Representing Objects, Relations, and Sequences

Stephen I. Gallant
sgallant@mmres.com
Pitney Bowes and MultiModel Research, Cambridge, MA 02138, U.S.A.

T. Wendy Okaywe
wokaywe@mmres.com
MultiModel Research, Cambridge, MA 02138, U.S.A.

Vector symbolic architectures (VSAs) are high-dimensional vector representations of objects (e.g., words, image parts), relations (e.g., sentence structures), and sequences for use with machine learning algorithms. They consist of a vector addition operator for representing a collection of unordered objects, a binding operator for associating groups of objects, and a methodology for encoding complex structures. We first develop constraints that machine learning imposes on VSAs; for example, similar structures must be represented by similar vectors. The constraints suggest that current VSAs should represent phrases ("*The smart Brazilian girl*") by binding sums of terms, in addition to simply binding the terms directly. We show that matrix multiplication can be used as the binding operator for a VSA, and that matrix elements can be chosen at random. A consequence for living systems is that binding is mathematically possible without the need to specify, in advance, precise neuron-to-neuron connection properties for large numbers of synapses. A VSA that incorporates these ideas, Matrix Binding of Additive Terms (MBAT), is described that satisfies all constraints.

With respect to machine learning, for some types of problems appropriate VSA representations permit us to *prove* learnability rather than relying on *simulations*. We also propose dividing machine (and neural) learning and representation into three stages, with differing roles for learning in each stage. For neural modeling, we give representational reasons for nervous systems to have many recurrent connections, as well as for the importance of phrases in language processing. Sizing simulations and analyses suggest that VSAs in general, and MBAT in particular, are ready for real-world applications.

1 Introduction

Representation is an important topic in its own right. Perhaps the most successful representation ever invented (other than writing itself) is the decimal representations of integers, a great advance over counting by simple hash marks. Decimal representations illustrate that a good representation can help us calculate more quickly, by orders of magnitude, and thereby enable computations that would otherwise be impossibly difficult. This is precisely our goal here. We want a representation of objects (e.g., words), multiple relations of those objects (e.g., assorted syntactic and semantic information), and sequences (e.g., sentences) that is hospitable to machine learning. For another example in the computer vision domain, we are interested in representing image parts, relations among those parts, and sequences of (sub) image presentations (e.g., from eye saccades).

We seek a representation that permits us to use standard machine learning techniques (e.g., neural networks, perceptron learning, regression) to simultaneously learn mappings of objects, relations, and sequences. Moreover, we want to use standard algorithms out of the box on vector inputs, without the need for constructing a separate learning architecture for each task. This would open the possibility of higher-order holistic modeling, where the predicted output encodes objects simultaneously with their structure and where the structure can be more complex than selection from a small set of options. For example, we want to be able to predict full parse trees in one shot rather than word-for-word part-of-speech tags. Ultimately we would like to predict a translated or summarized sentence or a transformed image representation.

These are longer-term goals. A more immediate motivation for developing such representations is to facilitate the use of machine learning when starting from the outputs of structured classification approaches. For example, Collobert, Weston, Bottou, Karlen, and Kuksa (2011) produce a system that outputs structure information (part of speech, chunks, semantic roles) for each word in a sentence. We want to be able to cleanly incorporate these outputs into a fixed-length vector representing the entire sentence, for use with follow-on machine learning.

To address these goals, since the 1990s a number of investigators have worked on incorporating structure into high-dimensional, distributed vector representations. (A *distributed vector* represents objects or other information by patterns over the entire vector.) Following Levy and Gayler (2008), we refer to these architectures as *Vector Symbolic Architectures* (VSAs).

The desire to use machine learning techniques places a number of constraints on *representation*. Inputs and outputs to standard machine learning algorithms are most conveniently expressed as fixed-length vectors—vectors having a prespecified number of components. Thus, we cannot directly apply neural networks to sentences, because sentences have no fixed and bounded length (and also because they possess important structure that is not immediate from the string of letters in a sentence).

Here we will focus on developing a VSA that simultaneously represents multiple objects, multiple versions of relations among those objects, and

Table 1: Computational Example with 10-Dimensional Vectors.

Smart	Girl	Saw	Gray	Elephant	V = smart + girl
Vectors for words					
−1	1	−1	1	1	0
1	1	−1	−1	−1	2
1	1	1	−1	1	2
−1	−1	1	−1	−1	−2
−1	−1	1	−1	−1	−2
−1	1	−1	−1	1	0
1	−1	1	−1	−1	0
−1	−1	1	1	1	−2
−1	−1	1	1	−1	−2
1	−1	−1	−1	−1	0
Dot products with V					
12	**12**	**−8**	**−4**	**4**	

Note: This illustrates the sum of two vectors and the process for recognizing individual constituents from a sum using the dot product.

sequences of such objects or relations using a single fixed-length vector in a way that satisfies the representational constraints. We name the VSA we develop *Matrix Binding of Additive Terms* (MBAT).

1.1 Vector Symbolic Architectures.

To help with basic intuition for VSAs, consider Table 1, where five terms (*smart*, *girl*, *saw*, *grey*, *elephant*) are shown with their corresponding vectors ($\mathbf{V}^{\mathbf{smart}}$, $\mathbf{V}^{\mathbf{girl}}$, etc.). Notationally, we represent all matrices by **M** and vectors by other capital letters, such as **V**, **W**. We also follow the standard convention of representing the vector for a term ($\mathbf{V}^{\mathbf{smart}}$) by just the term (**smart**) where the context is clear. Table 1 suggests that we have a way of recognizing individual constituents of a vector sum using dot products (vector inner products). This will be formalized in section 3.

Vector symbolic architectures trace their origins to Smolensky's (1990) tensor product models, but avoid the exponential growth in vector size of those models. VSAs include Kanerva's (1994, 1997) *Binary Spatter Codes* (BSC), Plate's (1992, 2003) *Holographic Reduced Representations* (HRR), Rachkovskij and Kussul's (2001) *Context Dependent Thinning* (CDT), and Gayler's (1998) *Multiply-Add-Permute* coding (MAP). Vector symbolic Architectures can be characterized along five defining characteristics:

- Components of vectors are binary (BSC), sparse binary (CDT), "bipolar" (+1/−1) (MAP objects), continuous (HRR and MAP sums), or complex (HRR).

- Addition of vectors (also referred to as *bundling*) represents collections of (simple or complex) objects, but without any structure among the summed terms. If objects represent words, their addition gives an unordered "bag of words." Operators used for addition include normal vector addition as in Table 1 (HRR, MAP), and addition followed by conversion to binary components according to thresholds (BSC).
- Binding of vectors is used to group objects, and can also be used for ordering them. Binding operators include exclusive-OR or parity (BSC) and component-wise multiplication (MAP).

  A particularly important binding method is circular convolution (HRR). Letting **D** be vector dimensionality, the circular convolution of two vectors, $\mathbf{V} = \mathbf{X} * \mathbf{Y}$, is defined by

  $$\mathbf{V_j} = \sum_{(k=\mathbf{0},\ldots D-1)} \mathbf{X}_k \mathbf{Y}_{j-k},$$

  where the subscript calculation is taken mod **D**. In other words, reverse the numbering of **Y**'s indices, and now each component of the result $\mathbf{V}_j$ is just the dot product of **X** and (reverse-numbered) **Y**, where **Y** is first rotated *j* positions prior to taking the dot product. Binding is commutative with respect to its two operands, and VSAs typically include an inverse operation for recovering one operand if the other is known. Inverses can be mathematically exact inverses (BSC, MAP) or have mean 0 noise added to the result (HRR), in which case a cleanup step is required to find the unknown element. Cleanup consists of finding the closest resulting vector using dot products with all vectors or making use of autoassociative memories (Kohonen, 1977, Anderson, Silverstein, Ritz, & Jones, 1977). For CDT binding, sparse binary vectors (representing objects or substructures) are first ORed together forming vector **V**. Then **V** is ANDed with the union of a fixed number of permutations of **V** to control the expected number of 1s in the final vector. A separate addition operator is not needed for CDT.
- Quoting applied to binding produces unique binding operators in order to differentiate among groups joined by binding. This typically involves a random permutation of vector elements to represent, for example, two different subject phrases in a single sentence ("*The smart girl and the gray elephant went for a walk*").
- Complex structure methodology represents complex relations among objects, such as nested subclauses in a sentence. For VSAs, this consists of binding (and quoting) to get subobjects bound together, and addition to represent unordered collections of bound subobjects. For example, let us suppose each of the roles actor, verb, and object has its own vectors, as well as objects *Mary*, *loves*, and *pizza*. Then using *

to denote binding in VSAs, we might represent *"Mary loves pizza"* by the vector

$$(\underline{\textbf{actor}} * \textbf{Mary}) + (\underline{\textbf{verb}} * \textbf{loves}) + (\underline{\textbf{object}} * \textbf{pizza}).$$

This permits extraction of, say, the actor (Mary) by binding the final sum with the inverse of actor, and following with a cleanup step.

For MBAT, we will be presenting a different, unary binding operator, and a different complex structure methodology that emphasizes additive phrases.

1.2 Organization. This letter is organized as follows. We first propose in section 2 a collection of necessary *constraints* for representing structured objects for use by standard machine learning algorithms. In section 3, we describe the MBAT architecture that encodes objects, structures, and sequences into a single distributed vector. In section 4, we examine the role (and advisability) of machine learning during three information processing stages: preprocessing, representation generation, and output computation. We also see how for many tasks, with a suitable representation we can *prove* learnability for standard machine learning algorithms rather than rely on *simulations*.

Section 5 looks at capacity, namely, the required dimensionality for vectors. Both analytic estimates and simulation results are presented. Section 6 reexamines the constraints with respect to the MBAT architecture. Section 7 reviews prior research. Section 8 revisits VSAs with respect to complex structure methodology and suggests applications for MBAT in computational linguistics, computer vision, and modeling neural information processing. The appendix develops estimates for required dimensionality of vectors.

2 Requirements for a Good Representation of Objects, Relations, and Sequences

We want a representation of structured objects, such as sentences or images, that is directly suitable for machine learning, without a need for constructing special-case learning algorithms. Several requirements must be met:

Constraint 1: Fixed-Length Vector. Most standard machine learning approaches take inputs that are vectors of some prespecified length. Thus, if we want a way to simultaneously learn mappings of objects and structures, we need a way to represent many different objects and structures of those objects, simultaneously, in a single vector with a prespecified length (e.g., 1000 components). For simplicity and concreteness, we refer here to a 1000-dimensional system. However, a practical system may require a different dimensionality—either larger for increased capacity or smaller for increased speed of computations. Section 5 and the appendix explore dimensionality requirements.

Constraint 2: Distributed Representations. We need to represent hundreds of thousands of objects involved in an exponentially larger number of representations, so only one bit or vector component per object will not supply sufficient capacity. Therefore, we need to use a *distributed representation* for the vector, where information is stored in patterns and an individual component gives little, if any, information.

To take a specific example in natural language, we might represent a word as a 1000-dimensional vector, whose components are randomly generated choices of -1 and $+1$ (as in Table 1). Then we can represent a sentence as the (single vector) sum of the vectors for words in the sentence. We will examine disadvantages of this representation in constraint 4, but the sum of vectors gives one way to represent a variable-length sentence as a single, distributed, fixed-length vector.

It is amusing to note that the usual computer representation of a sentence as a text string qualifies as a distributed representation! Any individual letter gives little or no information; only the larger pattern of letters gives information. Similarly, an image bit map is also a distributed representation. These representations have a minor problem in that they are not fixed length, but they also have a major continuity problem, as discussed in constraint 4.

Constraint 3: A Complex Structure Methodology for Representing Objects and Structure Information within the Distributed Vector. For many natural language tasks, we clearly must take the syntactical structure into account. Here we encounter the binding encoding problem in cognitive science and artificial intelligence surveyed by Treisman (1999): for the word pair *"smart girl,"* we need to represent that *smart* refers to *girl*, and not some other word in the sentence. More generally, we need to be able to represent full parse information (or relations among image features) in a single distributed vector. This includes representing subclauses in sentences and representing parts of images with associated features (e.g., color, location, motion). Conversely, given a vector, we need to be able to recognize objects or structures encoded in the vector.

Constraint 4: Map Similar Objects and Structures to Similar Representations. For learning algorithms to be able to generalize, it is necessary that similar objects and structures be represented by similar vectors. This is a continuity property for maps from objects and their structures to their representations.

On the representation side, vector similarity is readily defined by Euclidean distance between vectors. Two vectors are similar if (after normalization) they are close in Euclidean distance or, equivalently, they have a significantly greater dot product than the dot product for two randomly chosen vectors. Starting with object similarity, we need to represent similar objects by similar vectors. For example, we want the vector for *smart* to be similar to the vector for *intelligent*.

Turning to structure representations, we also need to represent similar structures by similar vectors. For example, we want all of the following to have similar vectors (to varying degrees):

- *The smart girl saw the gray elephant.*
- *The gray elephant was seen by the smart girl.*
- *The smart girl I just met saw the young gray elephant eating peanuts.*

(The second case might also be considered as having a different structure but similar meaning.)

For images, we want to be able to replace similar image parts, or include additional features and structure information for an image, and have the new image vector similar to the original image.

This similarity constraint is where character strings and bit maps fail as vector representations. For a string, if we add a space, change to a similar word with a different number of letters, or switch active or passive voice, then the vector of letters changes drastically (as measured by vector Euclidean distance). Similarly, adding a row of pixels to an image can make a drastic difference in bit map vector similarity.

Constraint 5: Sequences. We need to represent sequences of objects and relations. For example, we want to represent a group of sentences, as well as images derived from a sequence of eye saccades. This requirement for sequences is especially strong for spoken language, where even a single two-syllable word like *baby* does not hit our auditory system all at once, but rather as a sequence of sounds. Thus, ultimately, we need to represent, over time, sequences of objects and relations: phonemes, words, sentences, images, or sensory inputs.

Constraint 6: Efficient Encoding into the Representation. If we want to be able to encode, say, 100,000 sentences as 100,000 vectors, we need the mapping computation from each sentence to its representation vector to be roughly linear in the length of the sentence (or number of objects and relations for the sentence). Methods that require a machine learning pass over all 100,000 sentences to represent one of the sentences, or that require n^2 computation to represent n sets of objects and structures, would seem to be impractical. Similarly, we cannot practically use a representation method that, when presented with a new object, requires recomputation of the representations for all previously seen objects.

Constraint 7: Neural Plausibility. Although not required for computational applications in language, vision, and so on, we are nonetheless interested in representations that can serve as abstract models that capture important representational functionality in living systems.

To summarize, we have listed six "must-have" constraints, along with one final "nice-to-have" constraint for representing objects, structures, and sequences so that we can use machine learning algorithms (and their extensive mathematical theory) without constructing special case learning algorithms.

3 Representing Objects, Relations, and Sequences Using a Single Distributed Vector

We now define MBAT, a vector symbolic architecture, and show how it represents objects, relations and sequences by a single, distributed, fixed-length vector while satisfying previously described constraints.

We employ two vector operations: *addition* (+) and *binding* (#), as well as a complex structure methodology of binding additive phrases as described below.

3.1 Vector Addition (+) and Additive Phrases. The familiar vector addition operator is sufficient to encode an unordered set of vectors as a single vector of the same dimension as its constituent vectors. For example, in previous work, we encoded a document as the sum of its constituent term vectors and used this document vector for information retrieval purposes (Caid, Dumais, & Gallant, 1995). The key property of vector addition, illustrated in Table 1, is:

Property 1: Addition Preserves Recognition. This property is nonintuitive. For example, with scalars if we know that six positive and negative integers added together sum to 143, we cannot say whether one of those numbers was 17. By contrast, as in Table 1, suppose we add together six 1,000-dimensional vectors with random $+1/-1$, components representing words:

$$\mathbf{V}^{\mathbf{Sum}} = \mathbf{V}^{1} + \mathbf{V}^{2} + \cdots + \mathbf{V}^{6}.$$

Let us denote the vector for the term *girl* by $\mathbf{V}^{\mathbf{girl}}$. Now we can be highly certain whether $\mathbf{V}^{\mathbf{girl}}$ was one of the six. We simply compute the inner product (dot product),

$$\mathbf{x} = \mathbf{V}^{\mathbf{girl}} \bullet \mathbf{V}^{\mathbf{Sum}} = \sum \mathbf{V}_i^{\mathbf{girl}} \mathbf{V}_i^{\mathbf{Sum}},$$

and if $\mathbf{x}$ is near 1000 the answer is *yes*; if $\mathbf{x}$ is near 0, then the answer is *no*.

Proof. If $\mathbf{V}^{\mathbf{girl}}$ is one of the six vectors, say $\mathbf{V}^{\mathbf{1}}$, then

$$\begin{aligned}\mathbf{V}^{\mathbf{girl}} \bullet \mathbf{V}^{\mathbf{Sum}} &= \mathbf{V}^{\mathbf{girl}} \bullet (\mathbf{V}^{\mathbf{girl}} + \mathbf{V}^{2} + \cdots + \mathbf{V}^{6})\\ &= \mathbf{V}^{\mathbf{girl}} \bullet \mathbf{V}^{\mathbf{girl}} + \mathbf{V}^{\mathbf{girl}} \bullet (\mathbf{V}^{2} + \cdots + \mathbf{V}^{6})\\ &= \mathbf{1{,}000} + \langle \textbf{mean 0 noise} \rangle.\end{aligned}$$

Similarly, if $\mathbf{V}^{\mathbf{girl}}$ is not one of the six vectors, then

$$\mathbf{V}^{\mathbf{girl}} \bullet \mathbf{V}^{\mathbf{Sum}} = \langle \textbf{mean 0 noise} \rangle.$$

This completes the proof except for one small point: we have to verify that the standard deviation of the ⟨**mean 0 noise**⟩ term does not grow as fast as the vector dimension (here 1000), or else the two dot products could become overwhelmed by noise, and indistinguishable for practical purposes. The appendix shows that the standard deviation of the noise grows by the square root of the vector dimension, completing the proof.

The addition property of high-dimensional vectors gets us part of the way to a good distributed representation for a collection of objects. For example, we can represent a sentence (or a document or a phrase) by a single (normalized) 1000-dimensional vector consisting of the sum of the individual word vectors. Then we can compute the Euclidean distance between vectors to find, for example, documents with vectors most similar to a query vector. This was the approach for our previous document retrieval efforts. However, we still need to represent structure among objects.

3.2 The Binding Operator. For both language and vision, relying solely on vector addition is not sufficient. Due to the commutativity of vector addition, multiple phrases such as in, *"The smart girl saw the gray elephant,"* will have exactly the same vector sum as, *"The smart elephant saw the gray girl,"* or even, *"Elephant girl gray saw smart the the."* In other words, vector addition gives us the bag of words used to create the sum, but no other structure information.

Here we run into the classic Binding Encoding Problem in cognitive science and artificial intelligence (Treisman, 1999). We need some way to bind *gray* to *elephant* and not to *girl* or any other word, while retaining a distributed representation. More generally, we need the ability to represent a parse tree for a sentence without abandoning distributed representations.

3.2.1 Phrases. It is first helpful to formalize the definition of *phrase* with respect to representations. We define a phrase as a set of items that can have their order changed without making the representation unusable. Phrases loosely correspond to language phrases, such as noun clauses and prepositional phrases, or "chunks" in computational linguistics. For example, in

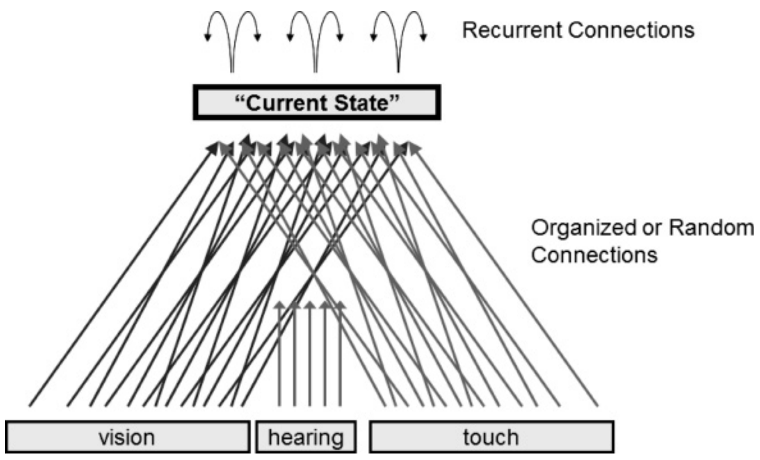


Figure 1: Recurrent connections give a way to bind inputs to the current and previous system states, as well as each other. These connections may be randomly generated.

"The smart Brazilian girl saw a gray elephant," we can reorder the leading four-word noun phrase as in, *"Brazilian the girl smart saw a gray elephant,"* and still understand the sentence, even though it becomes ungrammatical. Similarly, for machine vision, an example of a phrase would be the vectors for an associated shape, x- and y-positions, color, and motion. Again, order is not critical.

3.2.2 Neural Motivation. To motivate the binding operator we propose, consider the neural information processing schematic in Figure 1. Here we have inputs from various sensory subsystems: vision, hearing, and touch. The current state of the system ("neuron state") is modified by information from these inputs, as well as recurrent connections from itself.

The figure illustrates the Brain Binding Problem, where we need the capability of linking together diverse sensory inputs (or different neural regions) with the current state of the system. Sequences also come into play here, as when we hear *baby* as two phonemes over two time periods, we need to sequentially bind the inputs to recognize and represent the term *baby* and its associations.

For the binding task, the main resource we have to work with are the recurrent connections at the top of the figure. (Any theory of neural information processing that does not include a major role for such recurrent connections is missing a very big elephant in the neural physiology room!) Moreover, we cannot make too many organizational demands on the recurrent connections, because any complex structure needs to be passed genetically and hooked up during a noisy, messy growth phase.

So, continuing with our motivational exploration for binding, what if we take the easiest possible genetic/growth structural organization, namely, random? Can we have the recurrent connections compute a random map and have that be of any use for binding?

3.2.3 Binding Operator. Returning to the mathematics, let us now define the simplest version of a unary binding operator, #. (Below we will also define several alternatives.)

Let $\mathbf{M}$ be a fixed square matrix of appropriate dimension for our vectors, for example, 1000 by 1000. We let components of $\mathbf{M}$ be randomly chosen values (e.g., $+1/-1$). As a point of notation, when raising a matrix to a power, we will always use parentheses, as in $(\mathbf{M})^3$. This distinguishes it from the designation of several different matrices, for example, $\mathbf{M}^{\text{Actor}}$ and $\mathbf{M}^{\text{Object}}$.

If we have a sum of vectors, $\mathbf{V}^1 + \mathbf{V}^2 + \mathbf{V}^3$ (a phrase), we can bind them as part of a structure description by

$$\#(\mathbf{V}^1 + \mathbf{V}^2 + \mathbf{V}^3) \equiv \mathbf{M}(\mathbf{V}^1 + \mathbf{V}^2 + \mathbf{V}^3).$$

(The "#" operator "pounds" vectors together.) Thus, all terms in the vector of the form $(\mathbf{M})^1\ \mathbf{V}$ are differentiated from terms of the form $(\mathbf{M})^2\ \mathbf{V}$, $(\mathbf{M})^3\ \mathbf{V}$ and so on. We can think of $(\mathbf{M})^i\ \mathbf{V}$ as transforming $\mathbf{V}$ into a unique "bind space" according to i.

With respect to complex structure methodology, in MBAT *binding operates on additive phrases*, where the order of vectors in the phrase is not critical. Thus we bind

$$\#(\underline{\mathbf{actor}} + \mathbf{the} + \mathbf{smart} + \mathbf{Brazilian} + \mathbf{girl})$$
$$\equiv \mathbf{M}(\underline{\mathbf{actor}} + \mathbf{the} + \mathbf{smart} + \mathbf{Brazilian} + \mathbf{girl}).$$

Each term in the phrase may itself be the result of a binding operation, which allows us to represent complex structure (e.g., subclauses of a sentence). Some things to note:

- One of the vectors in the summed arguments can be the current state of the system, so letting $\mathbf{V}(n)$ be the current state at time n, we have the next state given by

  $$\mathbf{V}(n+1) = \mathbf{M}(\mathbf{V}(n)) + \sum \mathbf{V}^{\mathbf{inputs}}. \tag{3.1}$$

  Note that the binding operator in this formulation corresponds to the recurrent connections in Figure 1. $\mathrm{M}_{i,j}$ is the synapse between cell j and cell i. (Also, individual cells in the "current state" do not need to differentiate whether inputs are coming from feedforward sources or recurrent connections.)

- This formula for computing the next state also gives a way to represent input sequences. Kanerva (2009) and Plate (2003) previously employed this technique for sequence coding, using different binding operators.
- Phrases that get bound together must be unambiguous with respect to order. Thus, we can bind phrases like *"the smart girl,"* where order doesnot really matter in understanding the phrase. However, we could not bind in one step *"the smart girl saw the grey elephant"* because we would run into the binding ambiguity of whether *smart* refers to *girl* or *elephant*. Several binding operations would be required, as in Figure 2.
- We can make good use of tags, represented by (random) tag vectors added to phrases, to specify additional syntactic and semantic information such as $\underline{\textbf{actor}}$ (i.e., $\mathbf{V}^{\mathbf{actor}}$), $\underline{\textbf{object}}$, and $\underline{\textbf{phraseHas3words}}$.
- Computing binding (matrix multiplication) involves more work than computing circular convolution in holographic reduced representations if fast Fourier transforms are used for HRRs (Plate, 2003). Also, binding in MBAT requires us to make use of a different mathematical space, that is, matrices versus only vectors in HRRs.

Now we can see how to unambiguously represent The smart girl saw the gray elephant in Figure 2.

The resulting (single) vector, **V**, is formed from 13 object or tag vectors:

$$\begin{aligned}\mathbf{V} = (\mathbf{M})^2(&\underline{\mathbf{actor}} + \mathbf{the} + \mathbf{smart} + \mathbf{girl} + \underline{\mathbf{phraseHas3words}})\\ &+ \mathbf{M}(\underline{\mathbf{verb}} + \mathbf{saw} + \underline{\mathbf{phraseHas1word}}) + (\underline{\mathbf{object}} + \mathbf{the}\\ &+ \mathbf{gray} + \mathbf{elephant} + \underline{\mathbf{phraseHas3words}}).\end{aligned}$$

Tags such as $\underline{\textbf{phraseHas3words}}$ and $\underline{\textbf{phraseHas1word}}$, though perhaps not biologically realistic, greatly simplify the task of decoding the vector (i.e., producing the sentence encoded in the sum). If we need to speak the sentence, an approach to decoding a vector is to produce each phrase by first computing the number of words in the phrase and then finding that many terms with the highest dot products.

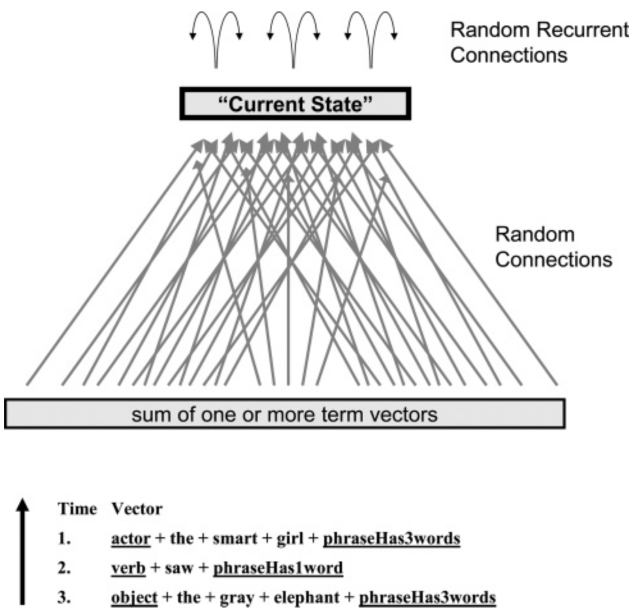
As desired, *smart* is associated with *girl* in this sum of 13 vectors, because we have term $(\mathbf{M})^2\ \mathbf{V}^{\mathbf{smart}}$ and $(\mathbf{M})^2\ \mathbf{V}^{\mathbf{girl}}$, but *elephant* appears as $\mathbf{V}^{\mathbf{elephant}}$.

We also have a recognition property for the binding operator:

Property 2: Binding (#) Preserves Recognition. Suppose we are given $\mathbf{V} = \#\ (\mathbf{V}^1 + \mathbf{V}^2 + \mathbf{V}^3)$. Can we tell if $\mathbf{V}^{\mathbf{girl}}$ is among the bound operands? Yes; we simply look at

$$\mathbf{M}\ \mathbf{V}^{\mathbf{girl}} \bullet \mathbf{V} = \mathbf{M}\,\mathbf{V}^{\mathbf{girl}} \bullet (\mathbf{M}\,\mathbf{V}^1 + \mathbf{M}\,\mathbf{V}^2 + \mathbf{M}\,\mathbf{V}^3). \tag{3.2}$$

The result follows similarly to property 1 for vector sums.



Time	Vector
1.	**actor** + the + smart + girl + **phraseHas3words**
2.	**verb** + saw + **phraseHas1word**
3.	**object** + the + gray + elephant + **phraseHas3words**

Figure 2: Representing a sentence by binding a sequence of phrase inputs. At each time step, a phrase consisting of a sum of word vectors is collected in the bottom vector. The phrase sum may also contain structure information (e.g., **actor**, **passive voice**) in the form of tag vectors. This vector is added to the random recurrent connections from $\mathbf{V}(n)$ to produce the next state vector, $\mathbf{V}(n+1)$.

3.3 Complex Structure Methodology. Figure 2 also illustrates the complex structure methodology we employ in MBAT. Binding is a *unary operator* that operates on phrases consisting of *bundled* (added) vectors. Each vector being summed may be an object (e.g., word), or the result of another binding operation (e.g., subclause). Thus *"the smart Brazilian girl"* is represented by

$$\#(\underline{\mathbf{actor}} + \mathbf{the} + \mathbf{smart} + \mathbf{Brazilian} + \mathbf{girl}).$$

Given a vector for a complex structure, we can check whether *girl* appears in any of the phrases at three outermost levels by taking a dot product with the single vector[1]

$$[(\mathbf{M})^{\mathbf{0}} + (\mathbf{M})^{\mathbf{1}} + (\mathbf{M})^{2}]\mathbf{V}^{\mathbf{girl}}. \tag{3.3}$$

[1]$(\mathbf{M})^{\mathbf{0}}$ is the identity matrix.

The dot product with $\mathbf{V}$ given by $[((\mathbf{M})^0 + (\mathbf{M})^1 + (\mathbf{M})^2\ )\ \mathbf{V}^{\mathbf{girl}}] \bullet \mathbf{V}$ will be large and positive only if $\mathbf{V}^{\mathbf{girl}}$ appears in one of the phrases—as $\mathbf{V}^{\mathbf{girl}}$, $(\mathbf{M})^1\ \mathbf{V}^{\mathbf{girl}}$, or $(\mathbf{M})^2\ \mathbf{V}^{\mathbf{girl}}$. Similarly, we can determine if *smart* and *girl* appear together in any phrase in $\mathbf{V}$ by checking if

$$\max_{i=0,1,2}[(\mathbf{M})^i(\mathbf{V}^{\mathbf{smart}} + \mathbf{V}^{\mathbf{girl}}) \bullet \mathbf{V}] \tag{3.4}$$

is sufficiently positive. Note that if *girl* appears without *smart* in a phrase, then the value above is still positive, but half of the value than when both appear in the same phrase. Also note that we cannot replace the max operator by a sum, or we run into binding ambiguity issues.

Thus, using *additive vector phrases*, $(\mathbf{V}^1 + \mathbf{V}^2 + \mathbf{V}^3)$, as operands for binding helps with subsequent recognition (and learning) of items. It also helps reduce computational demands compared to using only binding operations, because vector addition is cheaper than matrix multiplication.

3.4 Variants of the Binding Operator. As with vector addition, vector binding has several important variations:

- We can define a collection of binding operators with structural significance and give each phrase its own binding operator, such as $\mathbf{M}^{\mathbf{Actor}}$ and $\mathbf{M}^{\mathbf{Object}}$. This makes all phrases at the same level—for example,

$$\begin{aligned}\mathbf{V} = \mathbf{M}^{\mathbf{Actor}}(&\mathbf{the} + \mathbf{smart} + \mathbf{girl} + \underline{\mathbf{phraseHas3words}})\\ &+ \mathbf{M}^{\mathbf{verb}}(\mathbf{saw} + \underline{\mathbf{phraseHas1word}})\\ &+ \mathbf{M}^{\mathbf{Object}}(\mathbf{the} + \mathbf{gray} + \mathbf{elephant} + \underline{\mathbf{phraseHas3words}}).\end{aligned}$$

- As a special case, we can also define *Two-Input* binding operators. For example, if we want a binary parse tree, we can define $\#(\mathbf{V}^1, \mathbf{V}^2)$ to be $\mathbf{M}^{\mathbf{Left}}\ \mathbf{V}^1 + \mathbf{M}^{\mathbf{Right}}\ \mathbf{V}^2$, where $\mathbf{M}^{\mathbf{Left}}$ and $\mathbf{M}^{\mathbf{Right}}$ are two different fixed matrices. Note that "two input #" is noncommutative:

$$\#(\mathbf{V}^1, \mathbf{V}^2) \neq (\mathbf{V}^2, \mathbf{V}^1)$$

  as required for specifying a binary tree.
- *Binary World*: An interesting variation is to replace components of $\#(\mathbf{V}^1 + \mathbf{V}^2 + \mathbf{V}^3)$ by $+1$ if greater than or equal to 0 and -1 if less than 0 (as in Kanerva's Binary Spatter Codes). Restricting to $+1/-1$ components has the advantage of playing nicely with autoassociative learning algorithms (Kohonen, 1977; Anderson et al., 1977).

  It is worth noting that we can preserve many of the benefits of *continuous* vector components (e.g., for vector sums), while still restricting all vector components to $+1/-1$. We take a group of vector components computed from (approximately) the same connections and employ different thresholds, obtaining a binary representation for a continuous sum. For example, we can replace the first continuous component, $\mathbf{V_1}$, of an input by the group of binary components

$$\mathbf{V_{1a}} \equiv \mathbf{+1 \text{ if } (V_1 \geq 37); \text{ else } -1}$$

$$\mathbf{V_{1b}} \equiv \mathbf{+1 \text{ if } (V_1 \geq 5); \text{ else } -1}$$

$$\mathbf{V_{1c}} \equiv \mathbf{+1 \text{ if } (V_1 \geq 19); \text{ else } -1}$$

. . . .

- Permutation matrices: It is possible to use a permutation (random or not) for the binding matrix, as permutations are maximally sparse and easy to invert. However, an advantage of using matrices with many nonzero elements is that they can boost the representational dimensionality of isolated inputs. For example, suppose the goal is to learn Exclusive-Or (XOR) calculated on components 1 and 2 (and ignoring other components). A random permutation maps the two inputs to two different components but retains the same dimensionality, so that the probability of the resulting representation being linearly separable remains at 0. By contrast, in a "Binary World" architecture with $-1/+1$ components, when a nonsparse random matrix is applied to inputs followed by a thresholding step, components 1 and 2 are spread nonlinearly among many components. This increases the effective dimensionality of the representation (Gallant & Smith, 1987), and makes the probability of linear separability (and easy learnability) greater than 0. Such increased representational ability is an advantage with working in Binary World rather than using continuous vector components. Another advantage of using a nonsparse binding matrix is that the representation decays more gracefully when noise is added to the matrix. Finally, in the nervous system, the majority of neurons synapse with many other neurons rather than a single neuron, making a permutation matrix appear much less neurally plausible.
- An important performance tuning issue for practical implementations is scaling the binding operator so that, for example, an $(\mathbf{M})^2$ $\mathbf{V}^{\text{girl}}$ term does not dominate other terms. One approach is to normalize the result of a binding operation so that the resulting vector has the same length as a vector for a single term, $\sqrt{\mathbf{D}}$. Alternatively, the normalization can make each $\mathbf{M}\,\mathbf{V}^{\mathbf{i}}$ phrase component have length $\sqrt{\mathbf{D}}$. Finally, we could just work in Binary World, in which case the problem goes away.

3.5 Multiple Simultaneous Representations.

An important technique for reducing brittleness of the structure representation (such as parse information) is to simultaneously encode several structure descriptions (with different binding operators) in the vector by adding them. This increases robustness by having different structures "voting" in the final representation.

An example of multiple simultaneous representations is representing sentences as structureless additions of word vectors, plus binding of phrases, plus sequentially binding phrase components to fix their precise order. For example, with "*the smart Brazilian girl ...*", we might have

$$\begin{aligned}&(\mathbf{the} + \mathbf{smart} + \mathbf{Brazilian} + \mathbf{girl})\\ &\quad + \mathbf{M}^{\mathbf{Actor}}(\mathbf{the} + \mathbf{smart} + \mathbf{Brazilian} + \mathbf{girl})\\ &\quad + \mathbf{M}^{\mathbf{Actor\_Ordered}}(\mathbf{the} + \mathbf{M}(\mathbf{smart} + \mathbf{M}(\mathbf{Brazilian} + \mathbf{M}(\mathbf{girl})))).\end{aligned}$$

We may also specify different positive weights for each of the three groups, for example, to increase the importance of the top surface group with no binding. Multiple simultaneous representations are helpful because we cannot know, a priori, which kind of phrase grouping will be critical for capturing the essence of what is to be learned in later stages.

For another example, if parser A results in sentence representation $\mathbf{V}^{\mathbf{A}}$ and parser B produces $\mathbf{V}^{\mathbf{B}}$, then the final representation for the sentence can be $\mathbf{V}^{\mathbf{A}} + \mathbf{V}^{\mathbf{B}}$.

As a third example, if we have two (or more) image feature extraction programs (perhaps operating at different granularities on the image), each program's outputs can be converted to a vector and then added together to get the final vector representation.

To summarize this section, the two operators + and #, coupled with representing complex structure by applying # to additive phrases, permit us to represent objects, structures, and sequences in MBAT. In section 6, we check whether MBAT satisfies the representational constraints we have posed.

4 Learning and Representation: Three Stages

For both computational and neural systems, we distinguish three computational stages: *Preprocessing, Representation Generation*, and *Output Computation*. This distinction is helpful because learning plays a different role in each stage.

4.1 Preprocessing Stage.

The preprocessing stage occurs prior to actually generating a vector representation. Here is where vector representations for objects (words, images) are developed so that similar objects have similar vectors. Typically the mapping is the result of a preliminary learning phase to capture object similarities in vectors (as discussed in section 6).

As an important example of a preprocessing stage in living neural systems, there appears to be much feedforward processing of features of various complexity (e.g., line detectors, moving-edge recognizers). These computations can be genetically hardwired or learned during development, but then they do not need to be relearned in the course of the following representation generation stage. For automated systems, the identification of phrases (or chunks) is a typical preprocessing operation that can be quite helpful for the following stages.

Although the learning involved in the preprocessing stage may be computationally intensive, it is done only once and then can be used in an unlimited number of representation calculations. Thus, it avoids violating the efficient encoding constraint, because it is not a part of the representation generation stage. The features resulting from this stage serve as the inputs to the representational system.

4.2 Representation Generation Stage. Although the preprocessing stage (and output computation stage) can involve significant machine learning, there are reasons for the Representation Generating Stage to avoid machine learning:

- Internal learning means tailoring the representation for one set of applications, but this can make the representation less suited to a different set of problems.
- When representing inputs, a learning step might slow down processing so much as to make the resulting representation system impractical, thereby violating the Efficient Encoding Constraint.

We can also envision a good case for learning some vector representations as part of a separate long-term memory component, where we want to incrementally add a set of sequences to an existing set so that they may be recalled. Memory, recall, and novelty detection are important issues but beyond the scope of this letter.

4.3 Output Computation Stage. Finally, the output computation stage is clearly a place where learning is vital for mapping representations to desired outputs. Here is where we benefit from being able to use conventional fixed-length vectors as inputs. One major benefit is that a lot of previous theory is immediately applicable. These include the Perceptron Convergence Theorem (Rosenblatt, 1959; see also Minsky & Papert, 1969), the Perceptron Cycling Theorem (Minsky & Papert, 1969; Block & Levin, 1970), Cover's theorem for the likelihood of a set of vectors to be separable (Cover, 1965), and Vapnik-Chervonenkis (1971) generalization bounds. This body of theory permits us to prove learnability in many cases, as well as to set bounds on generalization.

To take a specific example, suppose we use random vector representations of words, and we have a collection of sentences with at most four phrases encoded as in Figure 2, and each sentence contains either the word *girl* or the word *elephant* (but not both). Then we can prove that perceptron learning will learn to distinguish the two cases, making a bounded number of errors in the process.

Proof. Consider

$$[((\mathbf{M})^0 + (\mathbf{M})^1 + (\mathbf{M})^2 + (\mathbf{M})^3)(\mathbf{V}^{\mathbf{girl}} - \mathbf{V}^{\mathbf{elephant}})] \bullet \mathbf{V}.$$

Excluding noise with mean 0, if $\mathbf{V}$ has $\mathbf{V}^{\mathbf{girl}}$ in a phrase, the dot product will be positive. If, by contrast, $\mathbf{V}$ has $\mathbf{V}^{\mathbf{elephant}}$ in a phrase, the computation will be negative. Therefore, the vector in brackets is a perfect linear discriminant. Now we can apply the Perceptron Convergence Theorem (Rosenblatt, 1959; see also Minsky & Papert, 1969; Gallant, 1993) to know that perceptron learning will find some error-free classifier while making a bounded number of wrong guesses. (A bound is derivable from the bracketed term.)

This proof illustrates a simple and general way of proving that these kinds of mappings are learnable using the MBAT representational framework. We merely show that at least one error-free linear classifier exists, and then we can immediately conclude that perceptron learning will learn some error-free classifier (perhaps a different one) in finite time.

Note that there is no need to decode (i.e., fully recover) vectors in the process of learning.

Reviewing the three stages, the overall processing picture is:

- A (one-time, done in the past) Preprocessing Stage, which may involve significant machine learning, feature computations, and novelty detection, which (efficiently) produces the inputs to:
- The Representation Generating Stage (our main focus), where there may be no learning, and followed by:
- An Output Computation Stage, which almost always involves machine learning for a particular task.

5 Capacity: Analytics and Simulation

For practical systems, we need to know what dimension, $\mathbf{D}$, is required for vectors to represent objects and relations, and how $\mathbf{D}$ scales with increased system sizes.

The appendix derives analytic estimates when adding $\mathbf{S}$ random $+1/-1$ vectors (referred to as the *bundled vectors*) to form vector $\mathbf{V}$, and where $\mathbf{N}$ other random vectors are present in the system. (A bundle can be the vectors in a phrase.) In particular, we derive bounds and estimates for the required dimension, $\mathbf{D}$, so that at least 98% of the time, each of the $\mathbf{S}$ bundled vectors has a higher dot product with $\mathbf{V}$ than each of the $\mathbf{N}$ other vectors. Said differently, we seek error-free separation performance at least 98% of the time.

For a small system where there are **S** = 20 bundled vectors and 1000 other random vectors, we derive that **D** = 899 dimensions guarantees error-free performance at least 98.4% of the time. An example of a small system application would be finding whether a simple diagram (collection of shapes in various configurations) is among 20 designated examples. Similarly, for a medium system with **S** = 100 bundled vectors and where there are 100,000 other random vectors, we derive an estimate for the required dimension **D** of 6927 for error-free performance 98.2% of the time. Finally, for a large system with **S** = 1,000 bundled vectors and where there are 1 million other random vectors, we derive an estimate for the required dimension **D** of 90,000 for error-free performance 99% of the time.

The approximation derived in the appendix allows us to say how required dimension **D** scales as **S** and **N** increase. In summary, for a given error threshold:

- For a fixed number of vectors **S** bundled together, as dimension **D** increases, the number of other vectors, **N**, which we can distinguish from the bundled vectors (while keeping the same error threshold), increases *exponentially* with **D**.
- For a fixed number of other vectors, **N**, as dimension **D** increases, the number of vectors **S** we can bundle together while distinguishing bundled and random vectors (and while keeping the error threshold) increases *linearly* with **D**.

Thus, representing additional bundled vectors (**S**) is fairly expensive (required **D** is linear in **S**), while distinguishing the bundled vectors from **N** other random vectors is fairly cheap (required **D** is logarithmic in **N**).

In addition to the analytic estimates in the appendix, we also performed capacity simulations for small and medium-sized problems. Here we investigated, for different values of vector dimension **D**, computing the sum of **S** vectors to form bundled vector **V**. We then found the **S** vectors with the highest dot products with **V** among the **S** bundled vectors and the **N** additional random vectors. We computed:

1. The fraction of bundled vectors that are in the **S** vectors having the highest dot product with **V**.
2. The fraction of trials that produce error-free performance: all bundled vectors have higher dot products with **V** than any of the other random vectors. (This is the same measure analyzed in the appendix.)

Figures 3 and 4 give the results. The small system required several hours computation time on a 2.4 GHz laptop, and the medium-sized system required 24 hours of simulation time.

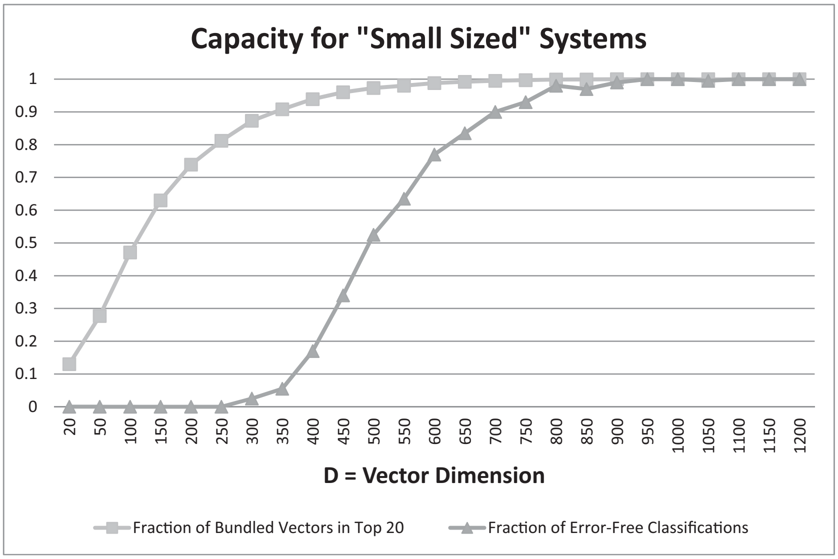


Figure 3: Capacity simulation for a small system consisting of $\mathbf{S} = 20$ vectors bundled together to form vector $\mathbf{V}$, plus $N = 1{,}000$ additional random vectors. When computing the top 20 vector dot products with V, the top series shows the fraction of bundled vectors in the top 20, and the bottom series shows the fraction of error-free separations (all bundled vectors have higher dot products with V than all random vectors). Averages over 200 trials.

From the simulations, we conclude:

- For a small system ($\mathbf{S} = 20$; $\mathbf{N} = 1{,}000$), a lower estimate for required vector dimension $\mathbf{D}$ is only 350. This gives 90% (18 of 20) of the top vectors being bundled vectors.
- For the same small system, an upper estimate for required vector dimension $\mathbf{D}$ is 900. This gives 98% probability of having error-free performance with the highest dot products all being the 20 bundled vectors.
- Similarly, for a medium system ($\mathbf{S} = 100$; $\mathbf{N} = 100{,}000$), we have lower and upper estimates for required dimension $\mathbf{D}$ of 2500 and 7000, respectively.

In the appendix, we derive an approximation formula for p, the probability of error-free performance. Letting $\mathcal{T}(\mathbf{x})$ be a one-sided tail probability in a normal distribution of a random variable being at least $\mathbf{x}$ standard deviations from the mean, we derive

$$p = \mathbf{1} - \mathbf{NS}\mathcal{T}(\surd(\mathbf{D}/(\mathbf{2S} - \mathbf{1}))),$$

where the approximation is valid when p is close to 1 (i.e., $p > 0.9$).

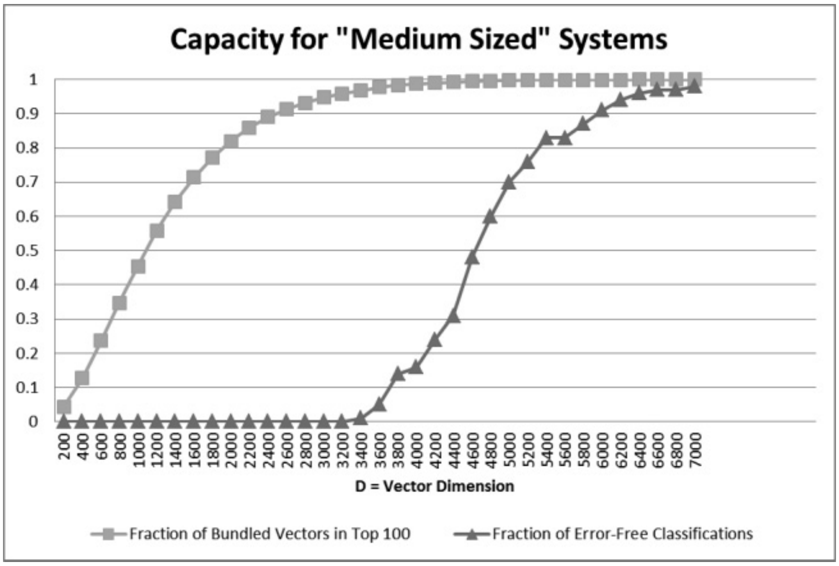


Figure 4: Capacity simulation for a medium system consisting of $S = 100$ vectors bundled together to form vector **V**, plus $\mathbf{N} = 100{,}000$ additional random vectors. When computing the top 100 vector dot products with **V**, the top series shows the fraction of bundled vectors in the top 100, and the bottom series shows the fraction of error-free separations (all bundled vectors have higher dot products with **V** than all random vectors). Averages over 100 trials.

Table 2: Simulated and Estimated Vector Dimension Required for 90% of the **S** Closest Vectors Being in the Bundle and at Least .98 Probability of Error-Free Performance.

			From Simulations		From Formula
System Size	**S** = Number of Vectors Bundled	**N** = Other Random Vectors	Required **D** for 90% of Closest Vectors Being in the Bundle	Required **D** for Probability ≥ .98 of Error-Free Performance	Required **D** for Probability ≥ .98 of Error-Free Performance
Small	20	1000	350	900	899
Medium	100	100,000	2500	7000	6,927
Large	1000	1,000,000			90,000

The analytic estimates for required dimensionality for error-free performance in the appendix are in close agreement with simulation results. Results are summarized in Table 2.

Some additional comments on capacity:

- It is possible to improve recognition ability for bundle vectors when the vectors added together are not random. For example, in natural-language applications we can use a language model (Brown et al., 1990), which gives statistics for a vector being included in the presence of other vectors in the bundle.
- When we use vectors such that similar objects have similar vectors (e.g., giving similar vectors to similar words), then this will decrease the raw discrimination ability to the extent that vectors are more similar than random. However, this should help, rather than hurt, an implementation—that is the reason we made the objects more similar in the first place!
- When machine learning follows representation in a practical system, this may require significantly *less* dimensionality than required for error-free discrimination, depending on the specifics of the learning task and performance requirements. For example, if there are **S** positive examples and **N** negative examples to be learned, we do not need $\mathbf{A} \bullet \mathbf{V} > \mathbf{R} \bullet \mathbf{V}$ for every case where **A** is a positive example and **R** is a negative example. Instead of using the sum of the positive examples, **V**, to discriminate, we have the liberty of finding any vector **X** that does a good job of making $\mathbf{A} \bullet \mathbf{X} > \mathbf{R} \bullet \mathbf{X}$ for each pair **A**, **R**. If any **X** exists that works for all pairs, it is a linear discriminant, which is easily learned by perceptron learning. Moreover, most practical modeling problems do not require perfect discriminants.

Finally, it is worth estimating computational requirements. Generating a representation vector of dimension **D** involves a $\mathbf{D} \times \mathbf{D}$ matrix multiply for each binding operation, plus vector additions as needed, regardless of the number of objects in the system. For a medium-sized system with $\mathbf{D} =$ 2000 to 7000, generating a representation on a single processor computer is clearly practical, although each binding requires 8 billion to 343 billion multiplications and additions. (Speed-ups are available using multiple processors and also by fast matrix multiplication techniques.)

For a large system with $\mathbf{D} = 90{,}000$, extensive parallelism would be required. In general, multiple processors divide the computation time by the number of processors for these computations. It is noteworthy that a living system with neurons summing synaptic inputs from many other neurons, and with neurons working in parallel, could conceivably compute binding operations of vector sums for the state update equation 3.1 in a single time step!

Follow-on learning in the output computation stage may require less computation. For example, perceptron learning requires only vector dot products and vector additions for each iteration, avoiding the more expensive matrix multiplications. Of course, the number of iterations required is also important. (Worst-case bounds on required iterations for linearly separable problems grow linearly with the dimension and the number of training examples, and grow by the square of the length of the shortest integral solution vector (Gallant, 1993).)

We conclude that vector dimension requirements and computations appear easily manageable for practical implementations.

6 Checking the Constraints

We want to verify that by using distributed vectors and two operations on vectors, addition (+) and binding (#), plus MBAT's approach for representing complex structure, we are able to satisfy the constraints from section 2. Constraints 1 (fixed-length vector) and 2 (distributed representation) are obviously satisfied. We have seen how the binding operator can represent structure (constraint 3), including a practical solution to the binding problem, as well as sequences (constraint 5). Computations are clearly linear in the number of objects, plus complexity (description size) of structures (Constraint 6).

Let us consider Constraint 4 (similar objects and structures map to similar representations). Although *similarity* is not precisely defined, nevertheless we can verify our benchmark test cases. With natural language, there are a number of ways to get similar vector representations for similar words, as surveyed in section 7. Note that this (learned) preprocessing is a one-time computation that does not affect speed during the Representation Generation Stage.

Now, suppose we encode, *"The smart girl saw the gray elephant,"* by the sum of vectors:

$$\begin{aligned}\mathbf{V} = \mathbf{M}^{\mathbf{Actor}}(&\mathbf{the} + \mathbf{smart} + \mathbf{girl} + \underline{\mathbf{phraseHas3words}}) + \mathbf{M}^{\mathbf{verb}}(\mathbf{saw}\\ &+ \underline{\mathbf{phraseHas1word}}) + \mathbf{M}^{\mathbf{Object}}(\mathbf{the} + \mathbf{gray} + \mathbf{elephant}\\ &+ \underline{\mathbf{phraseHas3words}}).\end{aligned}$$

If we have object encodings such that similar objects have similar vectors, we can interchange similar objects and leave the resulting representation vector similar to the original. For example, if $\mathbf{V}^{\mathbf{intelligent}}$ is similar to $\mathbf{V}^{\mathbf{smart}}$, then, *"The intelligent girl saw the gray elephant,"* will have a vector very similar to $\mathbf{V}$.

Turning to structure similarity, we need add only one more vector term, $\mathbf{V}^{\mathbf{Passive\_Voice}}$, to $\mathbf{V}$ in order to encode *"The gray elephant <u>was seen</u> by the smart girl."* Thus these two representation vectors are very similar as vectors. In a like manner, we can encode, *"The smart girl <u>I just met</u> saw the <u>young</u> gray elephant <u>eating peanuts</u>,"* by adding to the original vector those additional vectors for the two new clauses and adding *young* to the elephant clause. Once again, we arrive at a vector similar to the original one (i.e., significantly closer than two randomly chosen sentence vectors). The last remaining constraint, neural plausibility, is even less precise. However, we maintain that this constraint is satisfied by having a system with individual vector components (neurons), a notion of state consisting of which neurons are firing, and a way to represent objects and structural relationships in the overall state that does not make unreasonable demands on wiring of the system by genetics or during growth.

7 Prior Research

Vector symbolic architectures were summarized in section 1 and are further discussed in the next section.

A key early source for distributed representations is Hinton (1984, 1986a, 1986b). These papers present characteristics, advantages and neural plausibility arguments. (These topics are reviewed in Gallant, 1993.)

In information retrieval, the use of high-dimensional vectors to represent terms (words) was pioneered by Salton and McGill (1983). Deerwester, Dumais, Landauer, Furnas, and Harshman (1990) represented terms, documents, and queries by starting with a document-by-term matrix and then using singular value decomposition to reduce dimensionality. This also achieves some measure of similarity of representation among similar terms. A later approach used random vectors and learned modifications of those vectors to represent similarity of meaning among words in the MatchPlus system (Caid et al., 1995). The basic idea was to start with random vectors for terms and make passes over a corpus while modifying vectors by adding in a fraction of surrounding vectors (and normalizing).

In the same spirit, other language systems make good use of a language model to statistically compute probabilities of a word given its immediate predecessors (Brown et al., 1990) or computing the probability of a word given its surrounding window (Okanohara & Tsujii, 2007; Collobert & Weston, 2008). See also the Brown clustering algorithm (Brown, deSouza, Mercer, Pietra, & Lai, 1992), phrase clustering (Lin & Wu, 2009; Huang & Yates, 2009). Self-organizing maps present another possibility (Kohonen, 1995; Hagenbuchner, Sperduti, & Tsoi, 2009). However, all this information retrieval work is still within the bag-of-words limitations imposed by not having a binding operation.

More recently, Jones and Mewhort (2007) looked at incorporating positional information with semantic vectors created by a similar approach to MatchPlus. They capture order information by looking at surrounding windows (up to distance 7) for each term in a corpus. They then take the (HRR) convolution of each window, while replacing the target term by a dummy. For example, suppose we are learning the order vector for <u>King</u> in "*Rev. Martin Luther King, Jr. said . . .*" Then letting Φ be a constant placeholder for the target term *King*, we would add HRRs for *Luther* $* \Phi$, $\Phi *$ *Jr*, Luther $* \Phi *$ *Jr*, and so on. The resulting order vector is then normalized and added to the semantic vector for *King*. The result is a vector that captures semantics as well as word-order syntactic effects. Similar results were obtained by Sahlgren, Holst, and Kanerva (2008) by encoding order information with permutations (see also Recchia, Jones, Sahlgren, & Kanerva, 2010). Such vectors should provide interesting starting codings for terms in language systems, including MBAT.

With respect to matrix multiplication bindings, Hinton's (1981) "triple memory" system used random matrix connections in a subsidiary role while focusing on learning rather than representation. Also, Plate (2003, p. 22) later mentions in passing exactly the "Two-Input" version of the binding operator from section 3, which he attributes to Hinton. Plate also lists matrix multiplication as an alternative binding possibility in section 7.3, table 26.

In a computational linguistics setting, Rudolph and Giesbrecht (2010) proposed using only matrices (rather than vectors) to represent objects and examined matrix multiplication as a composition operation. In working with image data, Kelly (2010, in press) describe a representation based on square matrices, with matrix addition and multiplication as the composition operators. Similar results were obtained by Sahlgren et al. (2008) by encoding order information with permutations (see also Recchia et al., 2010). However, vector addition carried out by sparse matrices in $\mathbf{D}^2$ dimensions rather than $\mathbf{D}$ dimensions is inefficient. There is also loss of the binding recognition property once we use a large number of different matrices for multiplication rather than a small set of matrices for binding operators.

Mitchell and Lapata (2008), also in a linguistics domain, mention the binary version of the # operator in passing, although most of their efforts focus on a bag-of-words semantic space model.

When we turn to sequences, the traditional approach to dealing with sequential inputs (e.g., a sentence) is to use a sliding window. A related approach, Elman nets (1990) are three-layer neural networks that copy the hidden-layer outputs as net inputs for the next cycle, thereby producing an additional "sliding window over hidden layer outputs." Elman nets are therefore able to accumulate some state information over the entire sequence. Another related sequence approach, the time-delay neural networks of Waibel, Hanazawa, Hinton, Shikano, and Lang (1989), has several layers of groups of hidden nodes. The first node in each group sees (say) nodes 1 to 3 in the group (or input) immediately below, the second node of each group sees nodes 2 to 4 in the group below, and so on. Thus, we have a multistage fan-in from the input layer, with each stage reducing dimensionality while expanding global coverage.

All three of these approaches typically employ backpropagation (or variants) to adjust weights in response to training data. Therefore, they are primarily learning algorithms rather than approaches for representation. Although we could consider hidden-layer activations as representations for new inputs after learning has ended, there is a limited ability to recognize stored objects, and the only type of structure that is explicitly captured is sequentiality. Nevertheless, these techniques might prove useful in a preprocessing stage prior to generating representations.

For sequence representations that do not require learning, Kanerva (1988) represents sequences using pointer chains. Later, Plate (2003) employs trajectory association, where the idea is to bind powers of a vector to sequence information. For example, if we want to represent the sequence A, B, C, we can take some fixed vector **V** and compute

$$\mathbf{V} * \mathbf{A} + \mathbf{V} * \mathbf{V} * \mathbf{B} + \mathbf{V} * \mathbf{V} * \mathbf{V} * \mathbf{C}.$$

There are additional variations involving helper terms for easier decoding.

There is also a body of research on learning with structural inputs, much of which involves using backpropagation-related algorithms to learn weights in a predefined network without directed loops (Frasconi, Gori, & Sperduti, 1998). Again, the focus is on learning rather than representation. The backpropagation computations (along with potentially large numbers of hidden units) make this approach impractical for generating general-purpose representations.

Another early work involving reduced representations is Pollack's (1990) RAAM (Recursive Auto Associative Memory) architecture and later extensions, for example, LRAAM (labeling RAAM) by Sperduti, Starita, and Goller (1995). These approaches use backpropagation learning (or variants) on a network of inputs, hidden units, and outputs that attempt to reproduce inputs. The hidden units, after learning, encode the reduced representations of the inputs. A drawback of these approaches is the need for learning over all inputs to achieve the representations of the inputs. For example, adding input cases requires relearning the representation for all previous input patterns using backpropagation (violating the Efficient Coding Constraint). Improvements in capacity and generalization were reported by Voegtlin and Dominey (2005). Although these approaches are all too slow (nonlinear) for the representation generation stage, their abilities to capture generalization may present good synergy as part of the preprocessing stage.

Another important line of research for learning structures with generalization was Hinton's (1986a, 1986b, 1990) family tree tasks, followed by linear relational embedding (Paccanaro & Hinton, 2001a, 2001b; Paccanaro 2003). As with RAAM architectures, generalization ability may prove useful in producing preprocessing stage inputs for MBAT or other approaches.

Sperduti (1997) proposed a generalized recursive neuron architecture for classification of structures. This complex neuron structure can be seen as generalizing some other ways of encoding structure, including LRAAM, but representation size grows with respect to the size of the structure being represented, as do computational requirements.

Another recent approach represents structures by dynamic sequences, but requires a principal component analysis (PCA) for obtaining the representation. Sperduti (2007) reports a speed-up for PCA calculation; the result could play a role in the preprocessing stage learning of inputs, but is still too computationally demanding for computing representations in the Representation Generation Stage. It is worth noting that Sperduti et al. (1995) conduct *simulations* to show good performance for learning to discriminate the presence of particular terms in the representation. By contrast, section 4 proves *learnability* for a similar task, without the need for simulation.

More recently, Collobert and Weston (2008) show how a general neural network architecture can be simultaneously trained on multiple tasks (part-of-speech tags, chunks, named entity Tags, semantic roles, semantically similar words) using backpropagation. They encode sentences using the Time-Delay Neural Networks of Waibel et al. (1989). The network learns its own vector representations for words during multitask learning, so that different tasks in effect help each other for a shared portion of the vector representation for each task. However, each task ends up with its own vector representation for the nonshared part. Outputs consist of either a choice from a finite set (e.g., part-of-speech tags) or a single number (probability) on a word-by-word basis. It is not apparent that their internal vector encodings can serve as representations for, say, sentences, because each learning task produces a different vector for the same sentence. Nor are the sets of outputs, produced for each word, easy to directly convert into a fixed-length vector for the sentence.

However, there is an appealing natural synergy of Collobert and Weston's (2008) system with the representation we examine, because outputs from their system can serve as structure information to be included in the representation vector. In particular, the chunking outputs can identify phrases, which are ideal candidates for binding operations in constructing the vector for a sentence. Vector symbolic architectures in general, and MBAT in particular, give a natural way to leverage word-by-word information as inputs to learning algorithms by converting it to fixed-length, distributed vectors.

In a later work, Collobert et al. (2011) develop a unified neural network architecture for these linguistic tasks, where all tasks are trained using two somewhat complex architectures based on time-delay neural networks. As in previous work, their system outputs a set of tags for each word. Training time is 1 hour to 3 days, depending on the task, and scoring of new input is very fast after training. Their impressive point is that, at least for producing various tags for words in the domain of language, one of two neural network learning architectures can produce excellent results for a variety of tagging tasks. The architecture is highly tuned to producing word-by-word Tags from language input. Therefore, it seems very hard to adapt this approach to other tasks such as machine translation, where more general output structures must be produced, or to other domains such as image processing, without first converting outputs to fixed-length vectors.

These papers are examples of a broader category, structured classification, where for each sequential input object, we compute either one choice from a fixed and finite set of choices or we compute a single scalar value. There is much other research in the structured classification paradigm, which we do not review here.

Recently Socher, Manning, and Ng (2010), Socher, Lin, Ng, and Manning (2011), and Socher, Pennington, Huang, Ng, and Manning (2011) have shown how binding matrices can be learned using backpropagation with complex loss functions. Socher's Recursive Neural Networks (RNN) are binary trees with distributed representations, which are structurally identical to the "two-input" binding operators in Section 3. In particular, Socher, Lin et al.'s matrix (2011) for combining two vectors is equivalent to concatenating rows from $\mathbf{M}^{\text{Left}}$ and $\mathbf{M}^{\text{Right}}$ to form a single "double-wide" matrix for applying to pairs of concatenated column vectors.

The RNN approach is applied to Penn Treebank (http://www.cis.upenn.edu/~treebank/) data to learn parsing and the Stanford background data set (http://dags.stanford.edu/projects/scenedataset.html) to obtain a new performance level for segmentation and annotation. Socher et al. (2010) also report excellent results with the WSJ development data set and an unlabeled corpus of the English Wikipedia (Socher, Pennington, et al., 2011). Socher's work demonstrates that the kind of vector symbolic architectures we describe can be useful for practical problems.

Finally, there is interesting work by Maass, Natschläger, and Markram (2002) on asynchronous modeling of noisy neuron behavior, including recurrent connections, in their Liquid State Machine model. They show the ability to train output neurons to discriminate noisy input patterns in neurons with fully asynchronous firing times. This modeling is at a granular neuron level and is therefore more suited for neural modeling and less suited for large-scale, practical systems. For example, simulations typically deal with distinguishing among a small number of input patterns, and there is no attempt at explicitly representing complex structure among objects. Nevertheless, the Liquid State Machine models of Maass and colleagues share several characteristics with the kind of representational systems we examine:

- They are general representations, not tuned to any specific task.
- There is a specific task or output readout stage that involves learning.
- State transitions are similar to equation 3.1.
- The overall system does not need to converge to a stable state, as with most other learning algorithms.
- The mathematical model can be used to hypothesize computational explanations for aspects of neural organization and processing.

Maass et al. investigate local versus long-range recurrent connections, giving computational explanations for the distributions. They find that less-than-complete connections work better than complete connections with their asynchronous, noisy, dynamic models. (This result may not apply to nonasynchronous systems.)

8 Discussion

We have shown that a desire to apply standard machine learning techniques (neural networks, perceptron learning, regression) to collections of objects, structures, and sequences imposes a number of constraints on the *representation* to be used. Constraints include the necessity for using distributed, fixed-length vector representations, mappings of similar objects and structures into similar vector representation, and efficient generation of the representation. In response we have developed MBAT, a neurally plausible vector symbolic architecture that satisfies these constraints. MBAT uses vector addition and several possible variants of a vector binding operator, plus a complex structure methodology that focuses on additive terms (i.e., phrases).

8.1 MBAT as a Vector Symbolic Architecture. Viewed from the perspective of Vector Symbolic Architectures, MBAT can be characterized as follows:

- Vector components are either continuous or two-valued (e.g., $+1/-1$).
- Addition is vector addition, optionally followed by thresholding as in Binary Spatter Codes.
- Binding is a unary operator consisting of matrix multiplication. Either one matrix or a matrix chosen from a small set of matrices ($\mathbf{M}^{\mathbf{Actor}}$, $\mathbf{M}^{\mathbf{Object}}$) is used for binding. Components of matrices can be chosen at random and can have $+1/-1$ entries. If vectors are restricted to having $+1/-1$ components, then matrix addition and multiplication are followed by a thresholding step. Another variation adds the original binding operands back into the result of matrix multiplication. A two-argument version of binding is available by multiplying each argument by one of two fixed matrices, and adding the results.
- Quoting is by repeated matrix multiplication or by using different matrices from a small set of matrices. This is similar to quoting by permutation (e.g., Gayler, 2003; Kanerva, 2009; Plate, 2003), except we need not restrict ourselves to permutation matrices. (See the comments in section 3 on permutations.)
- The complex structure methodology applies (unary) binding to additive phrases, $\mathbf{M}(\mathbf{V} + \mathbf{W} + \mathbf{X})$. Each of the added terms may be the result of another binding operation. Several different representations can be simultaneously represented by adding their vectors.

8.2 Vector Symbolic Architectures and Complex Structure Methodology. The procedure for encoding complex structure deserves further comment with respect to VSAs in general and Holographic Reduced Representations in particular.

For VSAs, Context-Dependent Thinning will map similar structures to similar vectors, as required by constraint 4. Other VSA methods can run into problems with this constraint: there is no vector similarity between $\mathbf{V} * \mathbf{W}$ and $\mathbf{V} * \mathbf{W} * \mathbf{X}$. For example, ($\mathbf{smart} * \mathbf{girl}$) and ($\mathbf{smart} * \mathbf{Brazilian} * \mathbf{girl}$) have no similarity.

Another troublesome case for all VSAs is representing repeated objects whenever binding is used for phrases. We might like ($\mathbf{tall} * \mathbf{boy}$), ($\mathbf{very} * \mathbf{tall} * \mathbf{boy}$), and ($\mathbf{very} * \mathbf{very} * \mathbf{tall} * \mathbf{boy}$) to each be different, yet with appropriate similarity between pairs of phrases. However HRRs give no relation between pairs of phrases, and BSC gives no difference between phrases 1 and 3 (unless various workarounds are employed).

The issue is whether to represent phrases by binding the terms directly, ($\mathbf{V} * \mathbf{W} * \mathbf{X}$), or binding their sum, $\#(\mathbf{V} + \mathbf{W} + \mathbf{X})$, as in MBAT.

Note that in HRRs we can convert a two-argument $*$ operator to a unary $*$ operator by

$$*(\mathbf{V} + \mathbf{W} + \mathbf{X}) \equiv \mathbf{dummy} * (\mathbf{V} + \mathbf{W} + \mathbf{X}).$$

(There are other ways to create unary operators from two-argument operators.) When included in representations, such additive phrases permit HRRs to map similar phrases to similar vectors, thereby satisfying constraint 4 in section 2. We now have

$$*(\mathbf{smart} + \mathbf{Brazilian} + \mathbf{girl})$$

similar to

$$*(\mathbf{smart} + \mathbf{girl}),$$

which was not the case with ($\mathbf{smart} * \mathbf{Brazilian} * \mathbf{girl}$) and ($\mathbf{smart} * \mathbf{girl}$). Computation is also reduced, because $*$ requires more work than computing a vector addition. Moreover, it is also much easier to recognize whether $\mathbf{V}^{\mathbf{girl}}$ is in $*(\mathbf{smart} + \mathbf{Brazilian} + \mathbf{girl})$ than it is to recognize whether it is in ($\mathbf{smart} * \mathbf{Brazilian} * \mathbf{girl}$), because we can take a single dot product, similar to equations 3.2 to 3.4.

Thus, employing an additive phrase as the argument for a unary binding operator would appear beneficial for HRR representations (and also possibly Socher's models). Even better, we can combine an additive phrase vector with an HRR binding (rather than replacing it) as in

$$(\mathbf{V} * \mathbf{W} * \mathbf{X}) + \mathbf{dummy} * (\mathbf{V} + \mathbf{W} + \mathbf{X}).$$

This is an example of multiple simultaneous representations.

Finally, it can be seen that HRR binding of sums, as in $\mathbf{dummy} * (\mathbf{V} + \mathbf{W} + \mathbf{X})$, corresponds precisely to MBAT bindings, where the MBAT binding matrix is restricted to matrices having each row other than the first be a single rotation of the preceding row.

8.3 Jackendoff's Challenges. These complex structure issues connect to Jackendoff's (2002) challenges to cognitive neuroscience and to Gayler's (2003) response. Jackendoff issued four challenges for cognitive neuroscience, of which challenge 2, "the problem of 2," involves multiple instances of the same token in a sentence, for example, "*the little star* and *the big star*." We want to keep the two stars distinct while maintaining a partial similarity (both are stars). $\mathbf{M}^{\mathbf{Actor}}$ $(\mathbf{the} + \mathbf{little} + \mathbf{star})$ and $\mathbf{M}^{\mathbf{Object}}$ $(\mathbf{the} + \mathbf{big} + \mathbf{star})$ are different, yet both are similar to $(\mathbf{M}^{\mathbf{Actor}} + \mathbf{M}^{\mathbf{Object}})$ $\mathbf{star}$, as desired. Further, using additive phrases, we can encode the original example discussed by Jackendoff, "*The little star's beside a big star*," by the vector

$$\mathbf{M}^{\mathbf{Actor}}(\mathbf{the} + \mathbf{little} + \mathbf{star}) + \mathbf{M}^{\mathbf{Verb}}(\mathbf{'s}) + \mathbf{M}^{\mathbf{Relation}}(\mathbf{beside} + \mathbf{the} + \mathbf{big} + \mathbf{star})$$

and similarly for HRRs using additive terms.

8.4 Applications. For applications of MBAT, each modeling environment requires that we select the appropriate preprocessing stage details to take advantage of specific characteristics. For example, with language, we need to select a method for making similar terms have similar vectors, decide which chunking, tagger, or other software is available for specifying structure information, and decide which binding operator variation to employ. Similarly, for machine vision, we need to see what are the available feature detectors and what higher-level structure information is available. Once these details are specified, we can create representations using MBAT and then use standard machine learning algorithms directly out of the box to learn tasks of interest.

We believe that MBAT provides a practical playing field where machine learning can efficiently operate on objects, their structures, and sequences all at once—as either inputs or outputs. Let us briefly look at possible applications.

- For *information retrieval*, representing structure (in addition to terms) may improve performance in various learning tasks, for example, finding specific categories of articles, as in "*joint ventures where a specific agreement is announced*."
- In *natural-language processing*, MBAT gives a way to make good use of parse information keyed to sets of phrases as in

$$\mathbf{V} = \mathbf{M}^{\mathbf{Actor}}(\mathbf{the} + \mathbf{smart} + \mathbf{girl} + \underline{\mathbf{phraseHas3words}})$$
$$+ \mathbf{M}^{\mathbf{verb}}(\mathbf{saw} + \underline{\mathbf{phraseHas1word}})$$
$$+ \mathbf{M}^{\mathbf{Object}}(\mathbf{the} + \mathbf{gray} + \mathbf{elephant} + \underline{\mathbf{phraseHas3words}}).$$

  Thus, we have a direct approach, using existing taggers, for learning machine translation from paired corpora with paired sentences. For example, we can work from a collection of English sentences with corresponding French translations. (Different dimension vectors can be used for the two languages.) We take the vectors for English and French translations and then train a classifier to go from the **D** components of the French sentence vector to the first component of the English vector. The same applies for the other components of the English vector, resulting in **D** classifiers in total. The net result is a map from French words and parse structure to English words and parse structure. Whether this would work well, or would not work at all, would need to be explored in an implementation.

  A potential difficulty with translation is that it may be challenging to construct an output module that goes from a vector to a corresponding string of terms. For this task, we need to recover the sentence rather than just recognize the components of a sentence encoded by a vector. Here it is likely that embedding tags into phrases (e.g., $\underline{\mathbf{phraseHas3words}}$) will help with output. Nevertheless, constructing a vector-to-sentence module is a critical—and likely difficult—task for translation (or summarization) when using vector representations. There are other potential applications that share similarities with representing natural language, including representing genome sequences and chemical structures.
- Another application area is *computer vision*. Here a natural problem seems to be recognizing whether an image contains a specific object. Such tasks likely require multiple representations of structures at different scales in the image. This suggests combining multiple feature detectors (working on different scale sizes) and employing different binding operators (e.g., $\mathbf{M}^{\mathbf{close\_to}}$, $\mathbf{M}^{\mathbf{above}}$) to end up with sums of terms such as

$$\mathbf{M}^{\mathbf{close\_to}}(\mathbf{location} + \mathbf{shape\_1} + \mathbf{shape\_2}),$$
$$\mathbf{M}^{\mathbf{above}}(\mathbf{location} + \mathbf{shape\_1} + \mathbf{M}\ \mathbf{shape\_2}).$$

As with natural language, our representation gives a research path only to generating a practical system, but does not totally solve the problem. A creative implementation is still required.

- A final application is *neural modeling*. In particular, we want to capture the computational essence of neural information processing at a useful level of mathematical abstraction. Of course, the brain does not have complete recurrent connections, where every neuron is connected to every other. (In other words, binding matrices contain many zero terms, which does not fundamentally change the analysis.) Specialized brain substructures and many other details are also ignored in our abstract model.

  Nevertheless, the MBAT computational architecture suggests a number of computational explanations for large-scale aspects of neural organization. The most important example is that the binding operation suggests a plausible computational reason for the brain having so many *recurrent connections*. (A neuron has an estimated average of 7000 synaptic connections to other neurons.)

  A second, and more subtle, computation-based explanation is for an aspect of neural organization currently taken completely for granted: the need for a *separate memory mechanism*. In other words, why not have a unified "whole brain" that simply remembers everything? The computational explanation is that with the representation we have developed, objects and structures are expensive to store because the number of required vector components rises linearly with the number of stored items. Also, *recovery*, as opposed to *recognition*, of objects is not directly given by the representation we have explored. Hence, there is a need for a specialized memory functionality, separate from general binding, that efficiently stores large numbers of objects and facilitates recovery of stored vectors.

  Finally, the MBAT architecture can motivate cognitive science hypotheses. For example, we can hypothesize that there are neural resources devoted to recognizing phrases in language at an early processing stage. This hypothesis is supported by the computational benefits we have seen, as well as by the help given for phrase recognition in both written and spoken language: punctuation, small sets of prepositions and conjunctions, typical word ordering for phrases, pauses in speaking, tone patterns, coordinated body language, and so on. Recent magnetoencephalography studies by Bemis and Pylkkänen (2011) give additional support.[2]

For some applications, VSAs in general will need to be augmented by several very important missing pieces. These revolve around learning

[2]Localized neural response for minimal phrases (*red boat*) occurs not in traditional language areas, but instead first in areas associated with syntax, followed by areas associated with semantic processing.

(including long-term and short-term memory), storage and recall of vectors, novelty recognition and filtering, focus of attention, and dealing with large sequences or structured large objects (reading a book). Is there a good extension of MBAT that will properly accommodate these functions?

We leave such representation theory development, as well as construction of practical systems in natural language, computer vision, and other areas, to future work.

Appendix: Capacity for Vector Sums

It is possible to derive good approximation formulas for storage and recognition performance within a single distributed vector, as we show below. Previous approximations appear in the appendixes in Plate (2003) and Anderson (1973), both of which look at vector components drawn from normal distributions rather than +1/−1. Their findings for normally distributed components agree with propositions 1 to 3 below.

We use the following notation, assumptions, and simplifications:

- $\mathbf{D}$ = dimension of vectors
- $\mathbf{S}$ = number of vectors bundled together to form vector $\mathbf{V}$
- $\mathbf{N}$ = number of randomly generated vectors that we wish to distinguish from those used in the sum forming $\mathbf{V}$
- $\mathcal{T}(\mathbf{x})$ = one-sided tail probability in a normal distribution of a random variable being at least x standard deviations from the mean
- $\mathbf{Z} = \surd(\mathbf{D}/(\mathbf{2S}-\mathbf{1}))$ for fixed $\mathbf{D}$ and $\mathbf{S}$
- All object vectors are randomly generated +1/−1 vectors.
- For simplicity, we do not include continuous vectors produced by binding operations (vectors formed by random matrix multiplications). We could, however, include such vectors if we are thresholding matrix multiplication results to obtain +1/−1 vectors.

The first thing to note is that the dot product of two random vectors has mean 0 and variance $\mathbf{D}$ (and, hence, standard deviation $\surd\mathbf{D}$), because it is the sum of $\mathbf{D}$ independent random variables, each with mean = 0 and variance = 1. Similarly, when we add $\mathbf{S}$ vectors to form vector $\mathbf{V}$, then for a random vector $\mathbf{R}$, we have $\mathbf{R} \bullet \mathbf{V}$ also has mean 0 and variance $\mathbf{SD}$, giving standard deviation $\surd(\mathbf{SD})$.

Let $\mathbf{A}$ be a randomly chosen vector from the $\mathbf{S}$ vectors $\underline{\text{A}}$dded (bundled) to form $\mathbf{V}$, and $\mathbf{R}$ a $\underline{\text{R}}$andom vector.

We are interested in the probability of a mistaken recognition where $\mathbf{R} \bullet \mathbf{V} > \mathbf{A} \bullet \mathbf{V}$. As in the proof of property 1 for vector addition,

$$\mathbf{R} \bullet \mathbf{V} = \mathbf{0} + \langle \textbf{mean 0 noise from the dot product with a sum of S vectors} \rangle$$

$$\mathbf{A} \bullet \mathbf{V} = \mathbf{D} + \langle \textbf{mean 0 noise from the dot product with a sum of S} - \mathbf{1} \textbf{ vectors} \rangle.$$

For $\mathbf{D}$ large enough, the central limit theorem of statistics guarantees that the first noise term will be closely approximated by a normal distribution with mean 0 and standard deviation $\surd(\mathbf{SD})$, denoted $\mathcal{N}(\mathbf{0}, \surd(\mathbf{SD}))$. Similarly, the second noise term is closely approximated by $\mathcal{N}(\mathbf{0}, \surd((\mathbf{S}-\mathbf{1})\mathbf{D}))$. So the probability of a mistake with these two vectors is given by the probability of a random value selected from $\mathcal{N}(\mathbf{0}, \surd(\mathbf{SD}))$ being greater than a random value selected from $\mathcal{N}(\mathbf{D}, \surd((\mathbf{S}-\mathbf{1})\mathbf{D})$. This is equivalent to the probability of a random vector being negative when selected from $\mathcal{N}(\mathbf{D}, \surd((\mathbf{2S}-\mathbf{1})\mathbf{D})$, because the difference of two normal distributions, $\mathbf{X}-\mathbf{Y}$, is a normal distribution having mean equal to the difference of the two means, and variance equal to the sum of the two variances. (Many thanks to the referee who pointed this out.)

Proof. From basic definitions, if $\mathbf{Y}$ is normally distributed, then so is $(-\mathbf{Y})$, with $\text{mean}(-\mathbf{Y}) = -\text{mean}(\mathbf{Y})$ and with $\text{Var}(-\mathbf{Y}) = \text{Var}(\mathbf{Y})$. The result now follows from well-known properties for the sum of two normal variables, applied to $\mathbf{X}$ and $(-\mathbf{Y})$.

Thus, looking at standard deviations, an error occurs when a difference is in the tail probability at least $\mathbf{D}/\surd((\mathbf{2S}-\mathbf{1})\mathbf{D}) = \surd(\mathbf{D}/(\mathbf{2S}-\mathbf{1}))$ standard deviations from the mean.

Here it is convenient to introduce some simplifying notation. We define:

$$\mathbf{Z} = \surd(\mathbf{D}/(\mathbf{2S}-\mathbf{1})) \text{ for fixed } \mathbf{D} \text{ and } \mathbf{S}.$$

Thus, for prespecified $\mathbf{D}$ and $\mathbf{S}$, we have $\mathbf{Z}$ corresponding to $\mathbf{D}$ as measured in standard deviations of the difference in noise terms. We also adopt the notation $\mathcal{T}(\mathbf{x})$, a one-sided tail probability of a random variable being at least x standard deviations from the mean in a normal distribution. Thus an estimate for the error with the pair of randomly chosen vectors, one from the bundled vectors and some other random vector is

$$\mathcal{T}(\mathbf{Z}) = \mathcal{T}(\surd(\mathbf{D}/(\mathbf{2S}-\mathbf{1}))). \tag{*}$$

Now we only need to note that the tail probabilities of a normal distribution decrease exponentially (i.e., as $\mathbf{e}^{-\mathbf{x}}$) with the number of standard deviations to conclude:

Proposition 1. *For a fixed number, $\mathbf{S}$, of random +1/−1 vectors bundled together to get vector $\mathbf{V}$, the probability of a random vector having a greater dot product with $\mathbf{V}$ than a randomly selected vector in the sum decreases exponentially with vector dimension, $\mathbf{D}$.*

Proof. $\mathcal{T}(\surd(\mathbf{D}/(\mathbf{2S}-\mathbf{1})))$ decreases exponentially as $\mathbf{D}$ increases.

We are really interested in the probability of all **S** of the vectors in the sum having greater dot product with **V** than any of **N** random vectors. This probability is given by

$$[\mathbf{1} - \mathcal{T}(\mathbf{Z})]^{\mathbf{NS}} \approx \mathbf{1} - \mathbf{NS}\,\mathcal{T}(\mathbf{Z}),$$

assuming $\mathbf{NS}\,\mathcal{T}(\mathbf{Z})$ is close to 0, and dropping higher-order terms. Thus, we have:

Proposition 2. *For a fixed number, **S**, of random +1/—1 vectors bundled together to get vector **V**, if we generate **N** additional random vectors, then the probability that each random vector has less dot product with **V** than each vector in the sum equals 1 minus an exponentially decreasing probability as vector dimension, **D** increases.*

In other words, as **D** increases, the probability of less than error-free discrimination between **S** vectors in the bundle and **N** random vectors decreases exponentially.

Proposition 3. *For a fixed number, **N**, of random +1/—1 vectors, the number of vectors, **S**, in a bundle that can be perfectly discriminated against with constant probability increases nearly linearly with vector dimension **D**. (More precisely, the required **D** increases linearly with **S**, plus second-order terms.)*

In summary, as **D** increases, the number of random vectors we can discriminate from a sum of **S** vectors increases exponentially, and the number of vectors we can have in the sum while maintaining discriminatory power increases slightly less than linearly with respect to **D**.

It is instructive to compute several of these bounds. For a small system where we sum **S** = 20 vectors (i.e., terms, tags, image primitives) to form **V** and we have **N** = 1000 additional random vectors, we compute

$$\text{Probability of error} = (20)(1{,}000)[\mathcal{T}(\surd(\mathbf{D}/(\mathbf{2S} - \mathbf{1})))].$$

For the term in brackets, the tail probability, which is 4.8 standard deviations from the mean, is 1/1,259,000 which gives a probability of error of about 1.6%. For 4.8 standard deviations, we need **D** large enough to satisfy

$$\mathbf{4.8} = \mathbf{Z} = \surd(\mathbf{D}/(\mathbf{2S} - \mathbf{1})),\ \text{or}\ \mathbf{D} = \mathbf{4.8}^2 * \mathbf{39} = \mathbf{898.56}.$$

Thus:

- 98.4% of the time, a vector dimension of **D** = 899 will give error-free discrimination between 20 vectors in the sum and 1000 additional random vectors.

- Similarly, we can consider a medium-sized system with up to 100 vectors summed together and with 100,000 other random vectors. Here a 5.9 standard deviation value for $\mathbf{D}$ is required for 1.8% probability of error, which works out to a bound on required vector dimension of $\mathbf{D} = 6927$. (Details omitted.)
- Finally, we can consider a large-sized system with up to 1000 vectors summed together, and with 1 million other random vectors. Here a 6.7 standard deviation value for $\mathbf{D}$ is required for 1% probability of error, which works out to a bound on a required vector dimension of $\mathbf{D} = 90{,}000$.

By comparison, Plate (2003) gives simulation results that for 99% error-free performance in a small system with 14 bundled vectors and 1000 additional random vectors, the required dimension $\mathbf{D}$ is between 850 and 900, which is comparable to our simulations with $+1/-1$ components. He also derives a bound on required dimensionality where vector components are normally distributed. Letting q be the probability of error, Plate derives:

$$\mathbf{D} < 8(\mathbf{S} + \mathbf{1}) \ln(\mathbf{N}/q)$$

provided

$$\mathbf{D} > 2(\mathbf{S} + \mathbf{1})/\pi.$$

These bounds are consistent with propositions 1 to 3.

For small, medium, and large systems, Plate's bound for 1% probability of error yields required dimensions of 2000, 13,000, and 148,000, respectively. Thus, these bounds are not quite as tight as those derived above, or possibly systems with continuous vector components require greater dimensions than systems with binary components.

Acknowledgments

Thanks to Tony Plate, Pentti Kanerva, Ron Rivest, Charles Elkan, Bill Woods, and Bob Cassels for useful discussions and suggestions on an early draft, and to the referees for extremely helpful comments.

References

Anderson, J. A. (1973). A theory for the recognition of items from short memorized lists. *Psychological Review*, *80*(6), 417–438.

Anderson, J. A., & Silverstein, J. W., Ritz, S. A., & Jones, R. S. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model, *Psychological Review*, *84*, 413–451.

Bemis, D. K., & Pylkkänen, L. (2011). Simple composition: A magnetoencephalography investigation into the comprehension of minimal linguistic phrases. *Journal of Neuroscience*, *31*(8), 2801–2814.

Block, H. D., & Levin, S. A. (1970). On the boundedness of an iterative procedure for solving a system of linear inequalities. *Proc. Amer. Math. Soc.*, *26*, 229–235.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., et al. (1990). A statistical approach to machine translation. *Computational Linguistics*, *16*(2), 79–85.

Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C. (1992). Class-based *n*-gram models of natural language. *Computational Linguistics*, *18*, 467–479.

Caid, W. R., Dumais, S. T., & Gallant, S. I. (1995). Learned vector-space models for document retrieval. *Information Processing and Management*, *31*, 419–429.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning* (pp. 160–167). New York: ACM.

Collobert, R., Weston, J., Bottou, L., Karlen, K. K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*, 2461–2505.

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, *14*, 326–334.

Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the Society for Information Science*, *41*(6), 391–407.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Frasconi, P., Gori, M., Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, *9*, 768–786.

Gallant, S. I. (1993). *Neural network learning and expert systems*. Cambridge, MA: MIT Press.

Gallant, S. I., & Smith, D. (1987). Random cells: An idea whose time has come and gone. . .and come again? In *Proceedings of the IEEE International Conference on Neural Networks* (Vol. 2, pp. 671–678). Piscataway, NJ: IEEE.

Gayler, R. W. (1998). Multiplicative binding, representation operators, and analogy [Abstract of poster]. In K. Holyoak, D. Gentner, & B. Kokinov (Eds.), *Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences*. Sofia, Bulgaria: New Bulgarian University.

Gayler, R.W. (2003). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In P. Slezak (Ed.), *Proceedings of the ICCS/ASCS International Conference on Cognitive Science* (pp. 133–138). Sydney, Australia: University of New South Wales.

Hagenbuchner, M., Sperduti, A., & Tsoi, A. C. (2009). Graph self-organizing maps for cyclic and unbounded graphs. *Neurocomputing*, *72*, 1419–1430.

Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory*. Hillsdale, NJ: Erlbaum.

Hinton, G. E. (1984). *Distributed representations* (Tech. Rep. CMU-CS-84-157). Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

Hinton, G. E. (1986a). Distributed representations. In D. E. Rumelhart & J. L. McClelland (Eds.). *Parallel distributed processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, Cambridge, MA: USA.

Hinton, G. E. (1986b). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 1–12). Mahwah, NJ: Erlbaum.

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

Huang, F., & Yates, A. (2009). Distributional representations for handling sparsity in supervised sequence labeling. In *Proceedings of the Meeting of the Association for Computational Linguistics* (pp. 495–503). Stroudsburg, PA: Association for Computational Linguistics.

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

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

Kanerva, P. (1988). *Sparse distributed memory*. Cambridge, MA: MIT Press.

Kanerva, P. (1994). The binary spatter code for encoding concepts at many levels. In M. Marinaro & P. Morasso (Eds.), *Proceedings of International Conference on Artificial Neural Networks* (Vol. 1, pp. 226–229). New York: Springer-Verlag.

Kanerva, P. (1997). Fully distributed representation. In *Proc. 1997 Real World Computing Symposium* (Report TR-96001) (pp. 358–365). Tsukuba-City, Japan: Real World Computing Partnership.

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cogn. Comput.*, *1*, 139–159.

Kelly, M. A. (2010). *Advancing the theory and utility of holographic reduced representations* (Master's thesis, Queens University Kingston, Ontario, Canada).

Kelly, M. A., Blostein, D., & Mewhort, D. J. K. (in press). Encoding structure in holographic reduced representations. *Canadian Journal of Experimental Psychology*.

Kohonen, T. (1977). *Associative memory: A system-theoretical approach*. New York: Springer.

Kohonen, T. (1995). Self-organizing maps. New York: Springer.

Levy, S. D., & Gayler, R. W. (2008). Vector symbolic architectures: A new building material for artificial general intelligence. In *Proceedings of the First Conference on Artificial General Intelligence*. IOS Press.

Lin, D., & Wu, X. (2009). Phrase clustering for discriminative learning. In *Proceedings of the Meeting of the Association for Computational Linguistics* (pp. 1030–1038). Stroudsburg, PA: Association for Computational Linguistics.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*, 2531–2560.

Minsky, M., & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. Cambridge, MA: MIT Press.

Mitchell, J., & Lapata, M. (2008). Vector-based models of semantic composition. In *Proceedings of ACL-08* (pp. 236–244). Stroudsburg, PA: Association for Computational Linguistics.

Okanohara, D., & Tsujii, J. (2007). A discriminative language model with pseudo-negative samples. In *Proceedings of the 45th Annual Meeting of the ACL* (pp. 73–80). Stroudsburg, PA: Association for Computational Linguistics.

Paccanaro, A. (2003). Learning distributed representations of high-arity relational data with non-linear relational embedding. In *Proceedings of the 2003 Joint International Conference on Artificial Neural Networks and Neural Information Processing*. New York: Springer.

Paccanaro, A., & Hinton, G. E. (2001a). Learning hierarchical structures with linear relational embedding. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems, 14* (pp. 857–864). Cambridge, MA: MIT Press.

Paccanaro, A., & Hinton, G. E. (2001b). Learning distributed representations of concepts using linear relational embedding. *IEEE Trans. Knowl. Data Eng.*, *13*, 232–244.

Plate, T. A. (1992). Holographic recurrent networks. In C. L. Giles, S. J. Hanson, & J. D. Cowan (Eds.), *Advances in neural information processing systems*, *5*. San Mateo, CA: Morgan Kaufmann.

Plate, T. A. (2003). *Holographic reduced representation: Distributed representation of cognitive structure*. Stanford, CA: CSLI Publications.

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

Rachkovskij, D. A., & Kussul, E. M. (2001). Binding and normalization of binary sparse distributed representations by context-dependent thinning. *Neural Computation*, *13*, 411–452.

Recchia, G. L., Jones, M. N., Sahlgren, M., & Kanerva, P. (2010). Encoding sequential information in vector space models of semantics: Comparing holographic reduced representation and random permutation. In S. Ohisson & R. Catrambone (Eds.), *Proc. 32nd Annual Cognitive Science Society* (pp. 865–870). Austin, TX: Cognitive Science Society.

Rosenblatt, F. (1959). Two theorems of statistical separability in the perceptron. In *Proc. Symposium on the Mechanization of Thought, National Physical Laboratory*. London: H. M. Stationery Office.

Rudolph, S., & Giesbrecht, E. (2010). Compositional matrix-space models of language. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 907–916). Stroudsburg, PA: Association for Computational Linguistics.

Sahlgren, M., Holst, A., & Kanerva, P. (2008). Permutations as a means to encode order in word space. In *Proc. 30th Annual Conference of the Cognitive Science Society* (pp. 1300–1305). Austin, TX: Cognitive Science Society.

Salton, G., & McGill, M. J. (1983). *Introduction to modern information retrieval*. New York: McGraw-Hill.

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

Socher, R., Lin, C., Ng, A. Y., & Manning, C. D. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*. New York: ACM.

Socher, R., Manning, C. D., & Ng, A. Y. (2010). *Learning continuous phrase representations and syntactic parsing with recursive neural networks*. Presented at the Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2010.

Socher, R., Pennington, J., Huang, E., Ng, A. Y., & Manning, C. D. (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Stroudsburg, PA: Association for Computational Linguistics.

Sperduti, A. (1997). *A general framework for adaptive processing of data structures* (Tech. Rep. DSI-RT-15/97). Florence: Universita degli Studi di Firenze, Dipartimento di Sistemi e Informatica.

Sperduti, A. (2007). Efficient computation of recursive principal component analysis for structured input. In *Proceedings of the 18th European Conference on Machine Learning* (pp. 335–346). New York: Springer.

Sperduti, A., Starita, A., & Goller, C. (1995). Learning distributed representations for the classification of terms. In *Proceedings of the International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann.

Treisman, A. (1999). Solutions to the binding problem: Progress through controversy and convergence. *Neuron*, *24*, 105–110.

Vapnik, V. N., & Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, *16*, 264–280.

Voegtlin, T., & Dominey, P. F. (2005). Linear recursive distributed representations. *Neural Netw.*, *18*, 878–895.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *37*, 328–339.

Received October 29, 2011; accepted January 30, 2013.