COGNITIVE SCIENCE
A Multidisciplinary Journal

Cognitive Science (2015) 1–40

ISSN: 0364-0213 print / 1551-6709 online
DOI: 10.1111/cogs.12261

Biologically Plausible, Human-Scale Knowledge Representation

Eric Crawford,[a] Matthew Gingerich,[b] Chris Eliasmith[a]

[a]*Computational Neuroscience Research Group, University of Waterloo*
[b]*University of British Columbia*

Received 23 August 2013; received in revised form 17 February 2015; accepted 3 March 2015

Abstract

Several approaches to implementing symbol-like representations in neurally plausible models have been proposed. These approaches include binding through synchrony (Shastri & Ajjanagadde, 1993), "mesh" binding (van der Velde & de Kamps, 2006), and conjunctive binding (Smolensky, 1990). Recent theoretical work has suggested that most of these methods will not scale well, that is, that they cannot encode structured representations using any of the tens of thousands of terms in the adult lexicon without making implausible resource assumptions. Here, we empirically demonstrate that the biologically plausible structured representations employed in the Semantic Pointer Architecture (SPA) approach to modeling cognition (Eliasmith, 2013) do scale appropriately. Specifically, we construct a spiking neural network of about 2.5 million neurons that employs semantic pointers to successfully encode and decode the main lexical relations in WordNet, which has over 100,000 terms. In addition, we show that the same representations can be employed to construct recursively structured sentences consisting of arbitrary WordNet concepts, while preserving the original lexical structure. We argue that these results suggest that semantic pointers are uniquely well-suited to providing a biologically plausible account of the structured representations that underwrite human cognition.

Keywords: Knowledge representation; Connectionism; Neural network; Biologically plausible; Vector symbolic architecture; WordNet; Scaling

1. Introduction

One of the central challenges for contemporary cognitive modeling is scaling. As Geoff Hinton recently remarked in his address to the Cognitive Science Society, "In the Hitchhiker's Guide to the Galaxy, a fearsome intergalactic battle fleet is accidentally eaten by a small dog due to a terrible miscalculation of scale. I think that a similar fate

Correspondence should be sent to Eric Crawford, Computational Neuroscience Research Group, University of Waterloo, 200 University Ave West, Waterloo, Ontario, Canada, N2L 3G1. E-mail: e2crawfo@uwaterloo.ca

awaits most of the models proposed by Cognitive Scientists" (Hinton, 2010, p. 7). Whether or not we agree, this observation can at least be taken as a challenge for cognitive modelers: Will the principles demonstrated in small-scale cognitive models scale up to the complexity of a human-sized cognitive system?

This scaling problem has often been thought to be a special challenge for biologically inspired approaches to cognitive modeling (Hadley, 2009; Jackendoff, 2002). This is because the basic principles employed in such models do not provide a straightforward characterization of structured representations. Consequently, it is reasonable to wonder how such principles will ultimately be able to account for *human-scale* structured representations, which they clearly must do if they hope to provide convincing explanations of cognitive behavior. This same concern is not as immediate for symbolic approaches, which typically take structured representations to be primitive (Anderson, 2007; Pylyshyn, 1984).

In this paper, we present a novel method for representing structured knowledge in biologically plausible neural networks and show that it alone is capable of scaling up to a human-sized lexicon. This approach combines a method for encoding structured knowledge in vectors with a framework for building biologically realistic neural models capable of representing and transforming those vectors. In previous work, we have demonstrated that this approach meets many of the challenges that have been posed for connectionist accounts of structured representation, including the ability to account for the systematicity, compositionality, and productivity of natural languages, as well as the massive binding problem and the rapid variable creation problem (Eliasmith, 2013). That work has also given theoretical reasons to think that this approach will scale better than others; here our focus is on empirically demonstrating that claim. We achieve this by using our method to encode the human-scale knowledge base known as WordNet in a spiking neural network, and we show that, unlike past approaches, this network places plausible neural resource demands given what is known about the size of relevant brain areas.

The remainder of the paper is organized as follows. In Section 2, we review past connectionist approaches to the problem of representing structure, and we discuss recent criticisms of those approaches which suggest that they will not scale. In Section 3 we introduce the concept of a semantic pointer, the main type of representation employed by our approach. In Section 4, we present holographic reduced representations (HRRs), a vector algebra capable of encoding structured representations in vectors, which we use to create semantic pointers. In Section 5, we show how to use these semantic pointers to encode WordNet and outline an algorithm for extracting the relational information stored in the resulting encoding. In Section 6, we show how to build a biologically plausible neural network based on this extraction algorithm. In Section 7, we present the details of this neural network and show that it uses far fewer neural resources than the previously discussed approaches.

In Section 8, we demonstrate the capabilities of both the abstract extraction algorithm and its neural implementation by subjecting them to a number of experiments designed to confirm that WordNet is accurately encoded. In particular, these experiments show (a) that structural information can be extracted from arbitrary WordNet concepts, (b) that

hierarchies of arbitrary depth within WordNet are correctly represented, and (c) that the network can be used to extract the constituents of sentences involving arbitrary WordNet concepts. In Section 9, we conclude with an investigation of how our approach is able to achieve its superior scaling, as well as an exploration of how this work relates to theoretical debates in the cognitive science literature.

2. Past approaches

There have been many approaches to representing structure in connectionist networks. We consider three of the most successful: binding through synchrony, "mesh" binding, and tensor product binding.

2.1. Binding through synchrony

The suggestion that structured cognitive representations could be constructed using binding through synchrony (Shastri & Ajjanagadde, 1993) was imported into cognitive modeling from the earlier hypothesis that feature binding in vision can be accounted for by the synchronization of spiking neurons in visual cortex (von der Malsburg, 1981).

In their SHRUTI architecture, Shastri and Ajjanagadde (1993) demonstrated that exploiting synchrony can provide a solution to the variable binding problem in several simple examples. More recently, binding through synchrony has seen a revival in the LISA (Hummel & Holyoak, 2003) and DORA (Doumas et al., 2008) architectures, which focus on representing structures for analogical reasoning.

In all of these models, the temporal relationships between connectionist nodes are employed to represent structured relations. In DORA and LISA specifically, a structured representation such as bigger (Max, Eve) is constructed out of four levels of representation. The first level consists of nodes representing "sub-symbols" (e.g., furry, female, etc.). The second level consists of units connected to a set of sub-symbols relevant to defining the meaning of the second-level term (e.g., Max is connected to furry, Eve to female, etc.). The third level consists of "sub-proposition" nodes that bind roles to objects (e.g., Max+larger, or Eve+smaller, etc.). The fourth level consists of proposition nodes that bind sub-propositions to form whole propositions.

As has been argued in more detail elsewhere, this kind of representational scheme will not scale well (Eliasmith, 2013; Stewart & Eliasmith, 2012), because the number of nodes needed to support arbitrary structured representations over even small vocabularies (e.g., 2,000 lexical items) is larger than the number of neurons in the brain.[1]

Notably, this criticism is not problematic because of the use of synchrony per se, but rather because of the way binding has been mapped to network nodes. However, it has also been suggested that synchrony itself will not scale well to binding complex structures (O'Reilly & Munakata, 2000; Stewart & Eliasmith, 2012).

2.2. Mesh binding

A different approach to structure representation has been taken by van der Velde and de Kamps (2006) in their work on the neural blackboard architecture (NBA). To avoid the exponential growth in resources needed for structure representation in a DORA-like scheme, the NBA employs "neural assemblies." These assemblies are temporarily bound to particular symbols using a "mesh grid" of neural circuits (e.g., bind(noun1, Max)). Larger structures are then built by binding these assemblies to roles using a gating circuit (e.g., gate(agent1, bind(noun1, Max))). Neural assemblies that bind roles (and hence their gated word assemblies) are used to define higher level "structure assemblies." Such structure assemblies can be used to represent sentential structures.

The use of temporary binding in this manner significantly reduces the resource demands of this approach compared to the synchrony-based approaches. However, it does not reduce the demands sufficiently to make the NBA neurally plausible. As argued in Stewart and Eliasmith (2012), and demonstrated in more detail in Eliasmith (2013), in order to represent simple sentences of the form *relation(agent, theme)* from a vocabulary of 60,000 terms, this approach requires roughly 480 cm^2 of cortex, approximately one-fifth of total cortical area.[2] This is much larger than the combined sizes of "naming" cortex (about 7 cm^2; [1,000]), Wernicke's and Broca's areas (about 20 cm^2 each; Ojemann et al., 1989), and the remaining parts of the language "implementation system" (supramarginal gyrus, angular gyrus, auditory cortex, motor cortex, and somatosensory cortex, about 200 cm^2; Dronkers et al., 2000). Critically, these areas do far more than represent structure; they account for phonological processing, oral motor control, grammatical processing, sentence parsing and production, etc. Consequently, while the NBA has improved scalability compared to DORA, it remains implausible.

2.3. Tensor product binding

The final approach we consider, first proposed by Smolensky (1990), is the earliest and best-known member of the class of proposals broadly called vector symbolic architectures (VSAs; Gayler, 2003). In general, these approaches represent symbols with vectors, and propose some kind of non-linear vector operation to bind two vectors together. Later, the constituents of the binding can be extracted using an unbinding operation. Smolensky's architecture employs the tensor product as the binding operation, which has the advantage that it allows the constituents of a binding to be perfectly extracted.

In terms of scaling, this approach has the benefit that symbols and propositions can be represented by patterns of neural activity, alleviating the need for devoted neural resources for each proposition (as in the case of synchrony based approaches) and complex gating mechanisms (as in the case of the mesh binding approaches). However, the use of the tensor product as a binding operator creates a separate scaling issue. Because the tensor product of two n-dimensional vectors is an n^2-dimensional vector, this framework scales exponentially poorly as the depth of the encoded structure increases. For example, Eliasmith (2013) shows that encoding a two-level sentence such as "Bill

believes that Max is larger than Eve," where lexical items may have hierarchical relations of depth two or more, will require approximately 625 cm^2 of cortex.[3]

We now present the details of our approach for connectionist structured representation, with the aim of showing that, unlike the approaches explored here, it can encode a human-scale lexicon using a plausible number of neurons.

3. Semantic pointers

Semantic pointers are neurally realized vector representations generated through a compression method, and are typically of a high dimensionality (Eliasmith, 2013). In general, semantic pointers are constructed by compressing information from one or more high-dimensional vector representations, which can be semantic pointers themselves. The newly generated semantic pointer has a dimensionality that is less than or equal to the dimensionality of its constituents. Semantic pointers can be subsequently decompressed (or "dereferenced") to recover (much of) the original information.

Examples of compression that lowers dimensionality and loses information abound in the digital world. Jpegs, mp3s, and H.264 videos are all examples of "lossy" compression. These methods are lossy because, from an information theoretic perspective, the decompression of the compressed vector contains less information than the original uncompressed vector. However, from a psychophysical perspective, the pre-compressed and reconstructed vectors can be nearly indistinguishable. The reason such compression methods are ubiquitous is because they can massively decrease the amount of data that must be manipulated or transmitted, while preserving the essential features of that data. Semantic pointers are proposed to play an analogous role in our mental lives.

Because semantic pointers are compact ways of referencing large amounts of data, they function similarly to "pointers" as understood in computer science. Typically, in computer science a pointer is the address of a large amount of data stored in memory. Unlike the data in memory, pointers are easy to transmit, manipulate and store, because they occupy a small number of bytes. Hence, pointers can act as an efficient proxy for the data they point to. Semantic pointers are proposed to provide the same kind of efficiency benefits in a neural setting.

Unlike pointers in computer science, however, semantic pointers are *semantic*. That is, they are systematically related to the information that they reference, because they were generated from that information via compression. This means that semantic pointers carry similarity information that is derived from their source (unlike computer science pointers). If two uncompressed structures are similar along some dimension, then their compressed semantic pointers will also be similar, given an appropriate compression method and similarity measure.

The similarity relations between semantic pointers are best thought of as capturing "shallow" semantics. That is, semantics that can be read off of the surface features of the semantic pointers themselves. To get at "deep" semantics—for example, semantics dependent on subtle structural relations of the uncompressed data—it can be crucial to more directly

compare the uncompressed states. In many ways, this distinction between shallow and deep semantics is reminiscent of shallow and deep processing proposed in dual-coding theory (Paivio, 1986). Typically, this older distinction is taken to map onto the distinction between verbal and perceptual processing (Glaser, 1992). Recent fMRI and behavioral experiments are supportive of this view (Simmons et al., 2008; Solomon & Barsalou, 2004). Semantic pointers can be thought of as a computational specification of this distinction.

In sum, semantic pointers are neurally realized, compressed (and hence efficient) representations of higher dimensional data. They carry surface semantics, for which similarity can be cheaply computed, and they can be decompressed to access deeper semantics with additional computation. We note that we are not alone in finding it useful to understand efficient neural processing in terms of pointer-like structures. For example, both Kriete et al. (2013) and Schröder et al. (2014) have also found merit in the idea.

In this paper, we primarily use semantic pointers for their deep semantics and make little use of the shallow semantics. However, the shallow semantics are nonetheless present, which has important theoretical consequences and leaves open a number of interesting extensions that we discuss in Section 9.

Use of semantic pointers requires the specification of both a compression algorithm and a corresponding decompression algorithm. For example, in the vision system of the Spaun model, both compression and decompression take the form of a generative, hierarchical vision model (Eliasmith et al., 2012). A semantic pointer for a visual scene is created by running the model "forward," extracting a relatively low-dimensional representation that captures the scene's important features. The full visual scene can later be approximately reconstructed by running the model "backward," with the top of the hierarchy clamped to the desired semantic pointer. In the current study, we use compression algorithms that are better suited to structured representations. In particular, we use the operations provided by HRRs, a VSA.

In Section 6, we show how to use the neural engineering framework (NEF) to build spiking neural networks that represent and transform high-dimensional vectors, providing a neural implementation of this form of semantic pointer.

4. Holographic reduced representations

Holographic reduced representations are a type of VSA, and, as such, constitute a means of representing structured knowledge in a vector format. They have seen significant use in cognitive modeling; for instance, Murdock (1993) uses HRRs to provide an account of human working memory, and Jones and Mewhort (2007) use them to construct vector-space representations of English words from text, which are able to account for a number of classic empirical results.

HRRs have some similarities to Smolensky's tensor product technique, but they use a compressive binding operator which allows the dimensionality of the representations to remain constant as the depth of the encoded structure increases. This is an important difference which lends HRRs superior scaling properties. We should note that there are

other VSAs that share this feature of non-expanding dimensionality (e.g., Binary Spatter Codes; Kanerva [1994]). HRRs are used in the current work primarily because it is well-understood how to neurally implement the vector operations they require (Stewart, Bekolay, & Eliasmith, 2011).

Here, we briefly sketch the components of the HRR vector algebra. See Plate (1995) for a more complete introduction to HRRs, and Plate (2003) for an in-depth treatment of the mathematics involved as well as a number of applications. To begin, the basic elements of the structure that we want to represent are each assigned a random D-dimensional vector, where D is fixed ahead of time. We can then use the operations specified by the HRR formalism to create vectors encoding structured combinations involving those basic elements. Other operations can later be applied to the structured vectors to extract the constituent vectors. In this section, we discuss the necessary vector operations and their properties, before going on to explicitly show how they can be used together to encode structured information. The three vector operations specified by the HRR algebra are *circular convolution*, *vector addition*, and *involution*.

4.1. Circular convolution

Circular convolution, represented by the $\circledast$, plays the role of a *binding* operator. For two vectors $\mathbf{x} = [x_{(0)}, \ldots, x_{(D-1)}]$ and $\mathbf{y} = [y_{(0)}, \ldots, y_{(D-1)}]$, and $j \in \{0,\ldots,D-1\}$, the jth element of $\mathbf{x} \circledast \mathbf{y}$ is:

$$(\mathbf{x} \circledast \mathbf{y})_{(j)} = \sum_{k=0}^{D-1} \mathbf{x}_{(k)} \mathbf{y}_{(j-k),}$$

where the indices are taken modulo D. The circular convolution of two vectors is dissimilar to both of them, using the dot product as a measure of similarity.

4.2. Vector addition

Vector addition plays the role of a *superposition* operator. In particular, it allows multiple bindings to be stored in a single vector. The jth element of $\mathbf{x} + \mathbf{y}$ is:

$$(\mathbf{x} + \mathbf{y})_{(j)} = \mathbf{x}_{(j)} + \mathbf{y}_{(j).}$$

Vector addition returns a vector that is similar to both of its inputs, again using the dot product as a measure of similarity.

4.3. Involution

The third HRR operation, involution, is represented by an overbar (e.g., $\bar{\mathrm{x}}$). The jth element of $\bar{\mathrm{x}}$ is given by:

$$\overline{\mathbf{x}}_{(j)} = \mathbf{x}_{(-j)},$$

where the indices are again taken modulo D. Put simply, the first element of the vector stays in place, and the remaining elements are reversed. For example, if $\mathbf{x}$ = [1,2,3,4,5], then $\bar{x}$ = [1,5,4,3,2]. Involution is the approximate inverse of a vector with respect to circular convolution. Specifically, for two vectors $\mathbf{x}$ and $\mathbf{y}$, we have $\mathbf{x} \circledast \mathbf{y} \circledast \bar{\mathbf{y}} \approx \mathbf{x}$. It can thus be thought of as an *unbinding* operator, since it facilitates the extraction of the constituents of bindings. Circular convolution does have an exact inverse but, for reasons outlined in detail in Plate (2003), it performs poorly when $\mathbf{x}$ is noisy. Because we will later be concerned with neural representations of these vectors, which are inherently noisy, throughout this work we employ involution rather than the exact inverse.

The circular convolution, vector addition, and involution operations can be thought of as vector analogs of the familiar algebraic operations of multiplication, addition, and taking the reciprocal, respectively. Indeed, they have many of the same algebraic properties. For example, circular convolution is commutative, associative, and distributes over vector addition, similar to multiplication. The mathematical details of these operations, and, in particular, why involution is the approximate inverse of convolution, can be found in either of the above references on HRRs.

4.4. Semantic pointers for structured representations

Together, these operations permit the construction of vectors that represent complex structure. Most usefully for the current work, we can construct a vector which stores multiple pairs of other vectors. Later, given some query vector, we can determine which vector it is paired with in the structured vector. For example, suppose we have six elements in our vocabulary, each of which has been assigned a vector, **a, b, c, d, e, f** and we want to store the pairs $\langle \mathbf{a, b} \rangle$,$\langle \mathbf{c, d} \rangle$ and $\langle \mathbf{e, f} \rangle$. We can use circular convolution and vector addition to do this as follows:

$$\mathbf{t} = \mathbf{a} \circledast \mathbf{b} + \mathbf{c} \circledast \mathbf{d} + \mathbf{e} \circledast \mathbf{f}. \tag{1}$$

An important note is that $\mathbf{t}$ has dimensionality D, the same as that of each of the vectors on the right-hand side of this equation. This is because both circular convolution and vector addition return vectors with the same dimensionality as their inputs. This feature is what sets HRRs apart from Smolensky's tensor product technique. In particular, it prevents the size of the vectors from undergoing a combinatorial explosion as the depth of the encoded knowledge structure increases, permitting the efficient representation of hierarchical structures. For instance, $\mathbf{t}$ itself could be included in another structured vector, which would also have dimensionality D.

Given a query vector, we can then use circular convolution and involution to retrieve an approximation of the vector it is paired with in $\mathbf{t}$. For example, to extract the vector that **a** is paired with, we compute:

$$
\begin{aligned}
&\mathbf{t} \circledast \overline{\mathbf{a}} \\
&\quad = (\mathbf{a} \circledast \mathbf{b} + \mathbf{c} \circledast \mathbf{d} + \mathbf{e} \circledast \mathbf{f}) \circledast \overline{\mathbf{a}} \\
&\quad = \mathbf{a} \circledast \mathbf{b} \circledast \overline{\mathbf{a}} + \mathbf{c} \circledast \mathbf{d} \circledast \overline{\mathbf{a}} + \mathbf{e} \circledast \mathbf{f} \circledast \overline{\mathbf{a}} \\
&\quad = \mathbf{b} \circledast \mathbf{a} \circledast \overline{\mathbf{a}} + \mathbf{d} \circledast \mathbf{c} \circledast \overline{\mathbf{a}} + \mathbf{e} \circledast \mathbf{f} \circledast \overline{\mathbf{a}} \\
&\quad \approx \mathbf{b} + \mathbf{d} \circledast \mathbf{c} \circledast \overline{\mathbf{a}} + \mathbf{e} \circledast \mathbf{f} \circledast \overline{\mathbf{a}}.
\end{aligned} \tag{2}
$$

$$
= \mathbf{b} + \mathit{noise}, \tag{3}
$$

Here we have used both the commutative and distributive properties of circular convolution. The extra terms in Eq. (2) can be treated as noise because **a, c, d, e** and **f** are randomly chosen vectors.

We can now see that these structured vectors can be treated as a special kind of compressed representation. We can think of the original, uncompressed vector as the concatenation of **a, b, c, d, e** and **f**, which has a dimensionality of $6D$. **t**, which has only D dimensions, thus represents a significantly compressed version of the original vector. We can then look at the process of circularly convolving **t** with the involution of a query vector as lossy decompression, since we extract a vector that is similar but not identical to part of the original, uncompressed vector. But notice that this is a type of decompression that is especially well-suited to structured representation, because we do not have to reconstruct the entire original vector at once. Instead, we can choose which bit of it to decompress by changing the query vector. Throughout this paper we refer to the neural implementation of these structured vectors as semantic pointers.

5. Encoding structured knowledge in semantic pointers

Thus, far we have seen how to vectorially encode structured representations at a small scale. In this section we significantly scale up this technique. We first introduce WordNet, a human-scale semantic network, and then show how to create a vector encoding of WordNet using semantic pointers.

5.1. Wordnet

To empirically demonstrate that our technique can scale up to the size of a human vocabulary, we require a structured knowledge base of sufficient magnitude. One approach would be to construct an arbitrary structured representation, perhaps using random graph techniques; however, there is no reason that such a representation would statistically resemble the structure of human knowledge. A better approach is to choose a sufficiently large structured representation constructed with human knowledge specifically in mind. Fortunately, there are projects that have taken up the monumental task of encoding human knowledge in machine readable form. Two of the most well-known such projects are Cyc and WordNet.

Cyc's aim is ostensibly to codify the entirety of common-sense knowledge in a machine-usable format, with potential applications ranging from medicine to machine learning. Cyc is truly a marvel of perseverance; according to Lenat (1995), more than a person-century of work has gone into the manual construction of at least a million "common sense axioms."

WordNet is another manually constructed database (Fellbaum, 1998; Miller, Beckwith, Fellbaum, Gross, & Miller, 1990), but with slightly more modest goals. It aims to be a lexical database of the English language instead of a database of the entirety of human knowledge. Due to its reduced scope and its much smaller set of basic relationships, WordNet tends to have applied structural features more consistently than Cyc. As well, Cyc organizes concepts abstractly with logical assertions and microtheories, whereas WordNet's design is intended to reflect the organization of concepts in a psychologically plausible way using a handful of common relationships. In total, WordNet contains 117,659 concepts and a high degree of hierarchical structure (e.g., *entity* can be reached from *dog* in 13 steps), suggesting it will be an adequate test of the scalability of our technique. For these reasons, we have chosen WordNet as the structured knowledge base that we will encode.

The basic unit in WordNet is a *synset*, a set of words that have the same meaning. Words that have multiple meanings are listed in multiple synsets. Each synset possesses a number of *relations*, each of which represents a semantic link between that synset and another synset. Relations are unidirectional; each has a source and a target. Each relation also has a type, the most prominent being hypernymy (roughly "isA") and holonymy (roughly "partOf"). These relation types can be further subdivided: hypernymy into *instance* and *class*, and partOf into *part,member*, and *substance*. As a concrete example, we show a subset of the relational structure of the *dog* synset:

$$dog = class(canine)\ and\ member\,(pack). \tag{4}$$

Here *dog* is the source of a *class* relation and a *member* relation. *canine* is the target of the *class* relation, and *pack* is the target of the *member* relation. Only the five relation-types we have mentioned are encoded in our model. The inverses of these relations are also implicitly included, although we do not test their extraction as this requires more complex control of signal flow that is beyond our present scope. The depiction of lexical relations found in WordNet is somewhat simplified, though it is sufficient for our purposes; a complete description of the simplifications made can be found in Fellbaum (1998).

5.2. Semantic pointers and WordNet

It is relatively straightforward to use semantic pointers to encode the relational structure of a WordNet synset as represented in Eq. (4). The first step is to fix a dimension D for our vectors. Previous investigations have shown that using 512 dimensions provides sufficient representational capacity to accommodate human-scale knowledge bases

(Eliasmith, 2013), so $D = 512$ in all the work presented here. We then assign each WordNet synset a D-dimensional vector called an *ID-vector*, chosen uniformly at random from the D-dimensional unit hypersphere, which acts as a unique identifier for that synset. Each relation-type (*class*, *member*, etc.) is also assigned a vector in the same way.

The next step is to encode the information about how synsets are related to one another. Each synset is thus assigned a second D-dimensional vector storing the relational information about the synset. In particular, this vector is a semantic pointer constructed using the technique from Section 4, where each pair stored in the vector corresponds to a relation belonging to the synset and consists of the vector for the relation-type and the ID-vector for the target of the relation. The following equation demonstrates this process for the *dog* synset:

$$\mathbf{dog_{sp}} = \mathbf{class} \circledast \mathbf{canine_{id}} + \mathbf{member} \circledast \mathbf{pack_{id}}. \tag{5}$$

where all variables are D-dimensional vectors. We have disambiguated the two vectors assigned to a synset by denoting ID-vectors with the **id** subscript, and semantic pointers with the **sp** subscript.

We can now use the combination of circular convolution and involution to access the relations that belong to *dog*. As an example, imagine we want to extract the synset that *dog* is related to via the *class* relation-type. We could achieve this by circularly convolving $\mathbf{dog_{sp}}$ with $\overline{\mathbf{class}}$:

$$\begin{aligned}
&\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}} \\
&= (\mathbf{class} \circledast \mathbf{canine_{id}} + \mathbf{member} \circledast \mathbf{pack_{id}}) \circledast \overline{\mathbf{class}} \\
&= \mathbf{class} \circledast \mathbf{canine_{id}} \circledast \overline{\mathbf{class}} + \mathbf{member} \circledast \mathbf{pack_{id}} \circledast \overline{\mathbf{class}} \\
&= \mathbf{canine_{id}} \circledast \mathbf{class} \circledast \overline{\mathbf{class}} + \mathbf{member} \circledast \mathbf{pack_{id}} \circledast \overline{\mathbf{class}} \\
&\approx \mathbf{canine_{id}} + \mathbf{member} \circledast \mathbf{pack_{id}} \circledast \overline{\mathbf{class}} \\
&= \mathbf{canine_{id}} + \mathit{noise}
\end{aligned} \tag{6}$$

yielding a vector that is similar to $\mathbf{canine_{id}}$.

One might wonder why we need ID-vectors at all; it might seem more straightforward to define the semantic pointers for a synset directly in terms of semantic pointers for related synsets. For example:

$$\mathbf{dog_{sp}} = \mathbf{class} \circledast \mathbf{canine_{sp}} + \mathbf{member} \circledast \mathbf{pack_{sp}}. \tag{7}$$

This is problematic for a number of reasons. The most prominent is that WordNet (and semantic networks in general) have directed cycles, and thus some of the semantic pointers would have to be defined in terms of one another, which has no obvious solution.

5.3. Sentences

We can also use this technique to create semantic pointers encoding sentences involving any of the terms in WordNet. In this case, we pair up sentence roles and synsets filling those roles, and store the corresponding vectors in a semantic pointer. This requires assigning random vectors to the roles, just as we have done for relation-types. If we have the roles *subject*, *verb*, and *object* with assigned vectors **subject**, **verb**, and **object**, respectively, then the semantic pointer for the sentence "dogs chase cats" would be:

$$\mathbf{sentence_{sp}} = \mathbf{subject} \circledast \mathbf{dog_{id}} + \mathbf{verb} \circledast \mathbf{chase_{id}} + \mathbf{object} \circledast \mathbf{cat_{id}},$$

Circular convolution and involution can later be used to extract the synset filling a given role in a sentence, similar to Eq. (6). For example, $\mathbf{sentence_{sp}} \circledast \overline{\mathbf{object}}$ will be a vector similar to $\mathbf{cat_{id}}$.

We can also encode sentences with multiple levels of recursive depth. To demonstrate, the recursively structured sentence "mice believe that dogs chase cats" will have the semantic pointer:

$$\begin{aligned}\mathbf{deep\_sentence_{sp}} = {} & \mathbf{subject} \circledast \mathbf{mouse_{id}} + \mathbf{verb} \circledast \mathbf{believe_{id}} + \\ & \mathbf{object} \circledast (\mathbf{subject} \circledast \mathbf{dog_{id}} + \mathbf{verb} \circledast \mathbf{chase_{id}} + \mathbf{object} \circledast \mathbf{cat_{id}}).\end{aligned}$$

Top-level constituents (e.g., $\mathbf{mouse_{id}}, \mathbf{believe_{id}}$) can be extracted in the usual way, while constituents of the embedded clause can be extracted by using a compound query vector. For example, $\mathbf{deep\_sentence_{sp}} \circledast \overline{(\mathbf{object} \circledast \mathbf{verb})}$ will be a vector similar to $\mathbf{chase_{id}}$. Importantly, because we are using circular convolution for binding, $\mathbf{deep\_sentence_{sp}}$ still has the same dimensionality as all its constituents.

5.4. Associative memory

The result of computing $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ as in Eq. (6) is insufficient in two ways. First, because involution is only an approximate inverse, and because of the other term present, $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ is only *similar to* $\mathbf{canine_{id}}$; in other words, there is noise that must be removed. Second, $\mathbf{canine_{id}}$ is not particularly useful on its own; it would be more useful to have $\mathbf{canine_{sp}}$, from which we could recursively extract further structural information. These problems can be solved simultaneously by an associative memory.

Associative memories store ordered pairs of vectors $\langle \xi, \eta \rangle$. In an analogy with computer memory, the first vector ξ can be thought of as an address, and the second vector η can be thought of as the information stored at that address. When the memory receives an input, if that input is sufficiently similar to some ξ, then the memory outputs the corresponding η. It is easy to see how this solves our problems if we let the ξ's be ID-vectors and the η's be semantic pointers: The associativity provides us with the semantic pointers instead of the ID-vectors, and the fact that the input only has to be *sufficiently similar* to some ξ solves the denoising problem.

Given N pairs of vectors to associate, $\langle \xi_k, \eta_k \rangle$ for $k \in 1 \ldots N$, the following simple algorithm implements this associative memory functionality:

Algorithm 5.1: Associate (input)

$sum \leftarrow 0$
for $k \leftarrow 1$ **to** N
$\begin{cases} similarity \leftarrow \text{DOT PRODUCT}(\xi_k, input) \\ scale \leftarrow 1 \; if \; similarity > threshold \; \textbf{else} \; 0 \\ sum \leftarrow sum + scale * \eta_k \end{cases}$
return(sum)

If *threshold* is set correctly, *scale* will be non-zero for exactly one value of k, call it k', and the output of the function will be $\eta_{k'}$. For example, if we were using a vocabulary that contained only the synsets *dog*, *canine*, and *pack*, then the pairs stored in the associative memory would be $\langle \mathbf{dog_{id}}, \mathbf{dog_{sp}} \rangle$, $\langle \mathbf{canine_{id}}, \mathbf{canine_{sp}} \rangle$, and $\langle \mathbf{pack_{id}}, \mathbf{pack_{sp}} \rangle$. Now recall Eq. (6), where we showed that:

$$\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}} \approx \mathbf{canine_{id}} + \mathbf{member} \circledast \mathbf{pack_{id}} \circledast \overline{\mathbf{class}},$$

Since $\mathbf{canine_{id}}$ is contained in $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ through vector addition, $\mathbf{canine_{id}}$ and $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ will be similar. On the other hand, $\mathbf{dog_{id}}$ does not appear in this equation at all, and because two random high-dimensional vectors have a low probability of being similar, $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ is unlikely to be similar to $\mathbf{dog_{id}}$. Finally, $\mathbf{pack_{id}}$ appears in $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ through circular convolution, which, as mentioned above, returns a vector that is dissimilar to its inputs. In short, with high probability $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ will be similar to $\mathbf{canine_{id}}$ and dissimilar to the other ID-vectors. Thus, when $\mathbf{dog_{sp}} \circledast \overline{\mathbf{class}}$ is given as input to the associative memory, *scale* will only be non-zero for the *canine* pair, and the output will be $\mathbf{canine_{sp}}$.

5.5. Extraction algorithm

To summarize, we encode WordNet by assigning every synset two vectors: a randomly chosen ID-vector and a semantic pointer encoding the synset's structural relations. Later, given a semantic pointer corresponding to a synset and some query vector corresponding to a relation-type, if the synset has a relation of the given type, then we can extract the semantic pointer for the target of that relation with the following algorithm:

*Algorithm 5.2: Extraction (*sp, query*)*

$inv\_query \leftarrow \text{INVOLUTION}(query)$
$noisy\_id \leftarrow \text{CIRCULAR-CONVOLUTION}(sp, inv\_query)$
$target\_sp \leftarrow \text{ASSOCIATE}(noisy\_id)$
return$(target\_sp)$

This exact same algorithm can also be used to extract the constituents of semantic pointers encoding sentences composed of WordNet synsets.

One potential issue with this algorithm is the way in which it handles synsets that have multiple relations of the same type (which is not uncommon in WordNet). For example, the synset *lion* is related to both *pride* and *panthera* via the *member* relation-type. When the Extraction Algorithm is run with $\mathbf{lion}_{\mathbf{sp}}$ and **member** as input, the output will be $\mathbf{pride}_{\mathbf{sp}} + \mathbf{panthera}_{\mathbf{sp}}$. This vector still contains all the relational structure of both *pride* and *panthera*, and can be used in further extractions without difficulty. Consequently, returning the sum of these two vectors in response to a *member* query is considered correct in the experiments we run in Section 8. We also note that the fact that this is an issue may be a quirk of WordNet; in human knowledge bases, concepts with multiple relations of exactly the same type may be rare or non-existent. For example, these two relations ($lion \xrightarrow{member} panthera$ and $lion \xrightarrow{member} pride$) could be given two different relation types, reflecting the two different domains of knowledge they are concerned with.

We now move on to neural implementation and show that given a list of pairs of ID-vectors and semantic pointers encoding WordNet, we can construct a spiking neural network that performs the Extraction Algorithm.

6. Neural implementation

Since our end goal is a scalable, biologically plausible system for representing and manipulating structured knowledge, our next step is to show how the Extraction Algorithm we have been discussing can be implemented in realistic spiking neurons. We begin by presenting a framework that provides a principled approach to constructing networks of spiking neurons that represent and transform high-dimensional vectors. We then show how this technique can be applied to create a network implementing both involution and circular convolution. Finally, we use a slightly more advanced application of this method to construct a neural associative memory which is able to map a noisy version of any ID-vector to the corresponding semantic pointer. These networks can be composed into a single network implementing the Extraction Algorithm from the previous section, permitting the representation and extraction of structured knowledge by biologically realistic neurons.

6.1. Neural representation and transformation

For the purpose of neural representation and computation, we employ the NEF, a set of methods for building biologically plausible neural models (Eliasmith & Anderson, 2003). These methods have been broadly employed to generate detailed spiking neural models of a wide variety of neural systems and behaviors, including the barn owl auditory system (Fischer, 2005; Fischer, Peña, & Konishi, 2007), parts of the rodent navigation system (Conklin & Eliasmith, 2005), escape and swimming control in zebrafish (Kuo & Eliasmith, 2005), tactile working memory in monkeys (Singh & Eliasmith, 2006),

decision making in humans (Litt et al., 2008) and rats (Laubach et al., 2010; Liu et al., 2011), and the basal ganglia system (Stewart et al., 2010, 2012). These methods also underlie the recent Spaun model, currently the world's largest functional brain model (Eliasmith et al., 2012). Here, we present a brief discussion of the aspects of the NEF that are required for neural structured representation using HRRs. In particular, we discuss the NEF's principles of neural *representation* and *transformation*. The NEF also provides principles for *dynamics*, facilitating the implementation of arbitrary dynamical systems in recurrent neural networks, though these are not required for the feedforward networks used in the present model. All figures in this section were created using the Nengo neural simulation software package, which is available online at http://nengo.ca/.

The core idea behind the NEF's approach to neural representation is that the activity of a population of neurons at any given time can be interpreted as representing a vector. Importantly, the dimensionality of the represented vector is, in general, not equal to the size of the neural population. A typical case would have the activity of a population of 40 neurons representing a two-dimensional vector. We now outline the details of the relationship between the activities of a neural population and the vector those activities are taken to represent.

Let E denote the dimensionality of the vectors that a given neural population is capable of representing. A basic assumption of the NEF is that each neuron in the population has a "preferred direction vector" of dimensionality E, essentially a direction in the population's "represented space," which the neuron responds to most strongly. For instance, this is a useful way to characterize the behavior of motor neurons. Georgopoulos, Lurito, Petrides, Schwartz, and Massey (1989) found that neurons in motor cortex of rhesus monkeys have a preferred arm movement direction, the direction being different for each neuron. These neurons become more active as the monkey's current arm movement direction approaches their preferred direction. The activities of these neurons, taken together, can be interpreted as representing the direction of arm movement in three-dimensional physical space. This idea is quite intuitive in motor cortex, as the represented vector is directly observable; however, this notion is useful in general, and the NEF extends it to all neural representation.

To formalize the notion of preferred direction vector, the NEF assumes that the activity of the ith neuron in a neural population can be written:

$$a_i(\mathbf{x}) = G_i(\mathbf{e}_i^{\mathrm{T}}\mathbf{x}), \tag{8}$$

where a_i is the activity of the neuron, G_i is the neuron's activation function, $\mathbf{e}_i$ is the neuron's preferred direction vector, and $\mathbf{x}$ is the E-dimensional input to the neural population. $\mathbf{e}_i^{\mathrm{T}}\mathbf{x}$ is the dot product between the neuron's preferred direction vector and the input vector, and acts as a measure of similarity.

The NEF works for arbitrary neural activation functions, so, in accordance with our goal of biological realism, we choose G_i to be a spiking leaky integrate-and-fire (LIF) activation function in all the work presented in this paper. The details of the LIF activation function can be found in the appendix. The LIF is used because it provides a conve-

nient balance between computational efficiency and biological realism, capturing the primary non-linearity in neural systems: the neural spike. Moreover, the Spaun model was composed largely of LIF neurons, and there it was found that the statistics of neural firing in many of the simulated brain areas matched well with observed statistics in their real counterparts (Eliasmith et al., 2012).

Equation (8) is referred to as the *encoding* equation because it describes how an input vector, in this case **x**, is encoded into the activities of a neural population. Preferred direction vectors will henceforth be called *encoding vectors* because of their central role in this process. When building networks using the NEF, the encoding vectors for a neural population are typically chosen uniformly at random from the unit hypersphere in the population's represented space. The encoding process is depicted in Fig. 1 for a neural population capable of representing two-dimensional vectors. Note that while only four

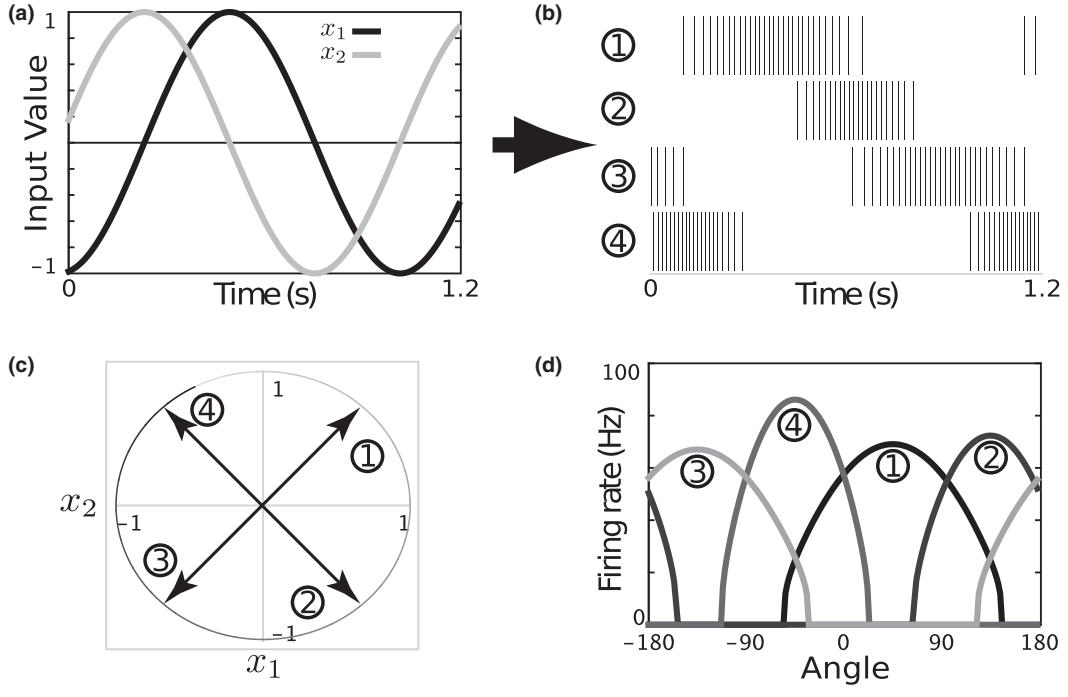

Fig. 1. NEF encoding. A population of four neurons encoding a two-dimensional vector. (a) Both dimensions of the input to the neurons, plotted over a period of 1.2 seconds. The input vector is determined by $x_1 = \sin(6t)$ and $x_2 = \cos(6t)$. (b) Spikes generated by four neurons driven by the input in (a), according to the encoding equation (Eq. (8)). (c) A different visualization of the input in (a). The input vector traces a clockwise path around a unit circle. Older inputs are in lighter gray. The encoding vectors of all four neurons are also shown. Comparing (b and c) shows that the neurons are most active when the input vector is closest to their encoding vectors. (d) The firing rate tuning curves of all four neurons as a function of the angle between the input vector and the encoding vector. Parameters for G_i, the neural activation function, are randomly chosen for each neuron, which is why the tuning curves are different heights and widths. (Reproduced from Eliasmith et al., 2012, with permission.)

neurons are used for ease of presentation, it typically requires more than four neurons to accurately represent two-dimensional vectors.

So far, we have shown how a vector can be *encoded* into neural activities. However, to fully characterize neural representation, we also need to say something about *decoding* those neural activities. In other words, given the activities of a neural population, how do we reconstruct the vector that the population is representing? The NEF assumes that a *linear* decoding is sufficient for capturing information transfer between neural populations. Given $a_i(\mathbf{x})$ for $i \in 1...N$, the set activities of a neural population, the vector represented by that population can be approximately reconstructed as:

$$\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x})\mathbf{d}_i, \tag{9}$$

where $\hat{\mathbf{x}}$ is a reconstruction of $\mathbf{x}$, and the $\mathbf{d}_i$ are a set of appropriately chosen column vectors (one for each neuron) called *decoding vectors*. These decoding vectors all have dimensionality E.

Decoding vectors that provide the best reconstruction can be found through a least-squares optimization process, outlined in detail in the appendix. Essentially, we find the $\mathbf{d}_i$ that minimize the equation:

$$\begin{aligned} Error &= \frac{1}{2}\int (\mathbf{x} - \hat{\mathbf{x}})^2 d\mathbf{x} \\ &= \frac{1}{2}\int (\mathbf{x} - \sum_i a_i(\mathbf{x})\mathbf{d}_i)^2 d\mathbf{x}. \end{aligned} \tag{10}$$

This optimization is typically performed offline, before the network is instantiated.

The decoding vectors found by minimizing Eq. (10) produce the optimal linear reconstruction of $\mathbf{x}$ from the activities of the neurons. In principle, however, we can also find decoding vectors that reconstruct $f(\mathbf{x})$, an arbitrary vector-valued function of $\mathbf{x}$. We denote these decoding vectors $\mathbf{d}_i^f$. The reconstruction of $f(\mathbf{x})$ is then computed from the activity of the neural population using the equation:

$$\widehat{f(\mathbf{x})} = \sum_i a_i(\mathbf{x})\mathbf{d}_{i.}^f \tag{11}$$

These decoding vectors are found by minimizing:

$$\begin{aligned} Error &= \frac{1}{2}\int (f(\mathbf{x}) - \widehat{f(\mathbf{x})})^2 d\mathbf{x} \\ &= \frac{1}{2}\int (f(\mathbf{x}) - \sum_i a_i(\mathbf{x})\mathbf{d}_i^f)^2 d\mathbf{x}, \end{aligned} \tag{12}$$

with respect to $\mathbf{d}_i^f$. In this more general case, the dimensionality of each decoding vector is equal to the dimensionality of the range of the function f. The accuracy of the reconstruction depends on the type of function and the tuning curves of the neurons. See Eliasmith and Anderson (2003) (section 7.3) for a discussion of this topic.

Thus far, we have primarily concerned ourselves with the question of neural representation, that is, how can an input vector be encoded in the activities of a neural population, and how can those activities be decoded to obtain a reconstruction of the encoded vector or a function thereof. However, representation alone is not terribly useful; to perform interesting information processing, we also need to be able to *transform* those representations. Fortunately, we've already defined the concepts required to understand the NEF's approach to neural transformation.

Suppose we have two neural populations A and B, and that there are all-to-all feedforward connections from the neurons in A to the neurons in B. Further suppose that we want to set the connection weights between A and B such that if, at a given time, A is representing some vector $\mathbf{x}$, then B will represent $f(\mathbf{x})$, where f is some arbitrary vector-valued function. In other words, we want to derive connection weights between the neurons in A and B such that $\widehat{f(\mathbf{x})}$ is first decoded from the activities in population A, and then encoded into the activities of population B. Conveniently, the NEF tells us that we can derive connection weights that achieve this in terms of the encoding vectors of B and decoding vectors of A for function f. Formally, we substitute $\widehat{f(\mathbf{x})}$ into Eq. (8) (modified for population B):

$$b_j(\mathbf{x}) = G_j(\mathbf{e}_j^{\mathrm{T}} \widehat{f(\mathbf{x})}), \tag{13}$$

$$= G_j(\mathbf{e}_j^{\mathrm{T}} (\sum_i a_i(\mathbf{x}) \mathbf{d}_i^f)), \tag{14}$$

$$= G_j(\sum_i (\mathbf{e}_j^{\mathrm{T}} \mathbf{d}_i^f) a_i(\mathbf{x})), \tag{15}$$

where a_i is the activity of the ith neuron in A, and b_j is the activity of the jth neuron in B. Thus, we have the activity of neuron j in B in terms of a weighted sum of the activities of the neurons in A. This indicates that weight on the connection from neuron i in population A to neuron j in population B should be:

$$\omega_{ij} = \mathbf{e}_j^{\mathrm{T}} \mathbf{d}_i^f, \tag{16}$$

which is simply the dot product, or similarity, between $\mathbf{e}_j$ and $\mathbf{d}_i^f$.

As a final note on transformation, if we additionally want to perform a linear transformation, represented by a matrix $\mathbf{L}$, on $f(\mathbf{x})$ (i.e., we want $\mathbf{L}f(\mathbf{x})$ to be represented in population B), then we can simply include $\mathbf{L}$ in the weight equation as follows:

$$\omega_{ij} = \mathbf{e}_j^{\mathrm{T}} \mathbf{L} \mathbf{d}_i^f. \tag{17}$$

This is the general weight equation for computing any combination of linear and non-linear functions between two neural populations.

To summarize, the process for creating two populations of neurons A and B, and deriving connection weights from A to B such that $\mathbf{L}f(\mathbf{x})$ is represented by population B whenever $\mathbf{x}$ is represented by population A, is as follows:

1. Create the neurons in populations A and B. For each neuron in each population, randomly choose parameters for the neural activation function and an encoding vector from the unit hypersphere.
2. Calculate the decoding vectors for population A that minimize Eq. (12).
3. Calculate the weight matrix between A and B using Eq. (17).

Simulations of spiking neural networks with connection weights derived using this technique are depicted in Fig. 2 for the identity function and the element-wise square function, with $\mathbf{L}$ set to the identity matrix in both cases.

This brief discussion does not capture the generality of the NEF, although it is sufficient for characterizing neural structured representation. As we are concerned with scaling, an important final note is that as more neurons are added to a population, the quality of its representation improves. Specifically, the mean-squared-error goes down as $1/N$ (Eliasmith & Anderson, 2003). Consequently, representations and transformations can be implemented to any desired precision, as long as there is a sufficient number of neurons. One of the main concerns of this paper is to determine whether the transformations and representations necessary for representing human-scale lexical structure can be done with a reasonable number of neurons. We now show how the NEF can be applied to create spiking neural networks that compute the operations required by the Extraction Algorithm.

6.2. Circular convolution in spiking neurons

Like any kind of convolution, circular convolution can be formulated as an element-wise multiplication in the Fourier space. As both the Fourier transform and its inverse are linear operators, circular convolution can be written in terms of linear operators and an element-wise multiplication:

$$\mathbf{x} \circledast \mathbf{y} = \mathbf{F}^{-1}(\mathbf{F}\mathbf{x} \diamond \mathbf{F}\mathbf{y})$$

where $\mathbf{x}$ and $\mathbf{y}$ are two arbitrary input vectors, $\diamond$ indicates an element-wise multiplication, and $\mathbf{F}$ and $\mathbf{F}^{-1}$ are matrices computing the Fourier transform and its inverse, respectively. A neural network that computes circular convolution using this formulation is shown in Fig. 3. Populations A and B represent the two input vectors and have feedforward connections to population C. These connections are set up to compute the Fourier transform

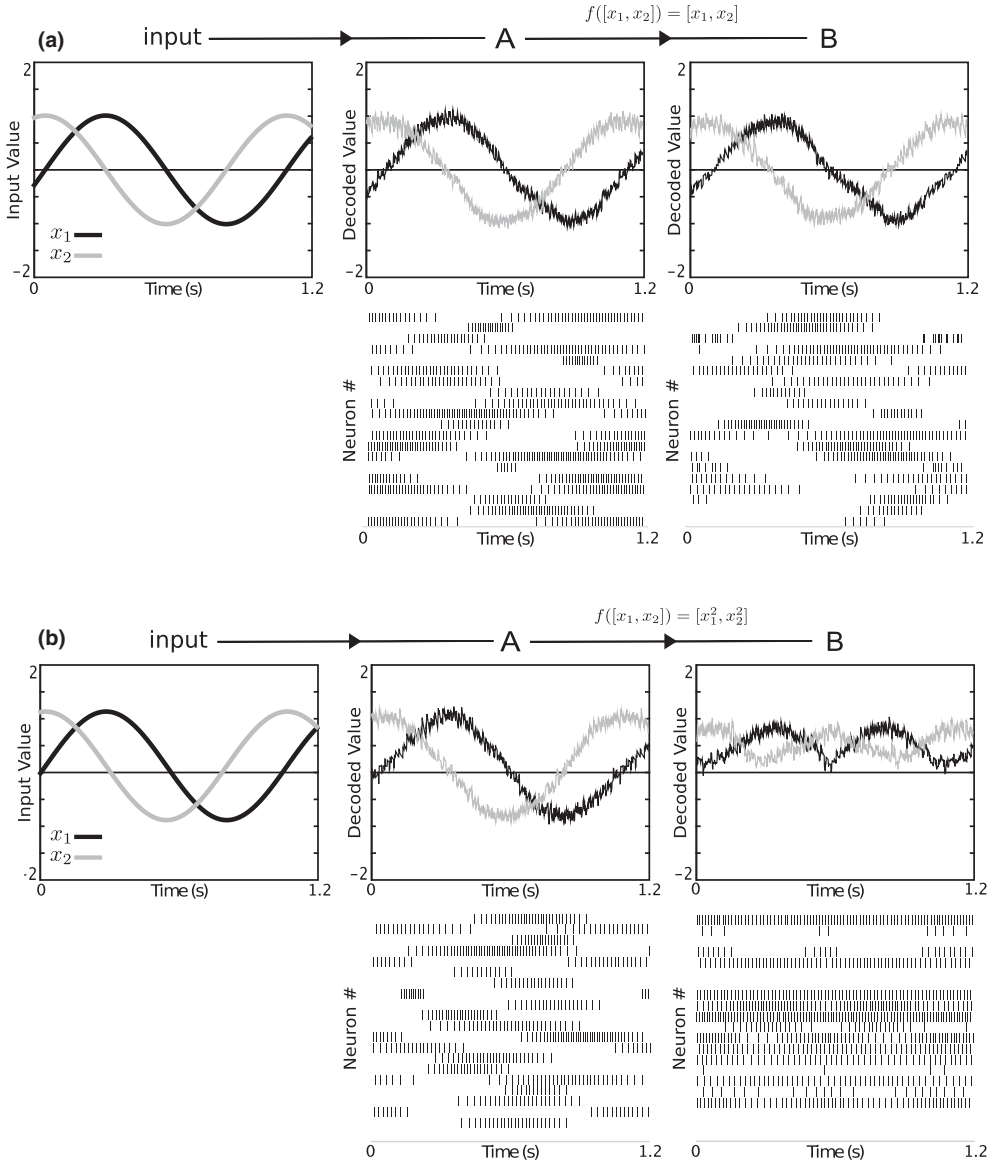

Fig. 2. (a and b) Each depicts a 1.2 s simulation of a spiking neural network designed using the NEF. In each case, the network consists of a neural population A with feedforward connections to a second neural population B. A and B each contain 20 neurons and each represent a two-dimensional vector. Input to population A in each network is given by $x_1 = \sin(6t)$ and $x_2 = \cos(6t)$, where t is current simulation time. In (a) the connection weights between A and B compute the identity function, and in (b) the weights compute the element-wise square function. To create each network, we first initialized the neurons in each population with random encoding vectors and parameters for the neural non-linearity G. We then computed connection weights implementing the desired functions by first minimizing Eq. (12) to find the appropriate decoding vectors, and subsequently using Eq. (16) to find the weights explicitly. Spike rasters for each population are shown. Neural decodings obtained by applying decoding vectors for the identity function to the spike rasters (using Eq. (9)) are shown above.

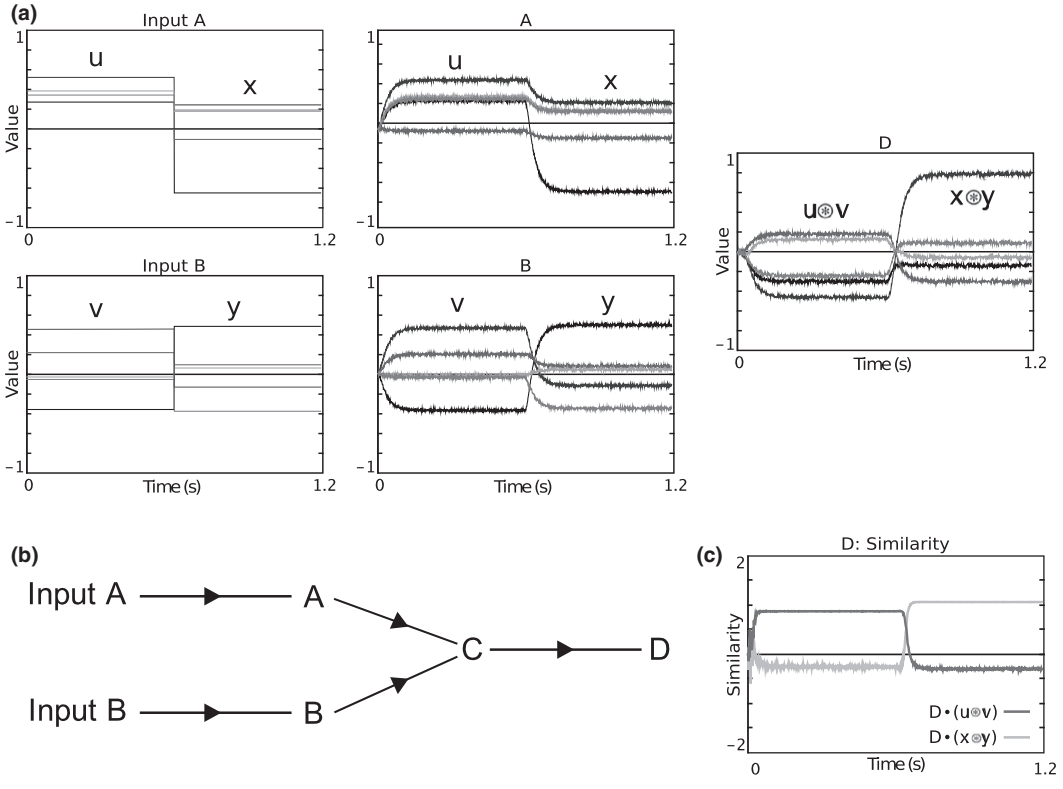

Fig. 3. Simulation of a neural circuit for computing circular convolution of two 10-dimensional vectors. A, B, C, and D are populations of spiking neurons. Connection weights between populations are derived using the techniques from Section 2 such that D represents the circular convolution of the two input vectors. Input vectors **u**, **v**, **x**, and **y** were randomly chosen from the 10-dimensional unit hypersphere. (a) Graphs showing input vectors and vectors represented by populations A, B, and D over a 1.2 s simulation, where the inputs change after 600 ms. In each graph, only 5 of the 10 dimensions are shown to reduce clutter. First column: the input vectors. Second column: vectors represented by the populations A and B, neural representations of the input vectors. Third column: vector represented by population D, which should be the circular convolution of the two input vectors. (b) Architecture of the neural circuit. The letters are populations of neurons and the arrows are all-to-all neural connections. (c) Similarity between the vector represented by population D and the vectors $\mathbf{u} \circledast \mathbf{v}$ and $\mathbf{x} \circledast \mathbf{y}$ over the 1.2 s simulation. If the circuit is correctly computing circular convolution, we would expect the similarity to $\mathbf{u} \circledast \mathbf{v}$ to be near 1 before 600 ms, and the similarity to $\mathbf{x} \circledast \mathbf{y}$ to be near 1 after 600 ms, which is clearly the case.

by multiplying by $\mathbf{F}$, causing C to represent the concatenation of the two Fourier transformed vectors. C is connected to population D, and these connections simultaneously compute the element-wise product of the two Fourier transformed vectors (using the appropriate decoding vectors), and take the inverse Fourier transform of the result by multiplying by $\mathbf{F}^{-1}$. The result is that $\mathbf{x} \circledast \mathbf{y}$ is represented in population D.

The weights for this network are found using Eq. (17). Here and throughout the paper, when showing connection weights, we will use i to index the neurons in the upstream

(pre-synaptic) population, and j to index neurons in the downstream (post-synaptic) population. The weights for the convolution network are:

$$\omega_{ij}^{A\to C} = \mathbf{e}_j^{\mathrm{T}}\mathbf{F}_1\mathbf{d}_i$$
$$\omega_{ij}^{B\to C} = \mathbf{e}_j^{\mathrm{T}}\mathbf{F}_2\mathbf{d}_i$$
$$\omega_{ij}^{C\to D} = \mathbf{e}_j^{\mathrm{T}}\mathbf{F}^{-1}\mathbf{d}_i^{\diamond}$$

where $d_i^{\diamond}$ are decoding vectors for the element-wise multiplication function for population C. Note that $\mathbf{F}_1 = \begin{bmatrix}\mathbf{F}^{\mathrm{T}} & \mathbf{0}\end{bmatrix}^{\mathrm{T}}$ and $\mathbf{F}_2 = \begin{bmatrix}\mathbf{0} & \mathbf{F}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$. This zero-padding is required because the default behavior for two neural populations converging on a single downstream population is for the vectors to be *added*. However, we want **Fx** and **Fy** to be *concatenated* in C, so that the network can subsequently perform element-wise multiplication. Zero-padding the Fourier transform matrices implements this concatenation.

Although this operation may seem complicated, it is surprisingly natural for neural computation in several important respects. First, it has been shown to be learnable in a spiking network (Stewart et al., 2011). Second, for 512-dimensional vectors it has been shown to result in connection matrices that respect known neural connectivity constraints (Eliasmith, 2013).

6.3. Involution in spiking neurons

As we saw in our initial discussion of the HRR algebra, the involution of a vector is an approximate inverse with respect to circular convolution. It can be computed by reversing all but the first element of the vector, which stays in place. Since this is a permutation, it is a linear operation, and therefore, there exists a matrix **V** which computes it. Consequently, there is a straightforward modification we can apply to our convolution network such that the second input is involuted before the Fourier transform is applied, resulting in a network that computes $\mathbf{x}\circledast\mathbf{y}$ rather than $\mathbf{x}\circledast\mathbf{y}$. Specifically, we change the connection weights between populations B and C to:

$$\omega_{ij}^{B\to C} = \mathbf{e}_j^{\mathrm{T}}\mathbf{F}_2\mathbf{V}\mathbf{d}_i, \tag{18}$$

We now have a neural network computing two of the three operations required by the Extraction Algorithm. The last component is a neural associative memory, which requires a slightly more nuanced application of the NEF.

6.4. Neural associative memory

Recall that the aim of the associative memory is to associate pairs of vectors $\langle\xi,\eta\rangle$, and for our purposes the ξ's are the ID-vectors and the η's are the semantic pointers. We now show how the NEF can be applied to create a spiking neural network capable of

efficiently implementing the associative memory functionality. The approach we employ here was first demonstrated by Stewart, Tang, and Eliasmith (2011) to build an *auto-associative* memory (where for each stored pair $\langle\xi,\eta\rangle$, we have $\xi = \eta$), though it can be trivially extended to implement a general hetero-associative memory. That paper demonstrated that networks built using this approach significantly outperform linear associators, direct function approximators, and standard multi-layer perceptrons in terms of both accuracy and scalability. Such networks also have an advantage in terms of biological plausibility, since they are implemented in spiking neurons.

This neural associative memory is essentially a neural implementation of the association algorithm presented in Section 4. Each pair to be associated is assigned a small (~ 20) neuron population. The encoding vectors of this population are set equal to ξ, and the neurons are given a high threshold so that they only spike when the input is sufficiently similar to ξ. The decoding vectors of the population are chosen to approximate a thresholding function, and η is used as a linear operator. The overall effect is that when a population is active, it outputs its assigned η, and a population is active if and only if the input is sufficiently similar to its assigned ξ. All these populations converge on a single output population, where the inputs are summed by the dendrites. In essence, each neural population computes, in parallel, one iteration of the loop in Algorithm 5.1. To be explicit, for sub-population k assigned the vector pair $\langle \xi_k, \eta_k \rangle$, the input and output weights are:

$$\omega_{ij}^{in} = \xi_k^T \mathbf{d}_i \qquad\qquad \omega_{ij}^{out} = \mathbf{e}_j^T \eta_k \mathbf{d}_i^{thresh}, \tag{19}$$

where $\mathbf{d}^{thresh}$ are decoding vectors for the thresholding function $f(x) = 1.0$ **if** $(x > 0.3)$ **else** 0.

We have described all of the techniques required to create a spiking neural network for extracting the constituents of semantic pointers. We claim that the network obtained by composing these two networks, such that the output of the involution/convolution network is fed into a neural associative memory, constitutes a neural implementation of the abstract Extraction Algorithm. In what remains, we present the details of the neural model and run experiments on it to determine how it performs at scale.

7. The neural model

The model consists of a network of 2,506,980 spiking neurons constructed using the techniques outlined above. Given a semantic pointer corresponding to a WordNet synset and a query vector corresponding to a relation-type, the network returns the semantic pointer corresponding to the target of the relation, implementing the Extraction Algorithm. The network can be used to traverse the WordNet hierarchy by running it recursively, with the output of one run used as input on the next run. Low-level details about the model and its parameters can be found in the Appendix.

A schematic diagram of the model is depicted in Fig. 4. The rectangles correspond to populations of spiking neurons, which represent and manipulate high-dimensional vectors. The dark gray population, which represents the concatenation of the Fourier transforms

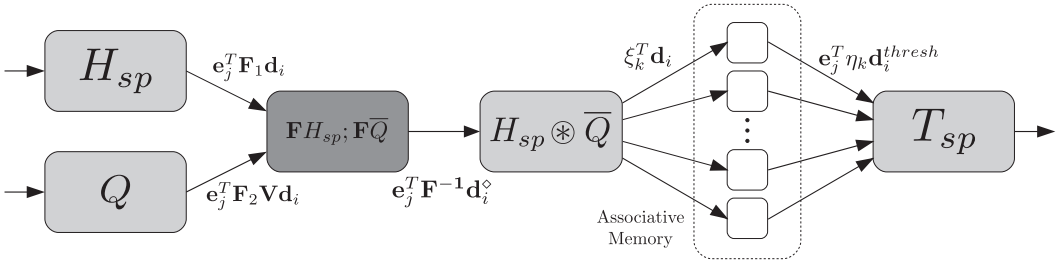

Fig. 4. The network of spiking neurons designed to implement the Extraction Algorithm from Section 5.5. Assume $\mathbf{H_{sp}} = \mathbf{Q}\circledast\mathbf{T_{id}} + \mathbf{R}\circledast\mathbf{U_{id}}$. The rectangles correspond to populations of spiking neurons and are labeled with the values we expect them to represent when the network is given $\mathbf{H_{sp}}$ and $\mathbf{Q}$ as input. Arrows represent all-to-all feedforward connections between populations and are labeled with the elements of the NEF-derived weight matrices mediating them. In these weight matrices, *i* always indexes neurons of the upstream population, *j* always indexes neurons of the downstream population, and *k* indexes pairs of vectors in our vector encoding of WordNet. Light gray populations represent 512-dimensional vectors. The dark gray population represents the concatenation of two Fourier transformed 512-dimensional vectors, and thus represents a 1,028-dimensional vector.

of the two input vectors, contains 51,400 neurons, and the four light gray populations contain 25,600 neurons each. The associative memory contains a separate 20-neuron population for each of the 117,659 synsets in WordNet. The grand total is thus 2,506,980 neurons, equivalent to approximately 14.7 mm^2 or 0.147 cm^2 of cortex (as there are about 170,000 neurons per mm^2; Eliasmith, 2013). This is much smaller than any of the approaches discussed in Section 2, all of which require on the order of 500 cm^2 of cortex or more. More significantly, ours is the only approach whose neural resource requirements do not contradict our empirical knowledge about the size of relevant brain areas. Consequently, if our experiments confirm that our network can accurately extract the relational structure from WordNet synsets, it will constitute a significant advance in the study of biologically plausible representations of structured knowledge.

The tasks of moving the output into the input for hierarchical traversals, controlling which vector is used as input to the query population, etc., are not neurally implemented here as they are peripheral to our central concern of representing human-scale structured knowledge in a biologically plausible manner. However, Spaun, a large scale, functional brain model constructed using the NEF, is evidence that it is possible to achieve this kind of control in a scalable spiking neural network (Eliasmith et al., 2012). Spaun also contains roughly 2.5 million neurons, which means adding in this control would at worst double our neuron counts, leaving the model well within the range of neural plausibility; moreover, Spaun contains several modules (e.g., vision and motor control) which are unrelated to the problem of representing structure, making this is an extremely conservative estimate.

Our model is a significant departure from the majority of connectionist work in that no online learning occurs; the connection weights implementing the required transformations are derived offline before the network is instantiated, using the NEF techniques. We believe the problem of large-scale connectionist knowledge representation is difficult enough that it

is useful to focus on representation, and we leave the question of learning for future work. We do note that it has been shown that circular convolution can be learned in spiking neurons using a biologically plausible learning rule (Stewart et al., 2011), and work is under way investigating how a neural associative memory of the kind used in our model could be learned from training data (Voelker, Crawford, & Eliasmith, 2014).

8. Experiments

We performed three experiments on both the abstract Extraction Algorithm and its neural implementation, to test whether WordNet is accurately encoded. For convenience, we will refer to both implementations as "models." A *trial* consists of using a model to answer a single question about the WordNet graph (the question is different for each experiment). A *run* consists of a group of trials. No information was added to or removed from either model's associative memory between experiments, demonstrating that both models are capable of performing all three tasks unmodified.

For each experiment, we execute 20 runs, calculate the performance on each run as the percentage of trials on which the model answered correctly, and report the mean performance over all the runs. To obtain distributional information about these results, we employ a bootstrapping method to obtain 95% confidence intervals.

Each trial consists of using a model for one or more extraction operations, where a semantic pointer and a query vector are presented as input and the algorithm outputs a vector (which may or may not be a semantic pointer). In the neural case, for each extraction operation the model was simulated for 100 ms with a simulation timestep of 1 ms, after which the vector represented by the rightmost population in Fig. 4 was taken to be the output of the model. Code for constructing the models and running the experiments is hosted online in a github repository at https://github.com/e2crawfo/hrr-scaling.

8.1. Experiment 1: Simple extraction

This experiment investigates the ability of a model to traverse a single edge in the WordNet graph. We present the model with a semantic pointer corresponding to a randomly chosen synset and the vector corresponding to a relation-type that the synset is known to possess, and see if the model outputs the semantic pointer corresponding to the target of that relation. For example, we might present the model with $\mathbf{dog}_{\mathbf{sp}}$ as the semantic pointer and **class** as the query vector, and expect the model to return $\mathbf{canine}_{\mathbf{sp}}$.

To be considered correct, the vector returned by the model must have a larger dot product with the correct semantic pointer than with any incorrect semantic pointer in the vocabulary, and this similarity must exceed a threshold of 0.7. The value of 0.7 is somewhat arbitrary, though it does ensure that the output vectors have sufficient fidelity to be put to further use. In this experiment, each run consists of 100 trials.

8.2. Experiment 2: Hierarchical extraction

The simple extraction experiment assesses the general accuracy of a model of knowledge representation, but it only tests individual relationship links. The hierarchy traversal experiment is designed to test a model's ability to traverse hierarchies of arbitrary depth in the WordNet graph.

To that end, we use the model to answer the following question: Given a starting synset, a goal synset, and a relation-type, can the goal synset be reached from the starting synset by following only links of the specified type? To have the model answer this question, we present it with the semantic pointer corresponding to the starting synset as well as the vector for the given relation-type. We then run the model and compare the output vector to the semantic pointer for the goal synset. If they are the same (their dot product is above a fixed threshold), then the model responds with a Yes. If not, we feed the output vector back into the model as the new semantic pointer and run the model again using the same query vector. This process is repeated until the model returns a vector with a norm below a fixed threshold, in which case the model responds with a No.

As an example, if the starting synset is *dog* and we follow only relations whose type is *class*, we first get *canine*, which in turn yields *carnivore*, followed by *placental mammal*, and so on, until the synset *entity* is finally reached in 13 links. The correct answer is Yes if and only if the goal synset is one of these synsets. Further concrete examples of possible queries and correct responses are given in Table 1. Our tests were performed using only the *class* relation-type as it is the most prominent in WordNet and permits the deepest traversals. Each run consists of 40 trials with an even split between positive and negative instances. A positive instance is one in which the goal synset can be reached from the starting synset in the WordNet graph, and the correct response is Yes. Results of a simulation where we test the neural model on an instance of the Hierarchical Extraction test are shown in Fig. 5.

8.3. Experiment 3: Extracting from sentences

The previous experiments have focused on relations included in WordNet. The current experiment is designed to go further, and demonstrate that we can use semantic pointers to encode arbitrary sentence-like objects possessing recursive structure, and that the model is capable of extracting the constituents of such constructions without modification.

We take the approach suggested in Section 5.3, modeling sentences as collections of roles paired with WordNet synsets as role-fillers. If each role is assigned a random vector,

Table 1
Examples of instances and correct responses in the Hierarchical Extraction test

Starting Synset	Target Synset	Relationship Type	Is Related?
dog	vertebrate	class	Yes
vertebrate	dog	class	No
dog	entity	class	Yes
dog	cat	class	No

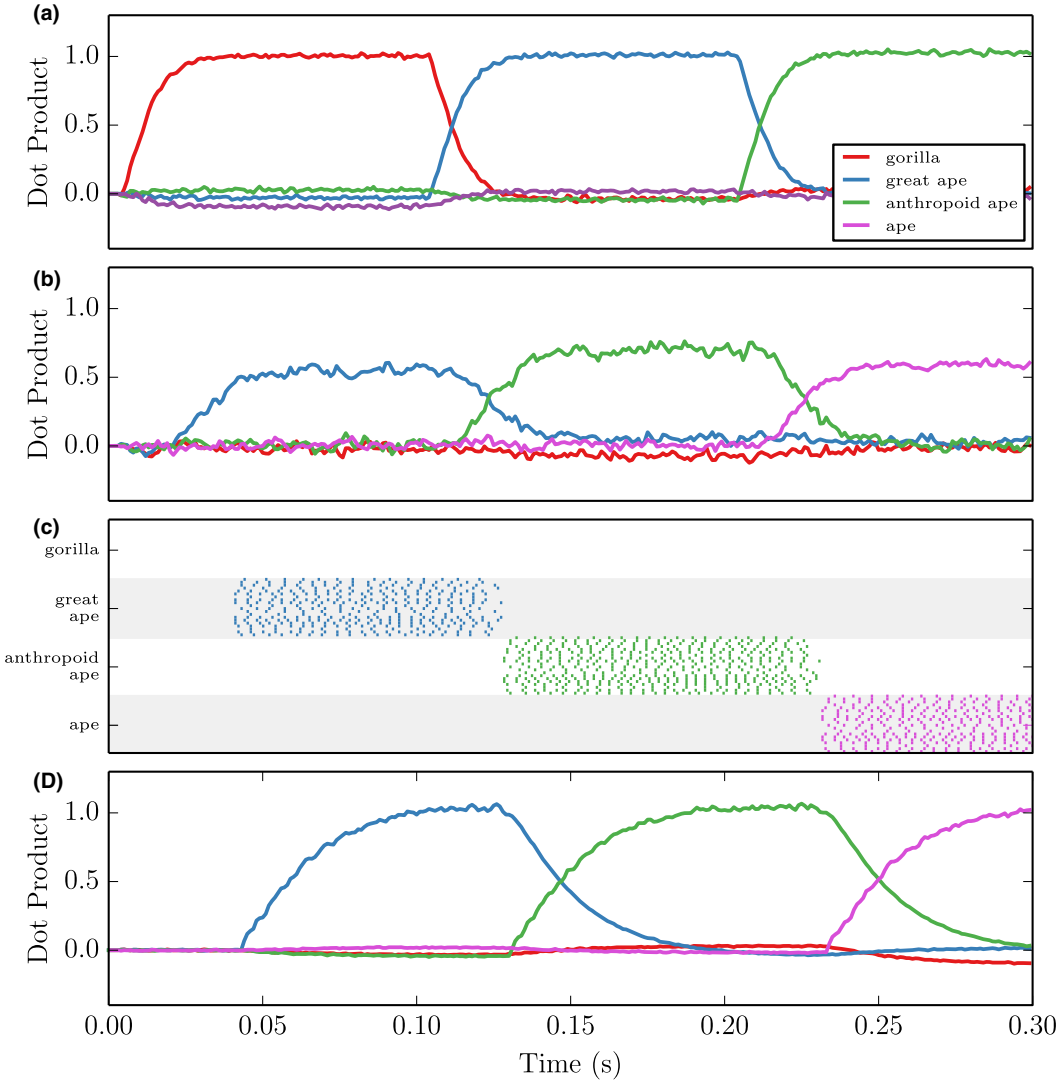

Fig. 5. Result of a simulation during which the network was used to perform the Hierarchical Extraction test. The starting synset is *gorilla*, the relation type is *class*, and the target synset is *ape*. Every 100 ms we compute the norm of the output vector and the dot product between the output vector and $\mathbf{ape}_{sp}$. If the norm is below a threshold, the network's response is No. If the dot product is above a threshold, the network's response is Yes. Otherwise, the output vector is fed back in as input, and the simulation continues. (a) Dot product between semantic pointers and the vector represented by the input population. (b) Dot product between ID-vectors and the vector represented by the population that feeds into the associative memory. (c) Spikes rasters for neurons in the association populations corresponding to the synsets listed in the legend. Association populations for all other WordNet synsets were included in the simulation but had no spiking activity. (d) Dot product between semantic pointers and the vector represented by the output population. The network can be seen to traverse the WordNet graph, starting from *gorilla* and eventually reaching *ape*. At 300 ms, the dot product of the output vector with $\mathbf{ape}_{sp}$ is above the threshold and the network responds (correctly) with Yes.

then such sentences can be encoded as semantic pointers just as we have previously encoded the relational structure of WordNet synsets as semantic pointers. For instance, recall that in Section 5.3, we showed that the sentence "mice believe that dogs chase cats" can be encoded as a semantic pointer in the following way:

$$\begin{aligned}\mathbf{sentence_{sp}} = {} & \mathbf{subject} \circledast \mathbf{mouse_{id}} + \mathbf{verb} \circledast \mathbf{believe_{id}} + \\ & \mathbf{object} \circledast (\mathbf{subject} \circledast \mathbf{dog_{id}} + \mathbf{verb} \circledast \mathbf{chase_{id}} + \mathbf{object} \circledast \mathbf{cat_{id}}).\end{aligned} \tag{20}$$

One difference here is that role vectors are chosen randomly from the set of so-called "unitary" vectors instead of from the unit hypersphere as with relation-type vectors. Unitary vectors have the special property that involution is their *exact* inverse (Plate, 2003), which permits the sentence constituents to be extracted with higher accuracy.

We emphasize that our neural network does not need to be modified in any way to extract the constituents of a semantic pointer encoding a sentence, even though the network will not have encountered such a semantic pointer previously. This is possible because the vectors that we want to extract from the sentence semantic pointers are, by construction, ID-vectors corresponding to WordNet synsets, which are stored in the neural network's associative memory.

On each trial of this experiment, we randomly generate a semantic pointer encoding a sentence according to the steps outlined in Fig. 6. Sentence-roles are included in sentences according to the probabilities in Table 2. We then test the model's ability to extract the constituents of the sentence. For the surface-level constituents, we present the model with the semantic pointer representing the sentence, and the appropriate role vector as the query vector. For example, if we present $\mathbf{sentence_{sp}}$ and **verb**, we should expect the model to return $\mathbf{believe_{sp}}$. The process is similar for constituents of the embedded clauses, except we use compound query vectors. If we present the model with $\mathbf{sentence_{sp}}$ and $\mathbf{object} \circledast \mathbf{subject}$, we expect it to output $\mathbf{dog_{sp}}$. Each run consists of 30 trials, and on each trial the performance is measured as the percentage of queries that the model responded to correctly, using the same correctness criteria as in Experiment 1. Results for extracting surface and embedded constituents are reported separately.

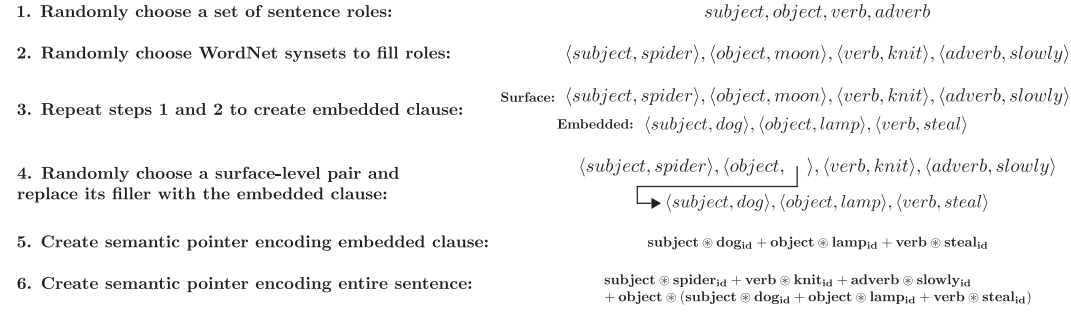

Fig. 6. Steps for creating semantic pointers for the sentence extraction experiment.

Table 2
Roles for sentence generation and their properties

Role Name	Probability of Occurrence	Part of Speech
Subject	1.0	Noun
Object	0.8	Noun
Verb	1.0	Verb
Adverb	0.6	Adverb
Subject adjective	0.3	Adjective
Object adjective	0.3	Adjective

8.4. Results

Results of the experiments are presented numerically in Table 3 and graphically in Fig. 7. Performance on all three tasks by both the abstract and neural implementations of the Extraction Algorithm is near 100%. The success of the abstract algorithm shows that WordNet is accurately stored in our vector encoding. The success of the neural implementation shows that no significant penalty is incurred by implementing the algorithm in spiking neurons. More generally, it shows that a spiking neural network is capable of extracting structure from any element of a human-scale vocabulary. Combined with the fact that, unlike previous approaches, this network places neural resource demands that are consistent with anatomical data, this constitutes the first biologically plausible neural implementation of a human-scale structured knowledge base.

9. Discussion

9.1. Scaling

We have presented the details of our approach to encoding structured representation and demonstrated empirically that it is capable of encoding a human-scale knowledge

Table 3
Numerical values for extraction performance

			95% CI			
Experiment	Type	% Correct	Lower	Upper	Runs	Trials (per Run)
Simple	Abstract	99.0	98.6	99.4	20	100
	Neural	99.2	98.9	99.3		
Hierarchical	Abstract	96.5	95.0	97.8	20	40
	Neural	98.5	97.8	99.3		
Sentence (surface)	Abstract	94.3	93.3	95.3	20	30
	Neural	97.2	96.3	97.9		
Sentence (embedded)	Abstract	95.1	94.2	95.9	20	30
	Neural	96.2	95.3	97.0		

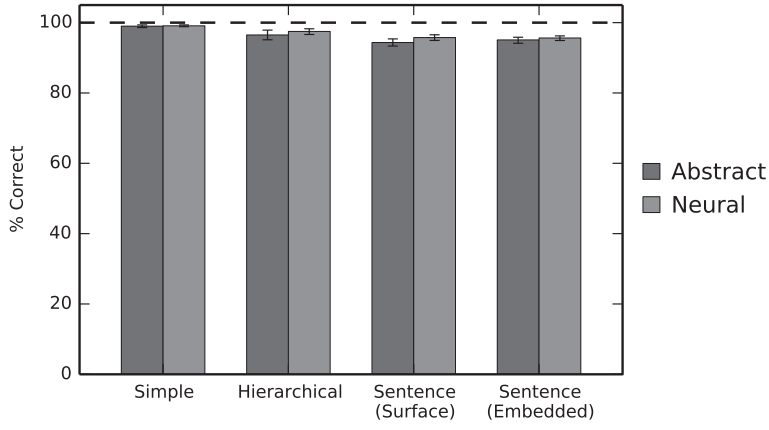

Fig. 7. Extraction performance. Error bars are 95% confidence intervals. On the Simple and Sentence experiments, chance performance is roughly 1/117,000%, while on the Hierarchical experiment it is 50%.

base with much more modest resource requirements than its competitors. Specifically, our model uses roughly 2.5 million neurons to encode a structured lexicon containing 117,659 words. Individual relations in the lexicon can be traversed with 99% accuracy, hierarchies with up to 13 levels can be traversed with 98% accuracy, and the network can be used to extract constituents of recursively structured sentences with 96% accuracy. Unlike past approaches, which use a minimum of 480 cm^2 of cortex to represent much simpler structures, the method as demonstrated here requires less than 1 cm^2 of cortex. Like past work, there are many aspects of linguistic processing that are not captured by our model. However, its modest use of cortical resources makes it plausible that it may do so with further development.

We believe that this improved scaling largely results from capturing structured representation using compressed, temporary signal processing states (i.e., semantic pointers encoding sentences or synsets), rather than using fixed neural resources. In contrast, synchrony-based approaches like DORA and LISA require a fixed neural node for every proposition that we require the network to be able to represent. The NBA alleviates this need by allowing bindings to be represented by the activation states of a neural "mesh." However, the substantial complexity required to implement this mesh results in scaling that is still implausible. Moreover, both synchrony-based approaches and the NBA are capable of representing only a few propositions at a time. This means that while they are capable of representing short-term bindings (such as the sentences in our sentence experiment), the long-term storage and retrieval of relations in a large, structured knowledge base is well beyond their abilities. In particular, it is unclear how they would accomplish the Hierarchical Extraction experiment.

Our approach bears some resemblance to Smolensky's tensor product VSA. However, we make a different set of tradeoffs which result in significantly improved scaling. In particular, we use a compressive binding operator, supported by an associative memory. This avoids the explosion in neural resources required by Smolensky's approach as the depth of

encoded structures increases. In particular, it permits us to encode and extract the constituents of recursively structured sentences without making implausible neural resource demands.

9.2. Theoretical considerations

Our model speaks to the long-standing debate regarding the relationship between classical, symbolic theories of mind and connectionist research. On one side of the debate are the researchers who take an implementational view of connectionism. These researchers hold that human behavior is best analyzed at a symbolic level, and that the role of connectionist research is to show how neurons might implement classical symbolic representations (Fodor & Pylyshyn, 1988; Jackendoff, 2002). The past approaches we discussed in Section 2 are all attempts to directly implement these classical symbols, where each element (e.g., a word) in a composite symbol (e.g., a sentence) is explicitly represented. On the other side of the debate are so-called eliminative connectionists who hold that human behavior can be accounted for without implementing a classical symbol system (Chalmers, 1990; Elman, 1991; Pollack, 1990). Upon first inspection, the approach we have presented here may appear implementational, since our neural network is essentially an implementation of the abstract Extraction Algorithm. However, semantic pointers, while possessing compositional properties, are not truly classical symbols, and consequently, we take our approach and its scalability as evidence in favor of non-classical architectures.

One reason that our approach is non-classical is that it is not necessary to perform the extraction operation on semantic pointers in order to do useful things with them, thanks to the shallow semantics that we mentioned in Section 3. For instance, semantic pointers that have similar relational structure will themselves be similar, which can be exploited to perform useful computation without first extracting the elements in the representation (Eliasmith, 2013). This is a hallmark of eliminative connectionism (Chalmers, 1990) and is, by definition, impossible in a classical symbolic architecture.

Another reason that we consider semantic pointers to be non-classical is that they are constructed through a lossy compression process. As a result, they do not contain complete information about their constituents. In particular, the result of the decompression operation (i.e., the involution/circular convolution combination) is only an approximation of a constituent vector. In contrast, tensor product vectors representing complex structures contain explicit representations of their constituents, and said constituents can be perfectly extracted. Consequently, McLaughlin's argument that tensor products are merely an implementation of a classical symbolic system (McLaughlin, 1997) does not carry over to semantic pointers.

We suspect that, in fact, the poor scaling of the past approaches is a direct result of their use of classical representations. For instance, it is precisely because tensor products include complete representations of every item in the structure being represented that they scale poorly with the depth of the structure. On the other hand, because semantic pointers are created from their constituent vectors through lossy compression, the dimensionality

of the representation remains constant as the depth of the encoded structure increases. This permits deep structures to be efficiently encoded, as we saw in the sentence extraction experiment. We are able to correct for the information lost through compression in a scalable manner using an associative memory.

One benefit of this kind of compressed representation is that models employing them have natural limits on the depth of structure they can encode. Thus, there is no need to appeal to a competence/performance distinction when theory and data differ. It is expected, rather, that the performance of theoretical models will reflect the actual observed performance of human subjects. For example, we have used these representations to capture human error rates as a function of list length in serial working memory tasks (Eliasmith et al., 2012). However, much work remains to be done to demonstrate that model and human performance will match across a wide variety of tasks.

9.3. Psychological plausibility

We do not believe that the model presented here is able to support significant claims of psychological plausibility, other than the very general observation that our method can be used to model lexical processing at a psychological scale. In short, this work is best interpreted as a proof of principle, demonstrating that very large structured representations can be efficiently encoded in a realistic neural network using our method. It is tempting to make more specific psychological claims. However, our choice of WordNet as a lexical structure makes such claims implausible. WordNet was chosen because it is a readily available human-scale, structured representation that is intended as a lexical database of the English language, bearing a resemblance to the human conceptual system.

However, we remain uncommitted to WordNet from a psychological perspective because of its significant limitations. For instance, human conceptual systems likely employ relation-types that do not appear in WordNet, may contain concepts that WordNet omits, and potentially has high-degree concepts in WordNet broken down into intermediate concepts. In addition, there are many concepts in WordNet that are unlikely to be in an average person's conceptual system, either because they are domain-specific (e.g., Gram's Method, a staining technique used to classify bacteria) or culture-specific (e.g., *eisteddfod*, any of several annual Welsh festivals involving artistic competitions). These limitations force the model to perform some of the tasks in ways that may not be psychologically plausible. For instance, in performing a trial in the Hierarchical Extraction experiment where the model has to decide whether *dog* has the type *mammal* (i.e., whether *mammal* can be reached from *dog* via the *class* relation-type), the model must traverse *canine*, *carnivore*, and *placental mammal* along the way. It is unlikely that this same traversal would occur for most human subjects.

In sum, we believe that the fact that our model is able to encode the WordNet semantic network in a reasonable number of neurons lends support to our technique, but we are not convinced that the specific lexical structure proposed by WordNet is psychologically plausible.

9.4. Extensions and future work

We have already mentioned a number of possible extensions to the present work. For instance, it is natural to embed our model within a control algorithm, such as that used in the recent Spaun model (Eliasmith et al., 2012). We could then make use of our network for cognitive tasks and bring it into better contact with behavioral data. Additionally, we have acknowledged that it will be crucial to investigate how this neural representation might be learned from training data while retaining its desirable scaling properties.

There are a number of avenues for improvement beyond these two. For instance, while we have considered only lexical encoding here, semantic pointers are flexible enough to allow many types of information to be encoded simultaneously (Eliasmith, 2013). To construct a representation more reminiscent of a full-fledged concept, we could add perceptual, motor, or dynamics information to each semantic pointer. For the visual modality, this could be accomplished by adding to each semantic pointer a term of the form **vision**⊛**visualData**, where **visualData** is visual information about the concept, and **vision** is a marker that is analogous to the relation-type vectors we have used throughout this study. **visualData** could then be extracted from the semantic pointer by a modified version of our model, with **vision** as the query vector. The model would have to be modified to use an associative memory storing visual information instead of the lexical information we have used in the present study. One could imagine a number of instances of our model in different cortical areas, each with an associative memory storing the type of information relevant for that brain area. Further modality-specific processing could then be performed on the extracted information. A similar approach for a small-scale vocabulary has been pursued in the Spaun model (Eliasmith et al., 2012).

10. Conclusion

We have provided empirical results demonstrating what we believe to be the first implementation of a human-scale structured lexicon in a biologically plausible spiking neural network. We have argued that this significant improvement in scaling over previously available approaches is a result of employing the representational resources provided by semantic pointers. We hope that by providing a specific, large-scale, functioning model we will encourage theoretical disagreements about structured representation to be replaced by implementations that can be quantitatively compared. In short, we believe that it will advance the field to expect that proposals regarding neural implementation of symbolic processing be implemented at scale.

Acknowledgments

Funding for this work was provided by the Air Force Office of Scientific Research (FA8655-13-1-3084), National Science and Engineering Research Council of Canada,

Canada Research Chairs, the Office of Naval Research (N000141310419), the Canadian Foundation for Innovation and the Ontario Innovation Trust.

Notes

1. Briefly, the calculation is as follows. Assume 1,500 nouns and 500 verbs. The number of nodes needed to represent arbitrary structures of the form relation(noun, noun) is $500\times\ 1{,}500\ \times\ 1{,}500 = 1.1\ \times 10^9$ nodes. Assuming 100 neurons per node, which provides a signal-to-noise ratio of 10:1 (Eliasmith & Anderson, 2003), results in 1.1×10^{11} neurons. There are about 20×10^6 neurons per cm^2 (Pakkenberg & Gundersen, 1997), and 2,500 cm^2 of cortex (Peters & Jones, 1984) giving about 50×10^9 neurons in cortex, less than that required for the assumed representation.
2. Following the values used in ff1, the calculation is as follows. van der Velde and de Kamps (2006) note that each connection between symbol and word assemblies requires 8 neural groups, and that 100 assemblies per role should be sufficient. Assuming only two grammatical roles (to be conservative) results in $60{,}000\times200\times8 = 96\times10^6$ groups needed. This suggests 96×10^8 neurons are needed, which works out to about 480 cm^2 of cortex.
3. Briefly, the calculation is as follows. Conservatively assume that only eight dimensions are needed to distinguish the lowest-level concepts (e.g., mammal). Then the representation of Eve requires $8\times8\times8 = 512$-dimensional vectors (i.e., Eve = isA⊗person+. . . = isA⊗isA⊗mammal+. . .). Assuming at least one concept at each level of the sentence requires such a representation means that $512 \times 512 \times 512 = 12.5 \times 10^7$ dimensions or 12.5×10^9 neurons are required, which works out to 625 cm^2 of cortex. Again, this is only for structure representation—not processing—and is significantly larger than relevant language areas.

References

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Chalmers, D. J. (1990). Why Fodor and Pylyshyn were wrong: The simplest refutation. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (pp. 340–347). Cambridge, MA: Erlbaum.

Conklin, J., & Eliasmith, C. (2005). An attractor network model of path integration in the rat. *Journal of Computational Neuroscience*, *18*, 183–203.

Doumas, L. A. A., Hummel, J. E., & Sandhofer, C. M. (2008). A theory of the discovery and predication of relational concepts. *Psychological Review*, *115*, 1–43.

Dronkers, N., Pinker, S., & Damasio, A. (2000). Language and the aphasias. In E. Kandel, J. Schwartz, & T. Jessell (Eds.), *Principles in neural science* (4th ed., pp. 1169–1187). New York: McGraw-Hill.

Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition*. New York: Oxford University Press.

Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: Computation, representation and dynamics in neurobiological systems*. Cambridge, MA: MIT Press.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science*, *338*(6111), 1202–1205.

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, *7*(2–3), 195–225.

Fellbaum, C. (1998). *WordNet: An electronic lexical database*. Cambridge, MA: MIT Press.

Fischer, B. J. (2005). A model of the computations leading to a representation of auditory space in the midbrain of the barn owl. Phd, Washington University in St. Louis.

Fischer, B. J., Peña, J. L., & Konishi, M. (2007). Emergence of multiplicative auditory responses in the midbrain of the barn owl. *Journal of Neurophysiology*, *98*(3), 1181–93.

Fodor, J., & Pylyshyn, Z. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, *28*, 3–71.

Gayler, R. W. (2003). Vector Symbolic Architectures answer Jackendoff's challenges for cognitive neuroscience. In P. Slezak (Ed.), *ICCS/ASCS international conference on cognitive science* (pp. 133–138). Sydney, Australia: University of New South Wales.

Georgopoulos, A. P., Lurito, J. T., Petrides, M., Schwartz, A., & Massey, J. (1989). Mental rotation of the neuronal population vector. *Science*, *243*, 234–236.

Glaser, W. R. (1992). Picture naming. *Cognition*, *42*, 61–105.

Hadley, R. F. (2009). The problem of rapid variable creation. *Neural Computation*, *21*(2), 510–32.

Hinton, G. (2010). Where do features come from? In W. Bechtel (Ed.), *Outstanding questions in cognitive science: A symposium honoring 10 years of the David E. Rumelhart prize in cognitive science* (pp. 7–8). Portland, OR: Cognitive Science Society.

Hummel, J. E., & Holyoak, K. J. (2003). A symbolic-connectionist theory of relational inference and generalization. *Psychological Review*, *110*(2), 220–264.

Jackendoff, R. (2002). *Foundations of language: Brain, meaning, grammar, evolution*. New York: Oxford University Press.

Jonas, P., Major, G., & Sakmann, B. (1993). Quantal components of unitary EPSCs at the mossy fibre synapse on CA3 pyramidal cells of rat hippocampus. *The Journal of Physiology*, *472*, 615–663.

Jones, M. N., & Mewhort, D. J. K. (2007). Representing word meaning and order information in a composite holographic lexicon. *Psychological Review*, *114*(1), 1–37.

Kanerva, P. (1994). The spatter code for encoding concepts at many levels. In M. Marinaro & P. G. Morasso (Ed.), *Proceedings of the International Conference on Artificial Neural Networks* (pp. 226–229). Sorrento, Italy: Springer-Verlag.

Kriete, T., Noelle, D. C., Cohen, J. D., & O'Reilly, R. C. (2013). Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. *Proceedings of the National Academy of Sciences of the United States of America*, *110*(41), 16390–16395.

Kuo, D., & Eliasmith, C. (2005). Integrating behavioral and neural data in a model of zebrafish network interaction [article]. *Biological Cybernetics*, *93*(3), 178–187.

Laubach, M., Caetano, M. S., Liu, B., Smith, N. J., Narayanan, N. S., & Eliasmith, C. (2010). Neural circuits for persistent activity in medial prefrontal cortex. In *Neuroscience 2010 Abstracts* (p. 200.18). San Diego, CA: Society for Neuroscience.

Lenat, D. B. (1995). CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, *38*(11).

Litt, A., Eliasmith, C., & Thagard, P. (2008). Neural affective decision theory: Choices, brains, and emotions. *Cognitive Systems Research*, *9*, 252–273.

Liu, B., Caetano, M., Narayanan, N., Eliasmith, C., & Laubach, M. (2011). A neuronal mechanism for linking actions to outcomes in the medial prefrontal cortex. Presented at *Computational and Systems Neuroscience 2011*. Salt Lake City, UT: Nature Precedings.

von der Malsburg, C. (1981). *The correlation theory of brain function*. *Internal Report 81-2*. Department of Neurobiology, Max Plank Institute for Biophysical Chemistry.

McLaughlin, B. (1997). Classical constituents in Smolensky's ICS architecture. In M. Chiara, K. Doets, D. Mundici, & J. Van Benthem (Eds.), *Structures and norms in science* (Vol. *2*, pp. 331–343). Florence, Italy: Springer.

Miller, G., Beckwith, R., Fellbaum, C., Gross, G., & Miller, K. (1990). Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, *3*, 235–244.

Murdock, B. B. (1993). Todam2: A model for the storage and retrieval of item, associative and serial-order information. *Psychological Review*, *100*(2), 183–203.

O'Reilly, R. C., & Munakata, Y. (2000). *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain* (1st ed.). Cambridge, MA: MIT Press

Ojemann, G., Ojemann, J., Lettich, E., & Berger, M. (1989). Cortical language localization in left, dominant hemisphere. An electrical stimulation mapping investigation in 117 patients. *Journal of Neurosurgery*, *71*(3), 316–326.

Paivio, A. (1986). *Mental representations: A dual coding approach*. New York: Oxford University Press.

Pakkenberg, B., & Gundersen, H. J. R. G. (1997). Neocortical neuron number in humans: Effect of sex and age. *The Journal of Comparative Neurology*, *384*(2), 312–320.

Peters, A., & Jones, E. G. (1984). *Cerebral cortex* (Vol. *1*). New York: Plenum Press.

Plate, T. A. (1995). Holographic reduced representations. *Neural Networks, IEEE Transactions on*, *6*(3), 623–641.

Plate, T. A. (2003). *Holographic reduced representations*. Stanford, CA: CSLI Publication.

Pollack, J. (1990). Recursive distributed representations. *Artificial Intelligence*, *46*(1-2), 77–105.

Pylyshyn, Z. (1984). *Computation and cognition: Toward a foundation for cognitive science*. Cambridge, MA: MIT Press.

Schröder, T., Stewart, T. C., & Thagard, P. (2014). Intention, emotion, and action: A neural theory based on semantic pointers. *Cognitive Science*, *38*(5), 851–880.

Shastri, L., & Ajjanagadde, V. (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings. *Behavioral and Brain Sciences*, *16*, 417–494.

Simmons, W. K., Hamann, S. B., Harenski, C. L., Hu, X. P., & Barsalou, L. W. (2008). fMRI evidence for word association and situated simulation in conceptual processing. *Journal of Physiology*, *102*, 106–119.

Singh, R., & Eliasmith, C. (2006). Higher-dimensional neurons explain the tuning and dynamics of working memory cells. *Journal of Neuroscience*, *26*, 3667–3678.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, *46*, 159–217.

Solomon, K. O., & Barsalou, L. W. (2004). Perceptual simulation in property verification. *Memory and Cognition*, *32*, 244–259.

Stewart, T. C., & Eliasmith, C. (2012). Compositionality and biologically plausible models. In W. Hinzen, M. Werning, & E. Machery (Eds.), *Oxford handbook of compositionality* (pp. 596–615). New York: Oxford University Press.

Stewart, T. C., Choo, X., & Eliasmith, C. (2010). Dynamic behaviour of a spiking model of action selection in the basal ganglia. In D. D. Salvucci & G. Gunzelmann (Eds.), *10th International Conference on Cognitive Modeling* (pp. 235–240). Philadelphia, PA: Drexel University.

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

Stewart, T. C., Bekolay, T., & Eliasmith, C. (2011). Neural representations of compositional structures: Representing and manipulating vector spaces with spiking neurons. *Connection Science*, *3*(2), 145–153.

Stewart, T. C., Bekolay, T., & Eliasmith, C. (2012). Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Decision Neuroscience*, *6*.

van der Velde, F., & de Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, *29*(29), 37–108.

Voelker, A., Crawford, E., & Eliasmith, C. (2014). Learning large-scale heteroassociative memories in spiking neurons. Poster presented at *13th International Conference on Unconventional Computation and Natural Computation*, London, Canada. Retrieved from http://compneuro.uwaterloo.ca/files/publications/voelker.2014b.pdf.

Appendix

Model Details

Finding decoding vectors

To find decoding vectors that decode a function f from the activity of neural population (denoted $\mathbf{d}_i^f$ where i indexes the neurons in the population), we said we had to minimize the expression:

$$\frac{1}{2}\int (f(\mathbf{x}) - \sum_i a_i(\mathbf{x})\mathbf{d}_i^f)^2 \mathrm{d}\mathbf{x}. \tag{21}$$

We minimize this numerically, using a finite number of evaluation points (values of $\mathbf{x}$) in some region of the represented space that we want our decoding vectors to perform well on. Let L denote the number of evaluation points, let M denote the dimensionality of the range of the function f, and let N denote the number of neurons in our population. We now define matrices that will aid us in the optimization. Let $\mathbf{D}$ denote the $N \times M$ matrix whose rows are the decoding vectors. Let $\mathbf{A}$ denote the $L \times N$ matrix whose rows are the activities of the neurons at a given evaluation point. Let $f(\mathbf{X})$ denote the $L \times M$ matrix whose rows are the values of the function f at different evaluation points. The jth row of $\mathbf{AD}$ is equal to the transpose of $\widehat{f(\mathbf{x}_j)} = \sum_i a_i(\mathbf{x}_j)\mathbf{d}_i^f$ where $\mathbf{x}_j$ is the jth evaluation point. Minimizing Eq. (21) is then equivalent to solving for $\mathbf{D}$ in the following equation:

$$\begin{aligned} f(\mathbf{X}) &= \mathbf{AD} \\ \mathbf{A}^{\mathrm{T}} f(\mathbf{X}) &= \mathbf{A}^{\mathrm{T}}\mathbf{AD} \\ (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}} f(\mathbf{X}) &= \mathbf{D} \end{aligned} \tag{22}$$

Since some neurons in the population are likely to have similar tuning curves, the matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is unlikely to be invertible. Thus, we typically take the *Moore-Penrose pseudoinverse* of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ using Singular Value Decomposition (SVD), which is guaranteed to provide the least-squares optimal solution to Eq. (22).

Sub-populations

This procedure for solving for the decoding vectors can be computationally intractable. Consider that in our neural model, one of the populations contains 51,400 neurons. The matrix $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ for that population would have dimensions 51,400 $\times$ 51,400. Taking the SVD of a matrix this large is not feasible.

Instead, we can consider these populations to be made up of many sub-populations, each of which represents a small subset of the dimensions of the overall population's represented space. The representational properties of the collection of sub-populations is very

similar to that of a single large population. However, the computational properties are different (Eliasmith & Anderson, 2003). The light gray populations from the Fig. 4 are split into one-dimensional populations, whereas the dark gray population is split into two-dimensional populations, each representing one dimension from each of the two Fourier transformed input vectors and computing their product (i.e., the ability to performed the required element-wise multiplication is preserved).

This implementation allows for much more efficient computation of decoding vectors. For example, each of the light gray populations, which represent 512-dimensional vectors, is taken to be composed of 512 one-dimensional sub-populations of 50 neurons each instead of a single population of 25,600 neurons. As a result, SVD on a 25,600 × 25,600 matrix is replaced by 512 SVD's on 50 × 50 matrices, which is computationally tractable. The only consequence of this is sparsification of the connection weight matrices; the same number of neurons are used in both cases.

Single neuron model and parameters

All neurons are modeled as point-processes and employ the LIF neuron model. The sub-threshold behavior of a LIF neuron is governed by the differential equation:

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \frac{-1}{\tau_{RC}}(V - J(\mathbf{e}^{\mathrm{T}}\mathbf{x})). \tag{23}$$

The parameter τ_{RC} is a time constant governing the sub-threshold dynamics of the neuron. When the voltage V exceeds a threshold of 1.0, a spike is emitted from the neuron, and a refractory period begins during which the voltage is fixed at 0. The length of the refractory period is given by a constant τ_{ref}. $J(\mathbf{e}^{\mathrm{T}}\mathbf{x})$ is the input current of the neuron and is given by $J(\mathbf{e}^{\mathrm{T}}\mathbf{x}) = \alpha\mathbf{e}^{\mathrm{T}}\mathbf{x} + J^{\mathrm{bias}}$. The quantity $\mathbf{e}^{\mathrm{T}}\mathbf{x}$ is the dot product between the neuron's encoding vector $\mathbf{e}$ and the input vector $\mathbf{x}$. The parameters α and J^{bias} are uniquely determined by the neuron's radius, maximum firing rate, firing threshold, τ_{ref} and τ_{RC}. The radius specifies a value of $\mathbf{e}^{\mathrm{T}}\mathbf{x}$ for which the neuron fires at its maximum firing rate. Past this, the neuron is largely saturated, and changes to the input will not be reflected in the firing rate. Finally, the firing threshold specifies a lower bound on values of $\mathbf{e}^{\mathrm{T}}\mathbf{x}$ for which the neuron fires. Maximum firing rates and firing thresholds are chosen randomly for each neuron. Numerical values for the parameters used in the neural model, as well as distributions for the values that are chosen randomly, are presented in Table A1.

For the synapse model, we take each spike from a pre-synaptic neuron to evoke a post-synaptic current in the dendrites of all downstream neurons. The equation governing this current is:

$$h_{\mathrm{PSC}}(t) = e^{-t/\tau_{\mathrm{PSC}}}. \tag{24}$$

These post-synaptic currents build up additively over time. The post-synaptic time constant, τ_{PSC}, controls the decay time of the waveform; smaller values cause it to decay

Table A1.
Parameters used in the neural model

Parameter	Association Neurons	Standard Neurons
τ_{RC}	34 ms	20 ms
τ_{ref}	2.6 ms	2 ms
τ_{PSC}	5 ms	5 ms
Radius	1.0	$5/\sqrt{512}$
Max Firing Rate	Uniform(200, 350) spike/s	Uniform(200, 400) spike/s
Firing Threshold	0.3	Uniform($\frac{-5}{\sqrt{512}}, \frac{5}{\sqrt{512}}$)

faster. All connections between neurons are assumed to be mediated by AMPA neurotransmitters, so all post-synaptic time constants are set at 5 ms (Jonas, Major, & Sakmann, 1993). We can now write an explicit expression for the neural activity of the LIF neuron. Let $t_1, t_2, \ldots t_n$ be the times that spikes are generated by the sub-threshold dynamics of Eq. (23). Then, since a post-synaptic current is generated each time a spike is emitted, the activity of the neuron at time t, as seen by its downstream neurons, is:

$$a(t) = \sum_{i=1}^{n} h_{\text{PSC}}(t - t_i) = \sum_{i=1}^{n} e^{-(t-t_i)/\tau_{\text{PSC}}}$$

Examples of population tuning curves for both associative and standard (i.e., not in the associative memory) neural sub-populations are shown in Fig. A1. The effects of several of the parameters can be observed. In particular, the association neurons have firing thresholds above 0, in contrast to the standard neurons. This helps the associative populations implement their thresholding behavior. Also visible is the larger radius for the associative neurons. This is a consequence of the fact that inputs to the associative

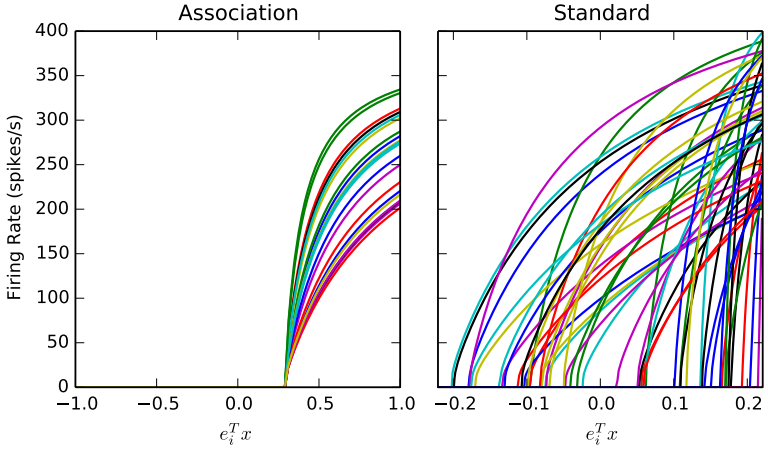

Fig. A1. Firing-rate tuning curves from different sub-population types.

populations are dot products which fall roughly within (–1.0, 1.0). On the other hand, the radius for the standard neurons does not have to be as large, because the inputs to the standard sub-populations are generally single elements of 512-dimensional vectors with norm around 1, which are highly unlikely to be far from 0.