From Vectors to Symbols to Cognition: The Symbolic and Sub-Symbolic Aspects of Vector-Symbolic Cognitive Models

Matthew A. Kelly (mkelly11@connect.carleton.ca)
Robert L. West (robert_west@carleton.ca)
Institute of Cognitive Science, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6 Canada

Abstract

To achieve a full, theoretical understanding of a cognitive process, explanations of the process need to be provided at both symbolic (i.e., representational) and sub-symbolic levels of description. We argue that cognitive models implemented in vector-symbolic architectures (VSAs) intrinsically operate at both of levels and thus provide a needed bridge. We characterize the sub-symbolic level of VSAs in terms of a small set of linear algebra operations. We characterize the symbolic level of VSAs in terms of cognitive processes, in particular how information is represented, stored, and retrieved, and classify vector-symbolic cognitive models in the literature according to their implementation of these processes. On the basis of our analysis, we speculate on avenues for future research, and suggest means for theoretical unification of existent models.

Keywords: Vector symbolic architectures; Holographic reduced representations; cognitive modelling; symbolic modelling; sub-symbolic modelling.

Introduction

To achieve a full, theoretical understanding of a cognitive process and how it relates to the physical world, explanations of the process need to be provided at both symbolic (i.e., representational) and sub-symbolic levels of description. The classic symbolic approaches to modelling do not account for how the symbol manipulations described in the model could arise from neural tissue, or account for how the symbols themselves come into existence. Classic connectionist approaches are more concerned with neural plausibility, but are notoriously opaque, doing little to aid our understanding of the cognitive processes modelled. By contrast, the vector-symbolic approach to modelling explicitly provides an account at both levels of description.

Vector Symbolic Architectures (VSAs), a term coined by Gayler (2003; but see also Plate, 1995), are a set of techniques for instantiating and manipulating symbolic structures in distributed representations. VSAs have been used to successfully model a number of different cognitive processes (e.g., analogical mapping in Eliasmith & Thagard, 2001; letter position coding in Hannagan, Dupoux, & Christophe, 2011; semantic memory in Jones & Mewhort, 2007). It has been argued that VSAs provide a bridge between conventional symbolic modelling and both connectionist modelling (Rutledge-Taylor & West, 2008) and more realistic models of neural processing (Eliasmith, 2007). However, if we are to take the bridging metaphor seriously, it is important to clarify which parts of a VSA are symbolic in nature and which are sub-symbolic. We will attempt to lay out a simple system for understanding VSAs in terms of basic operations and symbolic/sub-symbolic decisions, and thereby provide a comprehensive and comprehensible introduction to VSAs for newcomers, and a common frame of reference for those already using VSAs. By providing a high-level overview that integrates the techniques of existent VSA-based cognitive models into a coherent picture we hope to highlight as yet unexplored avenues of research and sketch what a VSA-based account of cognition as a whole would look like.

In this analysis, the vectors represent symbolic information. These vectors, or symbols, can be combined and manipulated using a small number of operations, which can be understood as sub-symbolic processes. However, the information processing models built from these operations are themselves, best characterized at a symbolic level of description. Importantly, the modelling decisions made at the sub-symbolic level are to some degree independent of the modelling decisions made at the symbolic level. This paper is divided into two parts to reflect these two levels of description, symbolic and sub-symbolic.

The Sub-Symbolic Level

VSAs are closely related to the better-known tensor product representations (Smolensky, 1990), but unlike tensor product representations, VSAs can compactly represent symbolic expressions of arbitrary complexity. A number of VSA techniques exist in the literature, including Holographic Reduced Representations (HRRs; Plate, 1995), frequency-domain HRRs (Plate, 1994), some earlier forms of holographic associative memory (Eich, 1982; Murdock, 1982), as well as binary spatter codes (Kanerva, 1992), Multiply-Add-Permute coding (Gayler, 2003), and square matrix representations (Kelly, 2010).

Each VSA technique uses the same set of basic operations, but implements the operations differently. Thus the choice of a particular VSA dictates how symbols are instantiated and manipulated and defines the model at the sub-symbolic level. To ground the discussion, we mainly discuss Holographic Reduced Representations (HRRs) (Plate, 1995), as HRRs are the most widely used VSAs in the cognitive modelling literature. Also, HRRs are used as the basis for the Neural Engineering Framework (NEF; Eliasmith, 2007) and thus demonstrably have a clearly defined and plausible neural implementation. However, the other VSA techniques are similar and most anything that can be done with an HRR can be done with any VSA technique.

n-Space and Similarity

In a VSA, a symbol, or representation, is an *n*-dimensional vector: a list of *n* numbers that defines the coordinates of a point in an *n*-dimensional space. VSAs work best for values of *n* in the hundreds or thousands (Plate, 1995).

A vector can be understood as a line drawn from the origin (the zero coordinates) to the coordinates specified by the vector. The length of the line is the vector's magnitude. The direction of the vector encodes the meaning of the representation. Similarity in meaning can thus be measured by the size of the angles between vectors. This is typically quantified as the cosine of the angle between vectors. The cosine of vectors **a** and **b** can be calculated as:

$$\textit{cosine}(\mathbf{a}, \mathbf{b}) = (\mathbf{a} \bullet \mathbf{b}) \,/\, (\,(\mathbf{a} \bullet \mathbf{a})^{0.5}\, (\mathbf{b} \bullet \mathbf{b})^{0.5}\,)$$

where • is the dot product. A cosine of 1 means the vectors are identical, -1 means they are opposites, and 0 means they are completely dissimilar. If each vector has a magnitude of one, the cosine is just the dot-product of the vectors. Thus, some systems rescale all vectors to a magnitude of one after vector operations. In memory systems where new memories are superimposed on old memories, such re-scaling causes a recency effect and rapid forgetting because new memories will make-up a fixed fraction of memory, regardless of the quantity of previous experience.

While the cosine measures the *angle* between two vectors, the cosine is often described as a measure of *distance*. As it is more intuitive to describe similarity as a measure of distance than as a measure of the angle, for convenience, we can imagine the vectors as describing points on a *hypersphere*, such that the size of the angles are the distances between them.

Atomic versus Complex representations

Representations in a VSA are either atomic or complex. An atomic representation is a unique representation, a symbol that cannot be broken down into sub-symbols. In an HRR, values for an atomic representation are typically generated by random sampling from a standard normal distribution. By assigning random values to the vectors, atomic representations will be uniformly distributed across the surface of the hypersphere, such that the atomic representations will have little to no similarity to each other.

Complex representations can be created by either combining atomic representations or recursively combining complex representations. Critically, in a VSA, a complex representation has the same dimensionality as an atomic representation, allowing representations both atomic and complex to be compared, or combined together to create representations of arbitrary complexity. VSAs have two operators for combining representations: superposition and binding. In HRRs, superposition is vector addition and binding is circular convolution. We denote vector addition by +, and circular convolution by *. Binding and superposition, along with random permutation, are the basic operations used to create complex representations in VSAs.

Superposition (+) versus Binding (*)

The key difference between superposition and binding is their effect on similarity. Superposition is similarity-preserving: the sum of two vectors is a vector that falls in the angle between them. Conversely, binding is similarity destroying: the circular convolution of two vectors is roughly orthogonal to the two original vectors. The purpose of superposition is to combine representations to create a new representation that is similar to all of the combined representations. The purpose of binding, on the other hand, is to create "chunks": unique identifiers for combinations of representations.

Most VSA use a form of vector addition for superposition. Vector addition is computed by adding together the corresponding elements of the two vectors. So, for example, {1,4,7} + {5,4,2} = {6,8,9}.

To bind, HRRs use circular convolution, * , which can be computed rapidly using element-wise multiplication, ∘ , and the fast Fourier transform, *fft*, and its inverse, $\textit{fft}^{-1}$:

$$\mathbf{a} * \mathbf{b} = \textit{fft}^{-1}(\,\textit{fft}(\mathbf{a}) \circ \textit{fft}(\mathbf{b})\,)$$

Essentially circular convolution is a lossy way of scrambling the information of the two vectors together to produce a new vector of the same dimensionally.

Consider the problem of learning the meaning of the phrase "kick the bucket", a colloquial euphemism for death. Suppose the cognitive model has a vector representation of the concept *kick* and a vector representation of the concept *bucket*. The sum (superposition) of those two vectors will produce a vector that is close to both *kick* and *bucket*, indicating that the phrase “kick the bucket" has a meaning similar to *kick* and to *bucket*. But in order for the cognitive model to be able to learn that the phrase "kick the bucket" has a distinct meaning that is not a function of its parts, the model needs to be able to assign to "kick the bucket" a distinct identifier. Binding is the operation that performs this function in VSA-based models. The vector *kick * bucket* is dissimilar to the vectors *kick*, *bucket*, and *kick + bucket*.

Binding and superposition can also be used jointly to address the binding problem (Gayler, 2003), that is, the question of how to couple sets of attributes together such that the attributes of one object are not confused with the attributes of another. For example, given a *small red square* and a *large blue circle*, the complex representation (**small** * **red** * **square**) + (**large** * **blue** * **circle**) creates a single vector that distinctively represents the knowledge that the *square* is small and red and the *circle* is large and blue.

Unbinding

Unbinding is an inverse of binding that allows vectors that have been bound together in a complex representation to be unpacked and recovered. Circular correlation, #, is the unbinding operator for HRRs. Given a pair of vectors bound together, and one of the pair, referred to as the probe, unbinding produces an approximation of the other vector, referred to as the target, i.e.,

$$\mathbf{p} \,\#\, (\mathbf{p} * \mathbf{t}) = \mathbf{a} \approx \mathbf{t}$$

where **p** is the probe, **t** is the target, and **a** is an approximation of the target.

Unbinding can be understood as binding with the inverse of the probe. The *inverse* of any vector **x** is a re-ordering of the elements of **x**, i.e. a permutation of **x**, such that,

$$\mathbf{x} \# \mathbf{x} = \mathbf{x} * \mathit{inverse}(\mathbf{x}) \approx \boldsymbol{\delta}$$

where $\boldsymbol{\delta}$ is the identity vector for binding, i.e. for any vector **x**, $\mathbf{x} * \boldsymbol{\delta} = \mathbf{x}$. Thus binding with the inverse of a vector unbinds what that vector has been associated with:

$$\mathbf{a} \# (\mathbf{a} * \mathbf{b}) = \mathit{inverse}(\mathbf{a}) * (\mathbf{a} * \mathbf{b}) \approx \boldsymbol{\delta} * \mathbf{b} = \mathbf{b}$$

In HRRs, the inverse of any vector $\mathbf{x} = \{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ is:

$$\mathit{inverse}(\mathbf{x}) = \{\mathbf{x}_1, \mathbf{x}_n, \mathbf{x}_{n-1}, \ldots \mathbf{x}_3, \mathbf{x}_2\}$$

Circular convolution, *, is commutative, i.e., the order of binding does not matter when using circular convolution. Given vectors **a** and **b**, their association $\mathbf{a} * \mathbf{b} = \mathbf{b} * \mathbf{a}$, and likewise, when unbinding, $\mathbf{b} \# (\mathbf{a} * \mathbf{b}) = \mathbf{b} \# (\mathbf{b} * \mathbf{a}) \approx \mathbf{a}$.

Permutation

Gayler (2003) describes random permutation as an operation used "to quote or protect the vectors from the other operations". Permutation of the numbers 1 ... *n* defines a unary function that can transform a vector. A randomly chosen permutation of a vector is unlikely to be similar to the original vector, but the permutation is also reversible. Given *p*, there is a permutation p^{-1} such that, $p^{-1}(p(\mathbf{a})) = \mathbf{a}$. When permuted, the information within a vector is essentially hidden and protected from being affected by other vector operations.

For example, as noted above, circular convolution is commutative, that is, $\mathbf{a}*\mathbf{b} = \mathbf{b}*\mathbf{a}$. This property of circular convolution can be useful, but it can be a hindrance in situations where the order of items matter, e.g. "dog feed" and "feed dog" are phrases which carry different meanings by virtue of differences in word order.

A non-commutative variant of circular convolution can be defined using a random permutation *p* and its inverse p^{-1}. By always randomly permuting one of the arguments before convolution, one defines a binding operation that is non-commutative, i.e. while $\mathbf{a}*\mathbf{b} = \mathbf{b}*\mathbf{a}$, $p(\mathbf{a})*\mathbf{b} \neq p(\mathbf{b})*\mathbf{a}$. Unbinding then uses the inverse permutation p^{-1}, e.g.

$$\mathit{cosine}(\, p^{-1}(\mathbf{a} \# (\mathbf{a} * p(\mathbf{b}))), \mathbf{b}) \approx 0.71$$
$$\mathit{cosine}(\, p^{-1}(\mathbf{b} \# (\mathbf{a} * p(\mathbf{b}))), \mathbf{a}) \approx 0$$

Non-commutative binding is used by the BEAGLE model (Jones & Mewhort, 2007) to bind vectors that stand for words in sentences in order to construct representations of the semantics of each of those words. For a variety of other uses of random permutation in VSAs, see Gayler (2003), Sahlgren, Holst, and Kanerva (2008), and Kelly (2010).

The Symbolic Level

When making a vector-symbolic model, decisions need to be made at both the symbolic and sub-symbolic levels. At the sub-symbolic level, the modeller needs to decide how to instantiate symbols as vectors and symbol-manipulation as vector algebra. Conversely, at the symbolic level, the modeller needs to make decisions about how to structure, manage, store, and retrieve those symbols. Choosing to use HRRs rather than another kind of VSA can define the sub-symbolic level, but this choice is largely independent of the decisions to be made at the symbolic level.

In fact, we have already seen two examples of manipulations at the symbolic level. The first was combining binding and addition to create a vector that encodes information about bound entities (e.g., *small red square* and *large blue circle*). The second was combining permutation and binding to create a bound entity that maintained information about order. Essentially, all VSA systems work in the same way. Vectors encode the desired information according to some sort of scheme (i.e., by combining the operations discussed above), and then, when needed, the information is retrieved from the vectors.

Encoding and Storage

BEAGLE (Jones & Mewhort, 2007) and DSHM (Rutledge-Taylor & West, 2008) use the terms *environmental vectors* and *memory vectors*. We extend the use of this terminology to other vector-symbolic models. An *environmental* vector is a vector that stands for atomic perceptions from the environment (e.g., a *red circle* needs two environmental vectors, one for *circle* and one for *red*). Environmental vectors are fixed and do not change. A *memory* vector is a complex representation stored by the model and used to produce behaviour. In some systems, memory vectors change with experience. Additionally, we use the term *experience vector* to refer to a representation that stands for the model's current experience of its environment created by combining environmental vectors (e.g., an experience vector could represent the perception of a *red circle* by convolving the environmental vectors of *circle* and *red*).

By examining the relationship between environmental, experience, and memory vectors across vector-symbolic models, we distinguish between three main approaches to storage. In a *many-to-one* vector model, all experience vectors are summed into a single memory vector for storage. In a *one-to-one* vector model, each experience vector is stored as a separate memory vector among an ever-growing number of memory vectors. In a *many-to-many* vector model, there are a fixed number of memory vectors, and incoming environmental vectors are used to update them. Each of these approaches has strengths and weaknesses.

Many-to-one In a *many-to-one* vector model, such as TODAM (Murdock, 1983) or CHARM (Eich, 1982), memory is modelled as a single, high-dimensional vector. All experience vectors are added to the memory vector. There is a limit to how much can be stored in the vector before mistakes start to be made. Mistakes are, of course, of interest to psychologists, and the pattern of mistakes made by a *many-to-one* vector model allow it to mimic human forgetting in list-recall tasks. If the goal is to model how people store a small amount of recently learned or closely related information, a single memory vector suffices.

Many-to-one vector models also have the advantage of a clear neural implementation. In the Neural Engineering Framework (NEF; Eliasmith, 2007) binding, unbinding, and and superposition can all be implemented through neural connectivity. In the NEF interpretation, a *many-to-one* memory is a neural group with self-recurrent connections that acts as a working memory or buffer, and many such buffers could exist in the brain.

One-to-one In a *one-to-one* vector model, such as MINERVA (Hintzman, 1986), the Iterative Resonance Model (Mewhort & Johns, 2005), and the Holographic Exemplar Model (Jamieson & Mewhort, 2011), each experience vector is represented as a separate memory. While this approach to modelling memory is both simple and successful, the ever growing number of vectors that need to be stored and accessed by the memory system is both neurally implausible and computationally impractical for modelling tasks in which very large amounts of knowledge are relevant, e.g., semantic priming tasks (Jones & Mewhort, 2007). However, these models are able to reproduce a wide variety of memory effects, providing a unitary account of episodic, semantic, and implicit memory, indicating that, although their warehouse-style management of vectors is implausible, their processes of storage and retrieval provide a good analogue for biological memory.

Many-to-many *Many-to-many* vector models, such as BEAGLE (Jones & Mewhort, 2007) and DSHM (Rutledge-Taylor, 2008), can be understood as a hybrid of the earlier *many-to-one* and *one-to-one* approaches. In *many-to-many* memory, for each item of interest, there is a randomly generated environmental vector and a specially constructed memory vector. In BEAGLE the items of interest are words: the environmental vector stands for the word's orthography or phonology and the memory vector stands for the word's meaning. In DSHM, the items are objects relevant to the experimental task: the environmental vector stands for the percept of the object and the memory vector stands for the concept of the object.

Like the one-to-one models, the management of the vectors in many-to-many systems is computationally expensive and, at this point, neurally implausible. However, the ability to generate memory vectors that stand for particular concepts in very powerful (e.g., Rutledge-Taylor, Vellino, & West, 2008) and allows these systems to capture numerous different phenomena (e.g., Rutledge-Taylor & West, 2008) and represent vast quantities of data (Jones & Mewhort, 2007).

For example, to create an association between *keyboards* and *computers*, each time a computer and keyboard co-occur a copy of the environmental vector for keyboard can be added to the memory vector for computer and a copy of the environmental vector for computer can be added to the memory vector for keyboard. The effect to this would be to move the memory vector for computer closer to the environmental vector for keyboard and move the memory vector for keyboard closer to the environmental vector for computer. Over time, the result of this is to organize the space so that memory vectors are clustered around environmental vectors that they co-occur with so that the distance between the vectors equals strength of association.

Another, more complicated example involves binding and the use of the *placeholder vector*. The placeholder vector is an atomic (i.e, random) vector, but it is used to encode all associations, and thus can be used as a universal retrieval cue. Consider the phrase or stimulus *blue triangle*. Without using the placeholder, we could update memory as follows:

$$\mathbf{memory}_{\textit{blue}} \mathrel{+}= \mathbf{blue} * \mathbf{triangle}$$
$$\mathbf{memory}_{\textit{triangle}} \mathrel{+}= \mathbf{blue} * \mathbf{triangle}$$

By binding together the environmental vectors for **blue** and **triangle** and adding the result to the memory vectors for *blue* and *triangle* (an operation denoted by +=), we move the two memory vectors towards the point in space described by the vector **blue** * **triangle**, and thereby move $\mathbf{memory}_{\textit{blue}}$ and $\mathbf{memory}_{\textit{triangle}}$ closer together. But people almost *never* get the concepts *blue* and *triangle* confused with each other. This is because *blue* is a colour (or an adjective), and *triangle* is a shape (or a noun), i.e. they are different sorts of thing.

Conversely, consider updating using the placeholder:

$$\mathbf{memory}_{\textit{blue}} \mathrel{+}= \mathbf{placeholder} * \mathbf{triangle}$$
$$\mathbf{memory}_{\textit{triangle}} \mathrel{+}= \mathbf{blue} * \mathbf{placeholder}$$

This moves $\mathbf{memory}_{\textit{blue}}$ towards **placeholder** * **triangle**, i.e. towards all properties of triangles, and moves $\mathbf{memory}_{\textit{triangle}}$ towards **blue** * **placeholder**, i.e. towards all things that are blue. Thus, by using a placeholder, the memory vectors for nouns will cluster together in one region of space, and the vectors for adjectives will cluster together in another region of space, and things that are *colours* will cluster separately from things that are *coloured*. This is a subtle distinction but Jones and Mewhort (2007) have shown it to be very important and very powerful.

Retrieval

There are two categories of information retrieval processes used in vector-symbolic models: *unbinding,* which retrieves information from a particular vector, and *resonance,* which allows information to be retrieved from the entire library of vectors in memory. *Many-to-one* models, such as TODAM (Murdock, 1982) only use unbinding. *One-to-one* models that do not use binding to encode associations, such as MINERVA (Hintzman, 1986), only use resonance. In *many-to-many* systems, these two retrieval processes are complementary. For example, resonance can be used to retrieve a vector, which can then be unbound.

Unbinding Consider a simple example where the, agent is given a set of coloured shapes to remember: *blue triangle, green square, red circle.* In a *many-to-one* vector model this could be encoded by binding (*) the vectors for the shapes to the colours, then summing to create a memory vector:

memory = **blue*triangle** + **green*square** + **red*circle**

The colour of any one of these shapes could then be recalled by unbinding (#) using the shape to probe memory:

triangle # **memory** ≈ **blue**

In a *many-to-many* vector model, unbinding may use the *placeholder* as the probe. The placeholder is a special, randomly generated atomic vector that acts as a key to all of memory. The placeholder is initially used in binding:

$\mathbf{memory}_{blue}$ = **placeholder** * **triangle**
$\mathbf{memory}_{triangle}$ = **placeholder** * **blue**

The placeholder can then be used in unbinding:

placeholder # $\mathbf{memory}_{triangle}$ ≈ **blue**

Resonance (*one-to-one*) The term resonance comes from MINERVA (Hintzman, 1986), but it is implemented differently across different models. In MINERVA, the process of resonance begins by measuring the similarity (cosine) of each vector in memory to the probe. Then resonance computes a weighted sum of all vectors in memory. This sum, termed the *echo*, is what the model retrieves from memory. Each vector in the sum is weighted by its similarity to the probe raised to an exponent.

For example, the three shapes might be represented as:

$\mathbf{memory}_1$ = **triangle** + **blue**
$\mathbf{memory}_2$ = **square** + **green**
$\mathbf{memory}_3$ = **circle** + **red**

If the probe is **triangle**, then the echo would approximate:

$\mathbf{echo} \approx 0.5^b\mathbf{memory}_1 + 0.0^b\mathbf{memory}_2 + 0.0^b\mathbf{memory}_3$
$\mathbf{echo} \approx 0.5^b(\mathbf{triangle} + \mathbf{blue})$

such that the memory system would remember that the triangle is blue. The exponent *b* is a small, positive integer that is odd-numbered so as to preserve the sign of the similarity. Note that the similarity values of 0.5 and 0.0 are approximate. Random vectors in a high dimensional space have an expected cosine of 0, but the actual cosine between any two random vectors will be a little more or a little less.

The exponent *b* critically allows *one-to-one* vector models to function even when there is a very large amount of data in memory. If the exponent *b* is 1, the result of resonance roughly imitates decoding in a simple associative memory, such as a Hopfield network. With an exponent greater than one, resonance increases the signal to noise ratio in the echo by increasing the relative weighting of the memory vectors most similar to the probe. If *b* is too low, a large number of partial matches in memory could easily overwhelm an exact match to the probe, resulting in a poor echo. With a high *b*, the echo will essentially just be the most similar vector in memory to the probe. In MINERVA, a *b* of 3 is standardly used, but a *b* of 3 may be too low when modelling a larger sum of knowledge than what is typically necessary to model a psychology experiment (e.g., in modelling word pronunciation, such as in Kwantes & Mewhort, 1999).

In the Iterative Resonance Model (IRM; Mewhort & Johns, 2005), resonance is iterated, and with each iteration *b* is increased until a decision to stop iterating is made, resulting in either successful retrieval or a failure to retrieve. This approach has two benefits: (1) the number of iterations can be used to predict response time in memory tasks, and (2) it eliminates *b* as a tweaking parameter by introducing a theory-driven approach to setting its value.

Resonance (*many-to-many*) Although the term *resonance* is used to describe retrieval in *many-to-many* vector models, the implementation is different and simpler: Essentially, the memory vector most similar to the probe is retrieved. This can be understood as a kind of spreading activation (Rutledge-Taylor & West, 2008). The probe and memory vectors can be understood as points on a hypersphere, such that the cosine measures the distance between them. One can imagine a ripple of activation spreading out from the probe across the surface of the hypersphere. The memory vectors closest to the probe become active in working memory, with the closer vectors becoming active sooner. This model of resonance allows BEAGLE (Jones & Mewhort, 2007) to make semantic priming reaction time predictions (e.g., that *doctor* is recognized faster when preceded by *nurse* than when preceded by an unrelated prime such as *stapler*) and to model the fan-effect in DSHM (Rutledge-Taylor & West, 2008).

Conclusions

We hold that, in order to bridge the gap between human experience and neural connectivity, explanations at both the symbolic and sub-symbolic levels of description are necessary parts of theory in cognitive science. As we illustrate in this paper, cognitive models that use vector-symbolic architectures intrinsically operate at both of these levels of description and thereby provide a needed bridge between the two kinds of explanation.

At the sub-symbolic level is the vector-symbolic architecture itself, and the linear algebra operations on vectors that comprise the architecture: *similarity, superposition, binding*, *unbinding*, *permuting, un-permuting.* All of these operations are easily amenable to neural implementation, as in the NEF (Eliasmith, 2007).

At the symbolic level, we have the cognitive model itself, and the cognitive processes that define it. On the basis of their storage and retrieval mechanisms, we classify existing vector-symbolic cognitive models into *many-to-one*, *one-to-one,* and *many-to-many* vector models. This classification scheme highlights stark differences between these models.

Many-to-many vector models differ from the other two classes of model in two important ways. First, *many-to-many* models use a *placeholder* vector to stand for "this item I am thinking about". The placeholder acts as a symbol with an important functional role but no perceptual or conceptual meaning. It may be useful to incorporate other kinds of function vectors in future models, e.g., a *wildcard* vector to stand for "that item that I'm not thinking about", vectors to stand for emotional states, for truth values, et cetera.

Secondly, in *many-to-many* models, memory vectors are labelled and stand for particular concepts, whereas in *one-to-one* models concepts are an emergent phenomenon produced by the echoes retrieved using resonance (Hintzman, 1986). While the conceptual representations in *many-to-many* models are powerful, having a predefined number of concepts is implausible and limiting.

Many-to-one vector models can be constructed in NEF as self-recurrent neural groups and are understood as working memory buffers. By contrast, *one-to-one* and *many-to-many* models are best understood as models of long-term memory, but as yet lack a neural explanation.

Finding a means of translating *one-to-one* and *many-to-many* vector models into neural models may provide a route to a unified, vector-symbolic account of memory storage and retrieval. As we noted earlier, a *one-to-one* model behaves somewhat like a Hopfield network when the resonance exponent *b* is set to 1. To implement a *one-to-one* model as a network, one needs to find a mechanism analogous to *b* that can act to increase the signal to noise ratio in the echo. We speculate that the vector-symbolic intersection circuit proposed by Levy and Gayler (2009) might provide a start for developing such a mechanism.

We suspect that the memory vectors that stand for concepts in *many-to-many* vector models are, in fact, the echoes in *one-to-one* vector models. That is to say, we agree with Hintzman (1986) that concepts are an emergent property of retrieval. Using a *one-to-one* model to do the kind of large scale modelling in *many-to-many* models is impossible because *one-to-one* models store all experiences without any form of compression. However, a neural implementation of a *one-to-one* model would naturally be lossy in its storage, and so could provide a plausible account of concept formation over a lifetime of experiences.

Unification in other areas, such as representation, is important too. Incorporating a vector-symbolic model of string encoding (Hannagan et al., 2011) into the BEAGLE model of semantics (Jones & Mewhort, 2007) could, for instance, allow BEAGLE to model how shared orthography can help and hinder in understanding the meaning of words.

Eventually, we hope to see developed a vector-symbolic cognitive architecture, which not only presents a unified and neurally plausible approach to representation, storage, and retrieval, but also extends the vector-symbolic account beyond its roots in memory theory, and integrating it into accounts of emotions, attention, perception, and consciousness. As cognitive scientists, it is important to keep in mind our ultimate, lofty, and collective goal of a theory that unifies not only all aspects of the cognition, but all relevant levels of description.

References

Eich, J. M. (1982). A composite holographic associative recall model. *Psychological Review, 89*, 627–661.

Eliasmith, C., & Thagard, P. (2001). Integrating structure and meaning: a distributed model of analogical mapping. *Cognitive Science, 25*, 245-286.

Eliasmith, C. (2007). How to build a brain: From function to implementation. *Synthese, 159*, 373-388.

Gayler, R. (2003). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. *ICCS/ ASCS International Conference on Cognitive Science*.

Hannagan, T., Dupoux, E., & Christophe, A. (2011). Holographic string encoding. *Cognitive Science, 35*, 79-118.

Hintzman, D. L. (1986). "Schema abstraction" in multiple-trace memory models. *Psychological Review, 93*, 441-428.

Jamieson, R. K., & Mewhort, D. J. K. (2011). Grammaticality is inferred from global similarity: A reply to kinder (2010). *The Quarterly Journal of Experimental Psychology, 64*, 209-216.

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

Kelly, M. A. (2010). Advancing the theory and utility of holographic reduced representations. (Master's thesis, School of Computing, Queen's University).

Kanerva, P. (1996). Binary spatter-coding of ordered k-tuples. *Proceedings of the 1996 International Conference on Artificial Neural Networks*, 869-873.

Kwantes, P. J., & Mewhort, D. J. K. (1999). Modeling lexical decision and word naming as a retrieval process. *Canadian Journal of Experimental Psychology, 53*, 306-315.

Levy, S.D., & Gayler, R.W. (2009). A distributed basis for analogical mapping. *New frontiers in analogy research; Proceedings of the Second International Analogy Conference - Analogy 09*, 165-174.

Mewhort, D. J. K., & Johns, E. E. (2005). Sharpening the echo: An iterative-resonance model for short-term recognition memory. *Memory, 13*, 300-307.

Murdock, B. B. (1982). A theory for the storage and retrieval of item and associative information. *Psychological Review, 89*, 609–626.

Plate, T. A. (1994). Distributed representations and nested compositional structure. (Doctoral dissertation, Department of Computer Science, University of Toronto).

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

Rutledge-Taylor, M. F., Vellino, A., & West, R. L. (2008). A holographic associative memory recommender system, *Proceedings of the Third International Conference on Digital Information Management*, 87-92.

Ruteldge-Taylor, M. F. & West R. L. (2008). Modeling the fan-effect using dynamically structured holographic memory. *Proceedings of the 30th Annual Conference of the Cognitive Science Society*, 385-390.

Sahlgren, M., Holst, A., & Kanerva, P. (2008). Permutations as a means to encode order in word space. *Proceedings of the 30th Annual Conference of the Cognitive Science Society*, 1300-1305.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence, 46*, 159-216.