic because it runs in real time. The elapsed time in seconds is shown on the x-axis. The input to the network changes over time: from 0 s to 0.25 s the input is a semantic pointer that corresponds to the question *Who does the cat like?*, from 0.25 s to 0.5 s to the question *Who does the dog chase?*, and from 0.5 s to 0.72 s to *Who likes the cow?* (no input in the remaining quarter of a second).

6 Conclusions and Future Work

In this paper we have sketched an approach to abstraction, questions, and answering in SPA which allows us to maintain compositional semantic analyses within a biologically plausible cognitive framework. This is of course just the first step for such an account which one should enhance with networks assessing whether a proposition *resolves* a question or constitutes a partial or indirect answer.

Work in VSAs has not to date addressed quantification, but we take this as a first step to showing that this can be done. We hypothesize that the non-Generalized Quantifier TTR-based approach to quantification developed in (Lücking and Ginzburg, 2022) affords a feasible path to this aim. Another important step involves providing an account of working memory, in order to integrate the current insights of dynamic dialogical semantic frameworks such as KoS (Ginzburg, Eliasmith, and Lücking, 2024), MSDRT (Kamp, 2024), and SDRT (Asher and Lascarides, 2003).

References

Nicholas Asher and Alex Lascarides. 2003. *Logics of Conversation*. Cambridge University Press, Cambridge.

Robin Cooper. 2023. *From Perception to Communication: a Theory of Types for Action and Meaning*. Oxford University Press. Open access: https://global.oup.com/academic/product/from-perception-to-communication-9780192871312.

Robin Cooper, Simon Dobnik, Staffan Larsson, and Shalom Lappin. 2015. Probabilistic type theory and natural language semantics. *Linguistic Issues in Language Technology*, 10.

Nicolaas Govert de Bruijn. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392.

Chris Eliasmith. 2013. *How to build a brain: A neural architecture for biological cognition*. Oxford University Press.

Chris Eliasmith and Charles H. Anderson. 2003. *Neural Engineering*. Computational Neuroscience. MIT Press, Cambridge, MA.

Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. 2012. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205.

Ross W. Gayler. 2003. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. pages 133–138. Slightly updated 2004 in paper cs/0412059 on arXiv.

Jonathan Ginzburg. 2012. *The Interactive Stance: Meaning for Conversation*. Oxford University Press, Oxford.

Jonathan Ginzburg, Robin Cooper, and Tim Fernando. 2014. Propositions, questions, and adjectives: a rich type theoretic approach. In *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, pages 89–96, Gothenburg, Sweden. Association for Computational Linguistics.

Jonathan Ginzburg, Chris Eliasmith, and Andy Lücking. 2024. Swann's name: towards a dialogical brain semantics. In *Proceedings of the Trentologue, the Twenty Seventh Workshop on the Formal Semantics and Pragmatics of Dialogue*.

Jonathan Ginzburg, Zulipiye Yusupujiang, Chuyuan Li, Kexin Ren, Aleksandra Kucharska, and Paweł Łupkowski. 2022. Characterizing the response space of questions: data and theory. *Dialogue and Discourse*. https://journals.uic.edu/ojs/index.php/dad/article/download/11531/10757.

Jan Gosmann and Chris Eliasmith. 2019. Vector-derived transformation binding: An improved binding operation for deep symbol-like processing in neural networks. *Neural Computation*, 31(5):849–869.

Geoffrey E. Hinton. 1990. Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46(1):47–75.

Hans Kamp. 2024. Can't believe it went by so fast. *Annual Review of Linguistics*, 10:1–16.

Staffan Larsson, Robin Cooper, Jonathan Ginzburg, and Andy Lücking. 2023. TTR at the SPA: Relating type-theoretical semantics to neural semantic pointers. In *Proceedings of the 4th Natural Logic Meets Machine Learning Workshop (NALOMA23)*. Association of Computational Linguistics.

Andy Lücking and Jonathan Ginzburg. 2022. Referential transparency as the proper treatment of quantification. *Semantics and Pragmatics*, 15:4.

Tony A. Plate. 2003. *Holographic Reduced Representation*. CSLI Publications, Stanford, CA.

Aarne Ranta. 1994. *Type-Theoretical Grammar*. Clarendon Press, Oxford.

Kenny Schlegel, Peer Neubert, and Peter Protzel. 2022. A comparison of vector symbolic architectures. *Artificial Intelligence Review*, 55(6):4523–4555.

Paul Smolensky. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216.

Terrence C. Stewart, Yichuan Tang, and Chris Eliasmith. 2011. A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research*, 12:84–92.

Aaron R Voelker and Chris Eliasmith. 2023. Programming neuromorphics using the neural engineering framework. In *Handbook of Neuroengineering*, pages 1519–1561. Springer.