flc37@cornell.edu

Graphs in space: A domain-general and level-spanning tool for representing structure.

Fred Callaway
Cornell University

May 2, 2016

Structured representation has a critical role in cognition, thus modeling structured representation has a critical role in cognitive science. However, the lack of a unified representational framework stands in the way of connecting the insights generated by models of different domains. To address this challenge, we suggest graphs as a domain-general tool for representing high level models. A further challenge arises from the need to model cognition at multiple levels of analysis. Drawing on work in Vector Symbolic Architectures, we suggest an implementation of a graph with high dimensional vectors, employing representations and operations thought to be characteristic of neural processing. We discuss approaches and challenges for modeling generalization and compositionality within this framework. Finally, we implement a simple graphical language acquisition model using the VectorGraph to demonstrate how it can be used in cognitive modeling.

1 Introduction

The representation of structure is a fundamental prerequisite for sophisticated cognition. Whether an agent wants to navigate a physical environment, select a socially successful mate, or read an undergraduate's half-baked honors thesis, she will need an internal model of the relevant system. These internal models go beyond Skinnerian stimulus-response pairings discovered through reinforcement learning: They form a coherent and veridical view of the represented system, improving the agent's ability to interact with that system (Edelman 2008). These internal models play such an important role in cognition that the study of their form, acquisition, and use makes up the majority of work in cognitive science.

Given that the goal of science is to create models of the world, cognitive scientists are presented with a unique challenge: modeling internal models, or representing

representations. As in other scientific fields, a model of internal models should be systematic and unified. It should explain how the details of specific internal models (e.g. of language) reflect general principles of mind/brain representations. Additionally, a theory of internal models must be described at multiple levels of abstraction (Edelman 2008; Marr 1982). Human representations are ultimately implemented with synaptic weights and neural activations; however, a theory at the implementational level is only partially explanatory. To fully understand a representational system, one must identify larger functional units that emerge from the representational substrate.

Computational-level theories, commonly found in linguistics, have unique explanatory power, and are essential to a complete theory of cognition. They are sometimes referred to as *rational* models because they describe the way a rational cognitive agent would solve a problem (John Robert Anderson 1990). However, for these theories to contribute to a unified theory of cognition, they must ultimately be related to lower level models. How can this be done? One option is to take a purely top-down approach, beginning with a computational model, and then constructing algorithms that implement or approximate the optimal solution (e.g. Hale 2011).

While this approach may provide some insights, there may be a limit to how far a purely top-down approach can go. A key challenge for this approach stems from the domain specific representations that computational models, especially those found in linguistics, typically employ. Ultimately, these representational abstractions must be related to neurons, ideally with a systematic theory that also explains the representations of unrelated domains. Isolated attempts of researchers in different fields to push their theories to the implementational level are unlikely to result in such a systematic explanation. Rather, we suggest that the greatest progress towards a unified theory will be achieved by considering multiple areas of cognition, and multiple levels of analysis, at the same time. Given the fundamental role of representation in cognition, a critical first step is to design a representational framework that is powerful enough to represent complex structures in a variety of domains, and yet simple enough to be a realistic goal for neuroscientists.

We suggest the graph as a potential candidate for this framework. Graphs are domain general, thus they can be used to model disparate cognitive phenomena. Additionally, with augmentations such as labeled edges, they can represent models of widely varying complexity, posed at any level of analysis. Brains themselves closely resemble a weighted directed graph with neurons as vertices and synapses as edges. Graphical models have also been applied at the level of anatomical regions, revealing the "small world" property of brain connectivity (Bullmore and Bassett 2011). Abstracting away from neurons, one can embed information and function into the nodes and edges, allowing the graph to represent abstract theories, such as those put forward in psychology and linguistics.

By representing models of different cognitive phenomena posed at different levels of analysis in one common format, we make it far easier to connect models. Two models at the same level may draw insight from each other, while a lower level model may suggest an implementation of a higher level model. For example, a linguist may incorporate many different edge labels to represent different dependency relationships between words. Meanwhile, a neuroscientist may discover a representational technique brains use to represent different connection types using

only unlabeled edges (i.e. synapses). These models can then be combined, allowing the abstract linguistic model to make specific neural predictions. Furthermore, a social psychologist studying the representation of social structure in baboons could see the parallel to her own work, and adopt the same neural model.

2 Graphical models of language

The simplest graphical model of language is the bigram model, which treats language as a first order Markov process: each word is assumed to depend stochastically on only the previous word. A bigram model is generally represented as a transitional probability matrix, that is, a graph with words as nodes and transition probabilities as edges. In this model, an utterance can be produced by starting at a node n_0 (often a special START node), and then choosing the next a node n_1 with probability equal to the edge weight from n_0 to n_1. This process can be iterated until a termination criteria is reached (often the arrival at a special STOP node). Thus, generation is modeled as searching the graph for a path between the STOP and STOP nodes

Even under the false assumption that people speak purely based on statistical dependencies between words, the bigram model is fundamentally lacking. Language is rife with long distance dependencies such as "either-or" that a bigram model cannot possibly capture. One strategy to capture long distance dependencies is to increase the order of the Markov process. For example, a second order Markov process, or trigram model, assumes that a word depends on both the previous word and the word before that one. With some squinting, a trigram model can be represented as a standard directed graph with two words in each node. For example, the transitional probability $p(w_i = z|w_{i-1} = y, w_{i-2} = x)$ would be represented as the edge between the node n_{xy} and n_{yz}.

However, increasing the Markov order has the undesirable side effect of exponentially increasing the dimensionality of the space. There are n^N possible N-grams, where N is the Markov order and n is the vocabulary size. Thus, as N increases, the percentage of grammatically valid N-grams that the learner will actually be exposed to will decrease exponentially. Many techniques in Natural Language Processing are designed to get around this problem of data sparsity, such as smoothing or variable order N-grams. For example, the back-off algorithm measures all N-gram probabilities of $N < N_{max}$, and dynamically decides which order, N to use in a probability estimation based on the number of relevant N-grams it has stored for each N (Katz 1987).

The ADIOS model (Solan et al. 2005) explores an alternative technique for tracking long distance dependencies that aims to respect the hierarchical nature of language. Unlike N-gram models which always predict the single next word based on some number of previous words, ADIOS directly models the statistical dependencies between multi-word units, e.g. between "the dog" and "ate the steak". These multi-word units or "patterns" are constructed recursively through an iterative batch-learning algorithm. When two nodes (each of which may represent any number of words) are found to frequently occur adjacently in the corpus, they are combined into a new node. Later iterations may discover that this node occurs

frequently with another node, allowing the creation of deep hierarchical patterns. The node composition function of ADIOS is a crucial development for graphical models of language. Nearly all modern syntactic theories take a binding function as a fundamental operation, although with different names: "Merge" (Chomsky 1995), "function application" (Steedman 2000), or "Unification" (Hagoort 2004). The implementation of this function (by a modeler or a language learner) is a formidable task, as we discuss below.

The second major innovation of ADIOS is the use of different classes of nodes and edges to represent slot-filler constructions. When several patterns are found to mostly overlap, with one position containing different nodes in each pattern, an *equivalence class* is identified. A unique class of node is used to represent the slot in the newly constructed pattern, with edges connecting to possible fillers. For example, upon finding the patterns "the big dog", "the nice dog", and "the recalcitrant dog", a new pattern would be created, "the E1 dog", where E1 is an equivalence class with edges pointing to "big", "nice", and "recalcitrant".

Although ADIOS demonstrated the utility of graphical representations in language modeling, the batch learning algorithm it employed casts some doubt on its relevance as a psychological model. However, this problem is not characteristic of graphical models in general. U-MILA (Kolodny, Lotem, and Edelman 2015) is an incremental model based on ADIOS that more closely reflect human learning abilities. The model is incremental, passing through the corpus a single time, building up the graph from an initially clean slate. U-MILA was found to replicate a number of psycholinguistic results in word segmentation, category learning, and structural dependency learning.

Another recent model of language acquisition, the Chunk Based Learner (McCauley and Christiansen 2014) can also be expressed as a graph. This model is similar to the "bottom-up" mode of U-MILA in that chunks are created based on transitional probabilities between words and existing chunks. Slot filler constructions are not represented. The main psycholinguistic principle behind all these models is the idea of a "chunk", a sequence of words that is treated as a single unit. As in ADIOS and U-MILA, chunking is recursive: two chunks can be combined two form another chunk. Unlike ADIOS, however CBL and U-MILA do not maintain hierarchical order in the representation of multi-word sequences. Only the process of learning the chunks is hierarchical (c.f. Christiansen and Chater 2015, section 6.2). Although CBL, U-MILA, and ADIOS have quite different theoretical motivations, they can all be expressed as a graph, facilitating direct comparison of the models.

3 Vector Symbolic Architectures

Vector Symbolic Architectures (VSAs) are a class of model that aim to implement a symbolic system with vectors in a very high dimension space (Gayler 1998; Kanerva 1988; Plate 1995). These models are often called connectionist because they share what Chalmers calls "the deepest philosophical commitment of the connectionist endeavor" (1990): distributed representation of meaning. Like more common connectionist models, (i.e. neural networks), VSA models employ simple distributed representations and linear algebra operations to transform these representations.

However, unlike neural networks, computation in VSAs is performed with algebraic operators. In addition to making the computation of VSAs relatively transparent, this gives VSA the power of recursive variable binding (Gayler 2004), an ability that neural networks have been criticized for lacking (Jackendoff 2003) [1]. As the name implies, the suggestion is that these models may provide a bridge between symbolic models and the distributed representations that are characteristic of neural processing.

VSAs are conceptually descended from the work of Smolensky, who demonstrated that variable binding can be accomplished with the tensor product operation (1990). Here, variable binding refers to the ability to rapidly synthesize novel representations by combining existing representations in a principled way, an ability that many cognitive scientists see as essential for human-like cognition (if not cognition more generally). The critical problem with Smolensky's model is that the tensor product causes dimensionality to increase exponentially as elements are recursively composed, making a large scale implementation infeasible. However, later work found that operations such as circular convolution (Plate 1995) and pairwise multiplication (Gayler 1998) serve as reasonable approximations of the tensor product, allowing all items to be represented in the same vector space, regardless of internal compositional structure.

Gayler (2004) suggests that VSAs all have three fundamental operations that are addition-like, multiplication-like, and permutation-like. However, the function of these operations differ across models, due partially to the fact that permutation and multiplication have similar properties (Kanerva 2009). Thus, we supplement Gayler's implementation-focused operator definitions with the following function-focused operators:

1. *bundle* aggregates vectors into a flat, set-like representation.
2. *label* tags a content vector with a variable/role vector.
3. *merge* composes two or more vectors into a structured, tree-like representation.

The bundle operation (terminology from Gayler 1998) is addition in every VSA we are aware of. This operation is often used to construct *memory vectors*, which store long term knowledge. Plate (1995) shows how such vectors can be used as an associative memory by labeling content vectors with a vector representing the name/variable for the stored item. The *semantic vectors* of Jones and Mewhort (2007) are also memory vectors, as are the *semantic pointers* of Eliasmith (2013).

Label and merge have widely varying and overlapping implementations. The term *bind* is sometimes used to refer to both types of operations; however we separate the terms due to their different functions. Label is the simpler of the two: it always takes two vectors, one which generally serves only as a label (e.g. a variable name), and another which generally represents a more contentful item (e.g. a word). All VSAs of which we are aware implement *label* as a single multiplicative or permutational operation.

1. VSA's are not actually recursive in the standard linguistic sense which implies unbounded recursion. Because all items are stored in vectors of the same size, the amount of interference increases as compositional depth increases, making recursion somewhat fuzzy. We see this as a strength.

Merge is more complex. It may take any number of arguments, and it can be implemented in numerous ways. Often, it is not a fundamental operation from an implementation view, but rather a composite of addition-like, multiplication-like, and permutation-like operations. For example, because circular convolution is transitive, Plate (1995) suggests that an ordered bind/merge operation could be implemented by permuting (labeling) the operands before convolving them. Jones and Mewhort (2007) employs this ordered bind operation in their own merge function, which represents N-grams surrounding a single word by binding the words in a chain, with a special vector representing the target word's location. For example, A ⊛ Φ ⊛ BIT would be added to DOG's semantic vector, representing the occurrence of "a dog bit". The merge operation can also be implemented by permutation (label) followed by addition (bundle). In this case, the label specifies the structural role of each element in the merge. This strategy maintains the property that the resulting vector is dissimilar from the constituents; however, it differs in that vectors with partially overlapping constituents (e.g. "the dog" and "a dog") will be somewhat similar (Sahlgren, Holst, and Kanerva 2008). Basile, Caputo, and Semeraro (2011) employ a similar technique; however, they label words by their syntactic dependencies rather than linear position.

The relationship of VSAs to neurons is not fully fleshed out. However, to supplement the intuition that distributed representations are more neurally representative than symbolic ones, we point to the Neural Engineering Framework (Eliasmith 2003), a neural spiking model that closely reflects neural behavior. These representations have a transparent relationship to vectors, and NEF has been described as a compiler that translates vector operations such as circular convolution into neural spikes (Eliasmith 2013). Indeed, Blouw et al. (2015) present a model of concept learning based on this framework, presenting results computed with a neural spiking model. However, the *semantic pointers* used in this model are very simple, structurally identical to the associative memories of Plate (1995). Further work is needed to determine whether structured, compositional representations and operations can be supported by this implementation.

4 VectorGraph

Thus far we have seen that the graph is a powerful and flexible tool for modeling structured representation. We have also seen that high dimensional vector spaces and a small set of operations may provide a connection between symbolic models and neurons. In this section, we unite the two frameworks, describing an implementation[2] of a graph using a VSA. If abstract models in linguistics can indeed be represented with graphs, and VSAs can indeed be implemented by neurons, a VSA implementation of graphs could lead to the unification of all three of Marr's levels of analysis.

To construct a vector representation of a graph, we begin with the traditional adjacency matrix. Noting similarities between this matrix and the co-occurrence matrices employed by distributional semantic models, we adopt a VSA-based

2. The code for the models and simulations presented in this paper can be viewed at https://github.com/fredcallaway/NU-MILA.

method that has been used effectively in distributional models: *random indexing* (see Sahlgren 2005, for a review). The resulting data structure closely mimics the behavior of an adjacency matrix representation when the vectors have sufficiently high dimensionality.

4.1 Random indexing for distributional semantic models

Distributional semantic models (DSMs), such as HAL (Lund and Burgess 1996), LSA (Landauer and Dumais 1997), and Topic Models (Griffiths, Steyvers, and Tenenbaum 2007), share with VSAs the notion that an item can be represented by a high dimensional vector. In DSMs, a word's meaning is approximated by the contexts in which it is used. The word-context associations are represented in a very large and sparse matrix, with one row for each word and one column for each document (or word, depending on how context is defined). To the extent that words with similar meaning occur in the same contexts, these words will have similar rows (similarity generally operationalized with cosine similarity).

The raw context vectors are generally too large and sparse to use effectively. To address this, distributional models employ some form of dimensionality reduction, such as singular value decomposition. An alternative technique, as suggested by Kanerva, Kristofersson, and Holst (2000), is *random indexing*. Rather than constructing the full word by document matrix and then applying dimensionality reduction, this technique does dimensionality reduction online. Each context is assigned an immutable *index vector* which in Kanerva's implementation are sparse ternary vectors. The meaning of a word is represented by a *context vector* (a domain-specific term for memory vectors). This vector is the sum of the index vectors for every context the word occurs in. Random indexing has been found to produce similar results to LSA with SVD at a fraction of the computational cost (Karlgren and Sahlgren 2001).

4.2 Random indexing for graphs

Table 1: Definition of symbols

Symbol	Meaning
a, b, x, y, n	A node
ab	A node composed of a and b
e	An edge label
$x \xrightarrow{e} y$	The edge from x to y with label e
$\boldsymbol{i}_x$	The index vector of node x
$\boldsymbol{r}_x$	The row vector of node x
$\Pi_e(\cdot)$	The permutation function for label e

Here, we generalize Kanerva's technique to represent any graph. A standard representation of a graph is an adjacency matrix, where each row/column represents the outgoing/incoming edges of a single node. Applying this interpretation to

the co-occurrence matrix used in a word-word distributional semantic model such as HAL, we have a graph with words as nodes and co-occurrence counts as edge weights. Observing that (1) random indexing can represent a co-occurrence matrix, and (2) a co-occurrence matrix can be interpreted as a graph, we suggest that random indexing can be used to represent any graph.

Indeed, we can directly map elements of Kanerva's algorithm to elements of a graph. Just as each context receives an index vector, each node receives an index vector. Context vectors become *row vectors*, which represent all outgoing edges of a node. Just as context vectors are the sum of index vectors of all contexts that a given word occurs in, row vectors are the sum of the index vectors of all nodes that a given node points to. Like context vectors, row vectors are a form of memory vector. They can be constructed incrementally with a series of edge weight increases: To bump the edge $x \to y$, we add $\boldsymbol{i}_y$ to $\boldsymbol{r}_x$.

To recover the weight of $x \to y$, we use cosine similarity: $\text{weight}(x \to y) = \cos(\boldsymbol{r}_x, \boldsymbol{i}_y)$. Intuitively, this value will be higher if $x \to y$ has been bumped many times, because this means that $\boldsymbol{i}_y$ has been added to $\boldsymbol{r}_x$ many times. We can also use cosine similarity to measure similarity between nodes: $\text{sim}(x, y) = \cos(row_x, row_y)$. Nodes that have similar outgoing edge weights will be similar because their row vectors will contain many of the same id vectors. Importantly, because random vectors in a high dimensional space tend to be nearly orthogonal, row vectors for nodes that share no outgoing edges will have similarity close to 0.

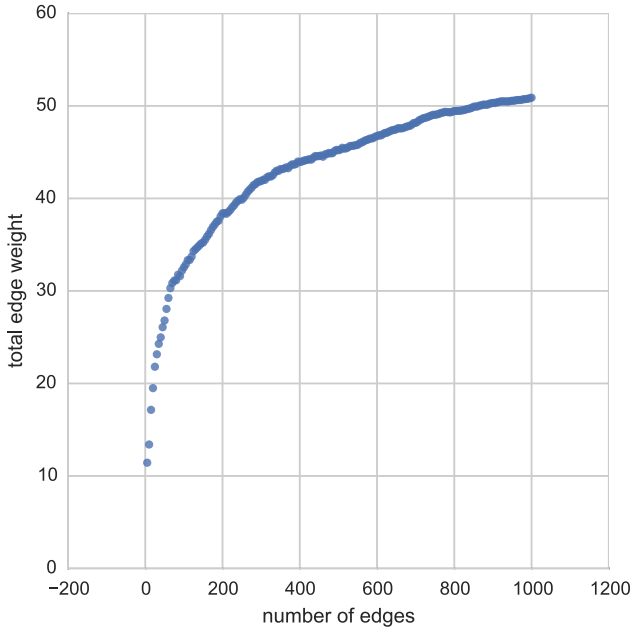

Figure 1: Total outgoing edge weight for a single node as a function of number of edges. Vectors are length 500.

An interesting attribute of this representation is that edge weights behave somewhat like probabilities. That is, bumping $x \to y$ will slightly decrease the weight of $x \to z$. Visually, adding $\boldsymbol{i}_y$ to $\boldsymbol{r}_x$ pulls $\boldsymbol{r}_x$ towards $\boldsymbol{i}_y$, and thus away from $\boldsymbol{i}_z$. However, unlike probabilities, there is no hard bound on the sum of all edge weights for a

node. The total outgoing edge weight for a single node increases as the number of outgoing edges increase, but at a decelerating rate, as shown in Figure 1.

4.2.1 Edge labels

A critical feature of U-MILA and ADIOS is the incorporation of different types of edges to represent different types of relationships. This is implemented in the VectorGraph using the *label* operation. Following Basile, Caputo, and Semeraro (2011), we use permutation as the label operation. A permutation function simply rearranges the elements of a vector, and it can be represented by another vector of the same dimensionality. Thus, each label is assigned a unique, immutable vector the first time it is used. The permutation function for a label e, represented by the vector $\boldsymbol{p}$ is

$$\Pi_e(\boldsymbol{v}) = [\boldsymbol{v}_{\boldsymbol{p}_0}, \boldsymbol{v}_{\boldsymbol{p}_1}, \ldots, \boldsymbol{v}_{\boldsymbol{p}_D}] \tag{1}$$

Given that $\boldsymbol{i}_y$ represents an edge to y, we can represent a labeled edge to y as $\Pi_e(\boldsymbol{i}_y)$. Thus, to increase the edge from x to y with label e ($x \xrightarrow{e} y$) we add $\Pi_e(\boldsymbol{i}_y)$ to $\boldsymbol{r}_x$. Similarly, to recover the weight of $x \xrightarrow{e} y$, we use $\cos(\Pi_e(\boldsymbol{i}_y), \boldsymbol{r}_x)$.

Table 2: Basic VectorGraph operations

Operation	**Meaning**	**Implementation**
$\text{bump}(x, y, e)$	increase the weight of $x \xrightarrow{e} y$	$\boldsymbol{r}_x \mathrel{+}= \Pi_e(\boldsymbol{i}_y)$
$\text{weight}(x, y, e)$	the weight of $x \xrightarrow{e} y$	$\cos(\boldsymbol{r}_x, \Pi_e(\boldsymbol{i}_y))$
$\text{sim}(x, y)$	similarity between x and y	$\cos(\boldsymbol{r}_x, \boldsymbol{r}_y)$

4.3 Generalization

Storing information in a structured form allows an agent to inform her decisions with past experience. However, for this knowledge to be widely applicable in the natural world, it cannot be rigid. This is especially true in complex systems such as language, where the exact same situation is unlikely to occur twice. When attempting to understand and react to events in these domains, an agent must generalize based on experiences similar but not identical to the current situation. For example, upon seeing a new breed of dog, you would likely still recognize it as a dog, and thus avoid leaving your meal unattended in its reach.

4.3.1 Previous approaches to generalization

Traditionally, psychology and linguistics has assumed that *categories* are the driving force behind generalization (Pothos and Wills 2011). Under this view, generalization is mediated by the application of discrete labels to groups of stimuli (as in exemplar models) or features (as in prototype models). For example, in the above example, you first identify the unfamiliar animal as a dog, and only then infer that it, like

other dogs, is liable to snatch up your dinner. Furthermore, much of the work on generalization treats categorization as an end-goal, rather than simply a means for informing actions (John R Anderson 1991; Ashby and Alfonso-Reese 1995; Kruschke 1992; Nosofsky 1986). Note a parallel to linguistics, where assigning a structure and denotational meaning to a speech act is assumed as the goal, rather than responding reasonably to the utterance (as discussed by Clark 1997).

However, explicit categorization is only one possible mechanism for generalization. An alternative approach, coming from the parallel distributed processing group (Rumelhart, McClelland, and Group 1986), is to learn higher order patterns in the environment directly, without mediating variables. This approach avoids challenges that arise with explicit categorization such as deciding when a group of items counts as a category and deciding which of several overlapping categories should determine a given property of an item (Pothos and Wills 2011). A downside to this approach, however, is that the representations and computations underlying generalization are relatively opaque to the modeler, limiting the explanatory power of these models (Griffiths et al. 2010).

Fortunately, we do not need to choose between these two approaches. Graphs are capable of representing both types of models. Many probabilistic models of categorization employ graphical representations (Tenenbaum et al. 2011), whereas artificial neural networks are themselves graphs. In addition to these extreme ends of the spectrum, the VectorGraph can take an intermediary approach. We suggest such an approach here. Like the PDP approaches, the proposed generalization algorithm does not employ explicit categories or any other latent variables. Like category approaches, the basic units of the algorithm are individual items (and their feature vectors). This gives us a balance between the flexibility of the PDP approach and the interpretability of the category approach.

4.3.2 Generalization in vector space

We view generalization as a function from a raw representation of an item to a generalized representation of that item. The function may be applied, for example, before retrieving an edge weight or measuring the similarity between nodes. In line with both PDP and categorical approaches, we take as a starting point the assumption that if two items are similar in many ways, they might also be similar in other ways. In spatial terms, if two vectors point in similar directions, they pull towards each other, becoming even more similar. If the vector space is uniformly distributed, this will only result in noise. However, if there is structure to the space (i.e. areas with higher and lower density), this results in a fuzzy version of online clustering in which vectors drift towards heavily populated areas.

Formally, for a node x, we create a generalized row vector as the sum of the all other nodes' row vectors, weighted by the similarity of each node, n to x. The generalized row vector for x is thus

$$\text{gen}(\boldsymbol{r}_x) = \sum_{n \in G} \boldsymbol{r}_n \text{sim}(x, n) \tag{2}$$

4.4 Compositionality

One form of generalization is unique and significant enough to merit separate discussion. The classical principle of compositionality, often attributed to Frege, states that the meaning of an expression is a function of the meanings of its constituent expressions and the rules for combining them. The principle is most often discussed in linguistics; however, language of thought theories (Fodor 1975; Goodman, Tenenbaum, and Gerstenberg 2014; Piantadosi 2011) suggest that compositionality may be a fundamental characteristic of other kinds of higher order reasoning (see also Werning, Hinzen, and Machery 2012).

Indeed, these models may shed greater insight on the role of compositionality in cognition. Language is a unique system because it is defined by individuals' attempts to represent it. Perhaps the compositional nature of language is a result of the human tendency to represent structure in this way. What role does compositionality play in other natural systems? Taking an example from Goodman, Tenenbaum, and Gerstenberg (2014), an agent hoping to predict the outcome of a ping pong match might do so by composing the results of previous matches (blue beats green, green beats red) using probabilistic rules that describe the system (X beats Y $\wedge$ Y beats Z $\Rightarrow$ X beats Z). Importantly, the agent's internal model generally does not perfectly describe the system, but it still leads to useful predictions.

This example points to an important distinction between compositionality as a property of a system (e.g. ping pong tournaments) and compositionality as a tool that cognitive agents use to understand that system. The first is a topic for philosophers, and perhaps physicists, and there may be deep, absolute truths regarding this kind of compositionality. The second, more relevant to cognitive scientists, is not a formal property, but rather a tool an agent may use to predict the properties of some new element based on past experience with related elements. In this sense, compositionality is a form of generalization.

4.4.1 Three approaches to compositionality in vector space

An immediate observation is that a bind operation such as circular convolution only makes a small step towards compositionality in the sense described above. The major challenge is how to encode the "rules" or patterns of compositionality. We see three possible approaches. The first is to create separate merge functions for separate domains. That is, the compositionality is encoded into the function itself. This appears to be the dominant strategy, exemplified by Plate (1995), who suggests many different ways to compose vectors. A drawback to this approach however, is that it requires knowledge of compositionality to take a fundamentally different form than other kinds of knowledge. This detracts from a major appeal of VSAs: the relatively transparent relation to neural processing. Additionally, it is likely that compositionality is itself generalized across domains to some extent. It's unclear how this could be done when the architecture of the merge function differs.

The second possible approach addresses this problem by representing the rules of compositionality numerically. For example, the compositionality of a given system could be represented as a vector which is bound to each input vector before combining them (through a binding or bundling operation). Alternatively,

compositionality could be spread across vectors, each of which is used to label a given input vector with a particular role (e.g. agent, action, patient). These vectors will of course have to be learned, which will be a significant challenge. It may be that the amount of information needed to represent compositionality will be too great for first order vectors, thus a matrix (or higher order tensor) might be required. Indeed, whereas a vector maps onto the activations of an ensemble of neurons, a matrix maps onto the synapses between two ensembles. The second may be a more likely way to represent compositional patterns, given that compositionality is generally learned gradually.

In the third approach, compositionality is not represented separately from the individual items. Rather, the way that an item combines with other items is stored directly in that item, reminiscent of combinatory categorial grammar (Steedman and Baldridge 2011). In this approach, the merge function itself could be very simple, perhaps just a bind operation. An advantage of the approach is that it treats a word's compositional behavior as no different from its other attributes, a theoretically elegant and perhaps intuitively appealing notion (e.g. adjective-iness is a feature of "red"). However, by forcing compositional features to reside in the same space as other features, the learning problem may become more difficult.

Baroni, Bernardi, and Zamparelli (2013) describe a system of this third kind in which composition is represented by the multiplication of words which are represented by variable order tensors. For example, a noun is a vector (1st order), while an adjective is a matrix (2nd order) because it is a function from nouns to nouns. A transitive verb is a 3rd order tensor because it first multiplies with an object, becoming a matrix, and then with a subject, becoming a vector. One important, and perhaps problematic, feature of this particular approach is that words with different syntactic categories have different shapes (i.e. they are different order tensors). In addition to the complications this approach creates for implementation theories, it requires that syntax be learned before, and independently from, semantics.

4.4.2 A trivial merge operation

Although we see more potential for the second and third approaches to compositionality, they are far more difficult to pursue. Thus we present a merge function of the first type, which is designed to model an especially simple kind of compositionality that can be approximated fairly well with rules over categories (e.g. syntax). To construct this function, we begin with an example rule: NP $\rightarrow$ DN. Replacing explicit categories with similarity, and the non terminal NP with its compositional structure, we can say that, [A B] will be similar to [D N] if A is similar to D and B is similar to N. We are still left with the categories D and N. Thus, in line with the generalization algorithm discussed above, we replace a category label with a weighted average of all nodes. That is, upon creating the new node [A B], we construct an initial row vector as the sum of every other chunk's row vector, weighted by the pairwise similarities of the respective constituents. Importantly, all parallel constituents must be similar (e.g. "the tasty macaroni" is not structurally similar to "ate tasty macaroni"). Thus we take the geometric mean (a multiplicative

operation) of the pairwise similarities. The row for a newly constructed node *ab* is

$$\boldsymbol{r}_{ab} = \sum_{xy \in G} \boldsymbol{r}_{xy} \sqrt{\text{sim}(a,b)\text{sim}(x,y)} \tag{3}$$

4.5 Simulations

4.5.1 Effect of dimensionality on storage capacity

To confirm that the sparse vector implementation of a graph reasonably matches a traditional graph representation, we compare the VectorGraph to a graph with true probabilities as edges, a ProbGraph. (Recall that VectorGraph edges roughly mirror probabilities). We expect that, as more edges are stored in a single vector, non-orthogonal index vectors will interfere with each other, resulting in noisy recovered edge weights. However, as the dimensionality of the vector increases, the chance of two random vectors being non-orthogonal decreases, making the edge weights more accurate.

To test this hypothesis, we provide a VectorGraph and a ProbGraph with the same random training data. If the VectorGraph implementation is sound, we expect the recovered edge weights after training to be very similar to the edge weights of the ProbGraph. However, because there is a chance that two randomly selected index vectors will not be orthogonal, the VectorGraph weights will be subject to some noise. We expect that the effect of noise will be greater for lower dimensionality vectors, and greater numbers of unique nodes. As shown in Figure 2, the results match our expectations.

4.5.2 Generalization

To test the generalization algorithm, we create a bigram model with a VectorGraph. The graph is trained on two corpora generated with probabilistic context free grammars. The grammars are nearly identical except for one determiner and one noun. The first has “that” and “table”, while the second has “my” and “bunny”. As a result, the strings “that bunny” and “my table” never occur in the combined corpus. However, the two determiners and the two nouns will have otherwise similar edge weights. If the generalization algorithm is successful, it will recognize this similarity and assign a non-zero weight the edge representing transitional probability between the unseen pairs. As shown in Figure 3, both generalization algorithms are successful.

4.5.3 Compositionality

To test the composition algorithm, we begin with the same bigram model as used in the previous simulation. We then create nodes representing noun phrases with all determiner-noun pairs, excluding THE and BOY. For each noun phrase, we assign high edge weights to SAW and ATE. As a test item, we create a new noun phrase node, [THE BOY], either using the composition algorithm or not. Critically, [THE

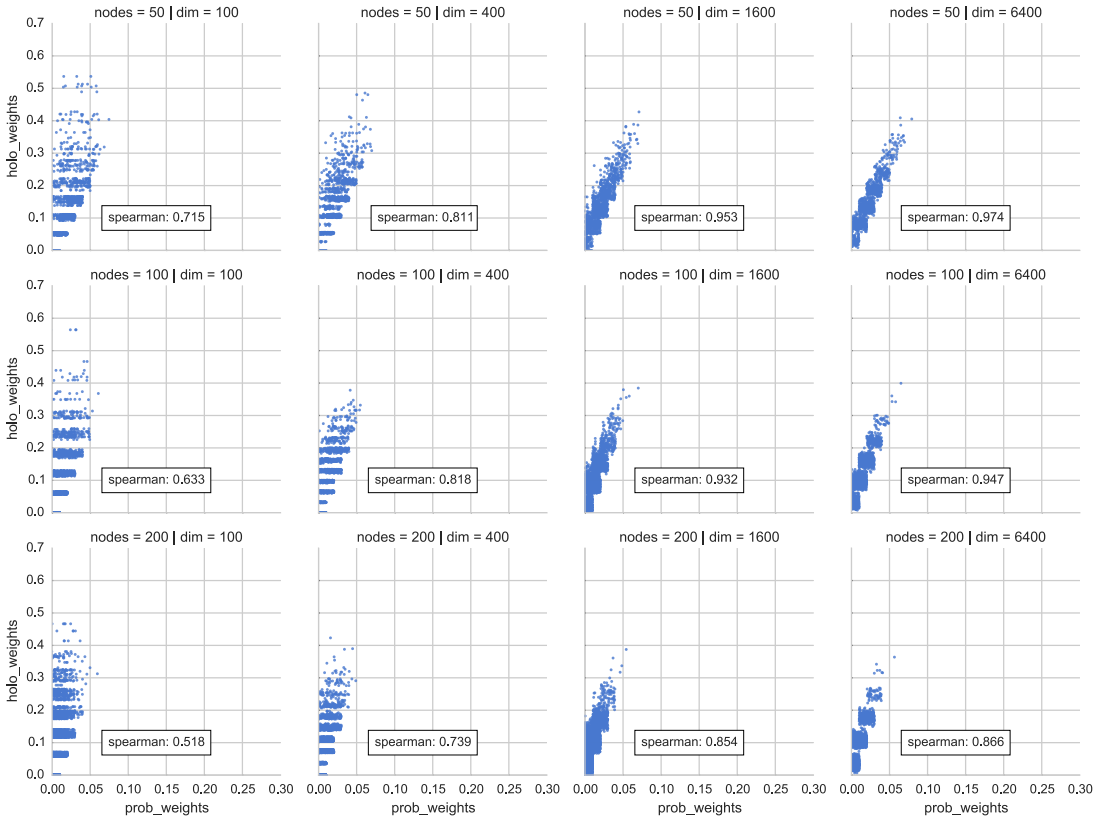

Figure 2: Correlation of sparse vector and probabilistic graph edge weights with varying number of nodes and vector dimensionality. Node count increases from top to bottom. Dimensionality increases from left to right.

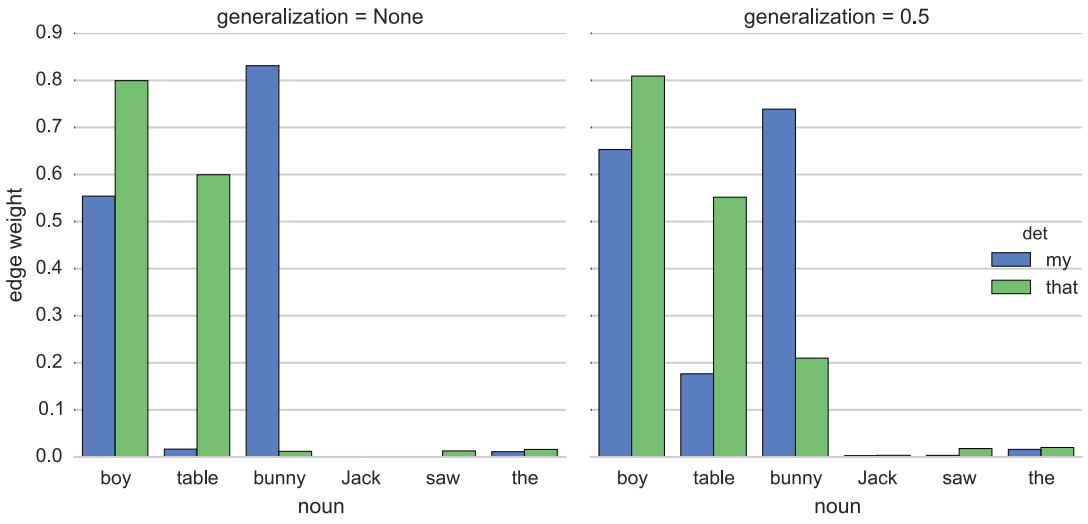

Figure 3: Generalization. A non-zero edge weight is assigned to edges that were never bumped based on the pattern of determiner-noun connections.

BOY] receives no direct training. We then measure edge weights from [THE BOY] to SAW and ATE. These will be near-zero when no composition is used. However, if the composition algorithm is successful, we expect [THE BOY] to have high edge weights to SAW and ATE. This is because previously encountered nodes composed of pairs of nodes like (THE, BOY) have high weights to SAW and ATE. As shown in Figure 4, the results match our expectations.

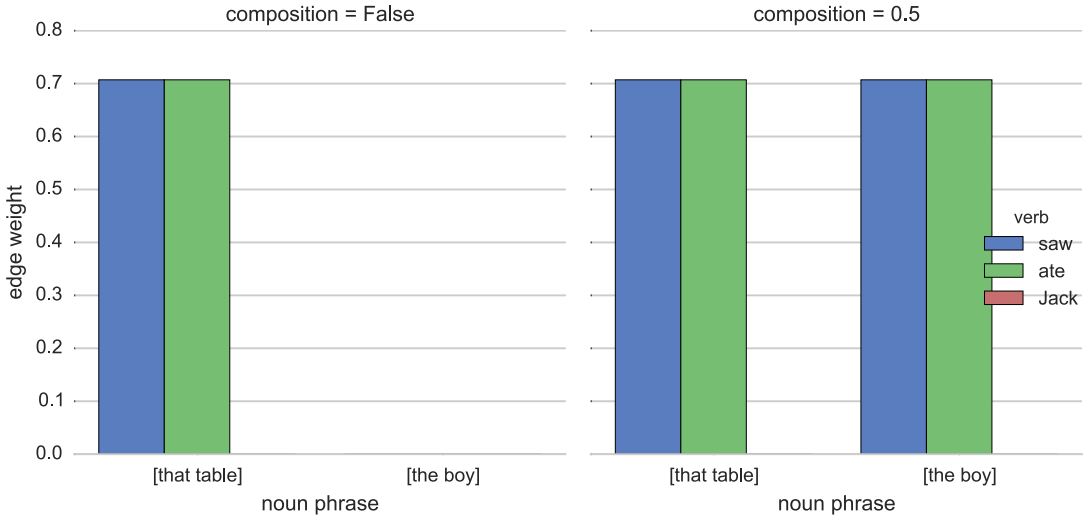

Figure 4: Composition. A newly created node "the boy" has edge-weights similar to an existing node "that table" because they are composed of similar elements.

5 Nümila

To demonstrate that a VectorGraph can be used as a component of another model, we created a simple language acquisition model, based on previous graphical models (Kolodny, Lotem, and Edelman 2015; Solan et al. 2005) and chunking models (McCauley and Christiansen 2011). Like these previous models, Nümila reduces the problem of language acquisition to the much simpler problem of producing grammatical utterances based on statistical patterns in speech, specifically the transitional probabilities between words and phrases. In reality, language acquisition is heavily dependent on the semantic and social aspects of language (Goldstein et al. 2010; Tomasello 2003), perhaps so much so that it cannot be effectively studied without considering these factors (see Frank, Goodman, and Tenenbaum 2009). Thus, we present the model mainly for illustrative purposes.

Nümila is a hybrid of ADIOS, U-MILA, and the Chunk Based Learner. It has the hierarchical representations of ADIOS, the bottom-up learning algorithm of U-MILA (roughly), and the incrementality of CBL. Nümila consists of a graphical model, a parsing algorithm, and a production algorithm.

5.1 Graphical model

The model represents its knowledge using a directed, labeled multigraph. Because the model requires only the basic operations of bumping and retrieving edge weights, any graph that implements these operations, with at least pseudo-probabalistic edge weights can be used. We test realizations of the model using a VectorGraph and a ProbGraph, as described above. Words and phrases ("chunks") are nodes, and transitional probabilities between those elements are edges. An idealized visualization of the graph is shown in Figure 5.

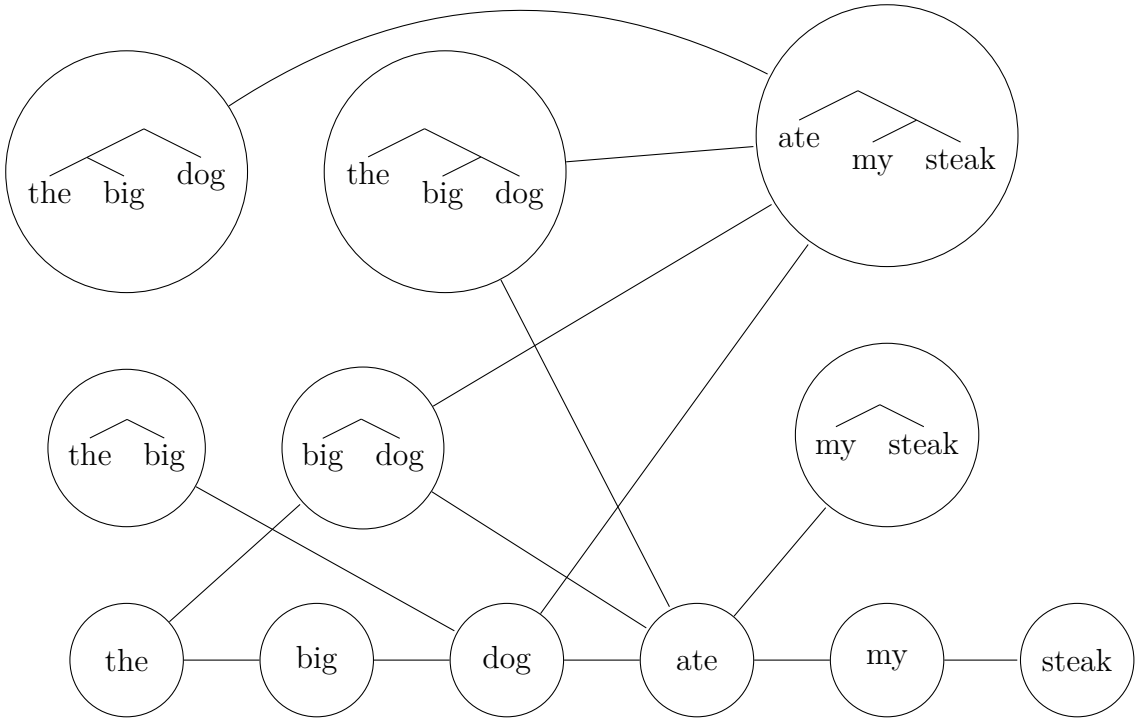

Figure 5: A simplified graph representing the utterance 'the big dog ate my steak'. For display, FTP and BTP edges are represented as a single undirected edge. Hierarchical structure is distinctive in this image; however, when the model is run in flat mode, the two nodes in the top left would be collapsed into a single node.

5.1.1 Edges

The model has two edge-types representing forward and backward transitional probabilities, that is, the probability of one word following or preceding a given word: $p(w_i = x|w_{i-1} = y)$ and $p(w_i = x|w_{i+1} = y)$ respectively. Both edge weights are used when computing the *chunkiness* of two nodes. Chunkiness measures the degree to which two words tend to occur together, and is defined as the weighted geometric mean of FTP and BTP between the two nodes. The chunkiness for the ordered pair (a, b) is

$$\text{chunkiness}(a, b) = w_{\text{FTP}}\text{weight}(a, b, \text{FTP})w_{\text{BTP}}\text{weight}(b, a, \text{BTP}) \tag{4}$$

Although forward transitional probability (FTP) is the standard in N-gram models, some evidence suggests that infants are more sensitive to BTP (Pelucchi, Hay, and Saffran 2009), and previous language acquisition models have been more successful when employing it (McCauley and Christiansen 2011). To examine the relative contribution of each direction of transitional probability, we make their relative weight an adjustable parameter.

5.1.2 Merge

When two nodes (initially words) are determined to co-occur at an unexpectedly high frequency (see below), a merge function is applied to create a new node. We consider three merge functions. The *flat* merge function takes two[3] nodes and concatenates them: [A B], [C D] → [A B C D]. The *hierchical* merge combines them into a tree: [A B], [C D] → [[A B] [C D]]. Finally, the *compositional* merge is like hierarchical merge, but additionally uses the composition algorithm discussed above to construct initial edge weights for the newly created node.

5.2 Parsing

The assignment of structure and learning occur in a single process that we call parsing. To parse an utterance, the model constructs a path through the graph, making local modifications as it goes. The initially blank graph is thus built with three operations: (1) adding newly discovered base tokens to the graph, (2) increasing weights between nodes in the graph, and (3) creating chunk nodes by merging existing nodes. In addition to modifying the graph, parsing assigns structure to the utterance, representing it as a path through the graph. The path spans the full utterance; however, because chunk nodes span multiple words, this path may have fewer nodes than the utterance has words.

The model uses a greedy algorithm to construct the path. It begins at the boundary node, and then proceeds forward one word/node at a time. Whenever a new node is added to the path, the FTP and BTP edge weights between that node and the previous node are bumped. In line with the effect of working memory on language use (Christiansen and Chater 2015), the model operates in a narrow, four-node window. When the path becomes four nodes long, the model attempts to consolidate by replacing pairs of nodes with chunks. To do this, the chunkiness (Eq. 4) between all adjacent nodes is computed; if the highest chunkiness exceeds a threshold, the pair is replaced by a single node representing that pair. If this chunk node is already in the graph, it is used; otherwise it is first created and added to the graph. Edge weights between this new node and the nodes on either side are bumped. If no pair exceeds the threshold, the oldest node in the path is removed from working memory to maintain the four-node window; it receives no additional processing.

The algorithm proceeds in this manner, adding the next word to the path and chunking, until it reaches the end of the utterance, at which point nodes are

3. The restriction to a binary merge function is a simplifying assumption, not a theoretical claim (in contrast to Chomsky 1999).

combined until the path consists of a single node, or no pair of nodes exceeds the chunkiness threshold. The final representation of the utterance is the full path, including those nodes dropped from working memory.

5.3 Simulations

To test the model, we use naturalistic child directed speech, specifically the corpora prepared by Phillips and Pearl (2014). For each of the seven languages, the input can be tokenized by word, syllable, or phoneme, giving a total of $7 \times 3 = 21$ corpora. All models are trained on the first 7000 utterances of each corpus, and tested on the next 1000. We test several instantiations of Nümila using different graph implementations, merge functions, and parameter settings.

5.3.1 Grammaticality judgment

As a first test, we use the common task of discriminating grammatical from ungrammatical utterances. This task is appealing because it is theory agnostic (unlike evaluating tree structures) and it does not require that the model produce normalized probabilities (unlike perplexity).

5.3.1.1 Generating an acceptability score To score an utterance, the model begins by parsing the utterance, discovering a path through the graph that passes through every word in the sentence (possibly visiting multiple words with one chunk node). The product of chunkinesses for every adjacent pair of nodes on the path is then calculated. Finally, to avoid penalizing longer utterances, the score is taken to the $n - 1$ root, where n is the length of the utterance.

5.3.1.2 Preparation of stimuli and analysis of performance To construct a test corpus, we first take 500 unseen utterances from the corpus, which are labeled "grammatical". For each utterance, we create a set of altered utterances, each with one adjacent pair of tokens swapped. For example, given "the big dog", we create "big the dog" and "the dog big". These altered utterances are added to the test corpus with the label "ungrammatical". The model's task is to separate grammatical from ungrammatical. Often, this task is modeled by setting a threshold, all utterances with scores higher than the threshold predicted to be grammatical. However, it is unclear how to set such a threshold without either specifying it arbitrarily, or giving the model access to the test labels. Thus, we employ a metric from signal detection theory, the Receiver Operator Characteristic.

An ROC curve, such as the one in figure Figure 6, plots true positive rate against false positive rate. As the acceptability threshold is lowered, both values increase. This curve is closely related to the precision-recall curve, but it has the benefit of allowing interpolation between data points, resulting in a smoother curve (Davis and Goadrich 2006). As a scalar metric, we use the total area under the curve. The better the separation of grammatical from ungrammatical by the acceptability score, the higher this value will be.

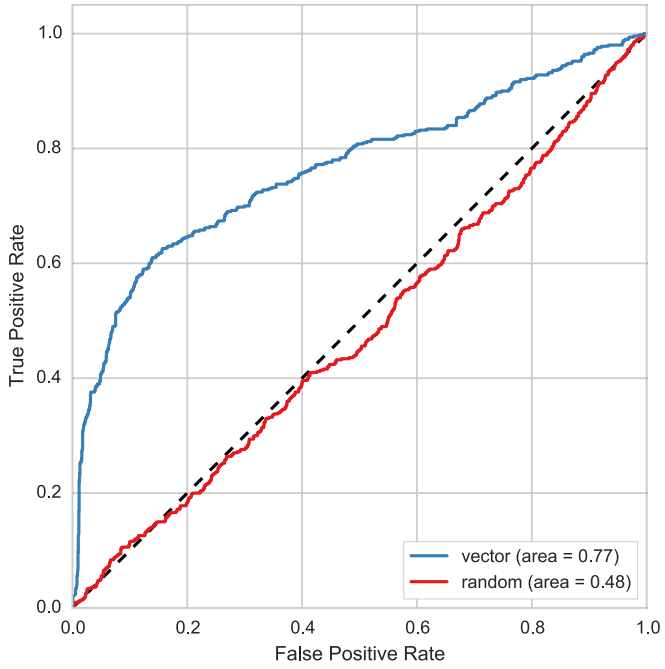

Figure 6: A Receiver Operator Characteristic curve showing the tradeoff between true- and false-positive rate on a grammaticality discrimination task. The default Nümila model and a random, dummy model are shown.

5.3.1.3 Results The results indicate that the VectorGraph can be successfully used in a language acquisition model. The models using a VectorGraph perform about as well as those using a ProbGraph, which has true transitional probabilities as edges. However, they perform slightly worse in some cases, likely due to interference between non-orthogonal index vectors (as in Figure 2). To support this explanation, we note that the Vector/Prob difference is most pronounced for words, of which there are the most unique tokens. Looking only at the phonemic input, we see that the two graphs perform more similarly.

Using multi-word chunks does not provide any benefit—in fact, it seems to be a disadvantage. The Markovian model that only tracks transitional probabilities (both FTP and BTP) between individual words does the best[4]. There is no clear difference between the hierarchical and flat merging rules, but given that chunking is not working at all, we cannot infer much from this result. However, we note that our method of producing foil utterances may be especially well-suited for a first-order Markov model. By simply swapping adjacent words, we are likely to introduce unlikely bigrams, which the Markov model will capture. Thus, we turn now to the next simulation, which may better distinguish the models.

4. We initially attempted to use SRILM bigram and trigram models as a baseline. We found that, with a variety of parameter settings, they consistently performed worse than a the FTP-only, non-chunking Nümila (a bigram model). We suspect this is due to optimizations designed to work for larger corpora.

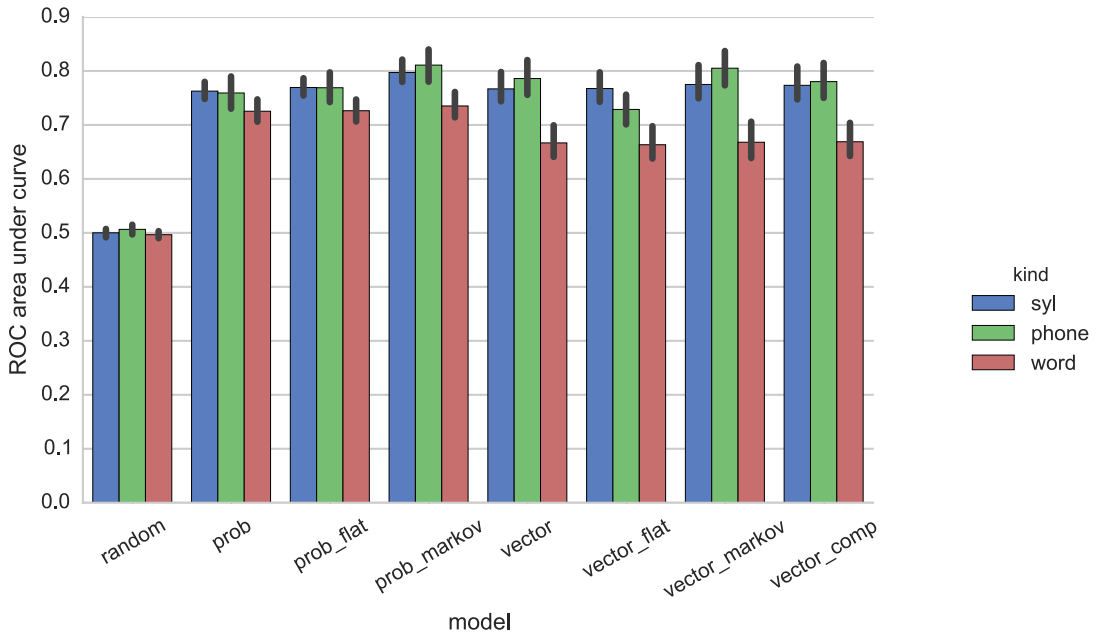

Figure 7: Grammaticality discrimination. Area under ROC curve for different input types and models, collapsed across languages. Error bars here and in all subsequent plots represent 95% confidence intervals by bootstrapping. Vector/prob refer to the graph. Flat indicates that the non-hierarchical merge rule is used. Markov indicates that chunks are not used. Comp indicates the compositional merge function is used.

5.3.2 Production

As a second test, we use the task of ordering a bag of words, a proxy for production. A more direct test of production would be to generate utterances without any input, for example, by concatenating arbitrary nodes in the graph based on transitional probabilities. However, this task has two disadvantages: (1) it is difficult to evaluate the acceptability of generated utterances without querying human subjects, and (2) speaking involves semantic as well as structural information, the first of which the present model does not attempt to capture. To avoid these problems, we follow previous work (Chang, Lieven, and Tomasello 2008; McCauley and Christiansen 2014) by using a word-ordering task to isolate structural knowledge. A bag of words is taken as an approximate representation of the thought a speaker wishes to convey; the syntactic task is to say the words in the right order.

5.3.2.1 Ordering a bag of words We treat ordering a bag of words as an optimization problem, using the acceptability score described above as a utility function. The optimal but inefficient strategy is to enumerate all possible orderings of the words and choose the one with the highest acceptability score. However, with $n!$ possible orderings, this becomes intractable for longer utterances. As with parsing, we propose a greedy algorithm, very similar to the one used by McCauley, Monaghan, and Christiansen (2015). As with parsing, production can be seen as forging a path through the graph; however in this case, the model must choose the order in which it visits the nodes.

The algorithm begins by greedily constructing chunks using the input words: The words are placed in a bag, and the most chunkable pair is replaced with their chunk node until no more chunks can be made. Because a word and chunk are both nodes, this process is applied recursively, sometimes resulting in a single chunk for the entire utterance. Next, the chunks are combined to form an utterance: Beginning from the utterance boundary node, the node in the that has the greatest chunkiness with the previous node is added to the path. This is somewhat like a Markov process, except that chunkiness is used in the place of forward transitional probability and a maximization rule is used as opposed to probability matching.

5.3.2.2 Preparation of stimuli and analysis of performance To test the model on this task, we take an unseen item from the corpus, convert it into a bag of words, and then compare the model's ordering to the original utterance. A simple comparison strategy is to assign a score of 1 if the model's output perfectly matches the original, and 0 otherwise (as in McCauley and Christiansen 2014). However, this metric selectively lowers the average score of longer utterances, which have $n!$ possible orderings. If the average score varies across utterance lengths, utterances of different lengths will have varying discrimination power (in the extreme, no discrimination power if all models fail all utterances of a given length). Given this, we use the BLEU metric (Papineni et al. 2002), which is more agnostic to utterance length. Specifically, we use the percentage of trigrams that are shared between the two utterances.[5]

5. Although we present only results with this metric, we remark that the pattern of results is roughly constant for various order N-grams, as well as for the strict perfect match metric.

5.3.2.3 Results The production results roughly parallel the grammaticality discrimination results. The VectorGraph suffers slightly from noise, and the Markovian models perform the best. In this case, the nature of the task does not explain the results. CBL outperforms trigrams on a very similar task, indicating that chunks can be useful for this task (McCauley, Monaghan, and Christiansen 2015). We remark, however, that their model only attempts to produce the utterances spoken by the child, while our model attempts to produce adult-generated utterances. If chunks play a more significant role in the linguistic knowledge of a child than that of an adult, we would predict a chunking model to perform better on child-produced utterances.

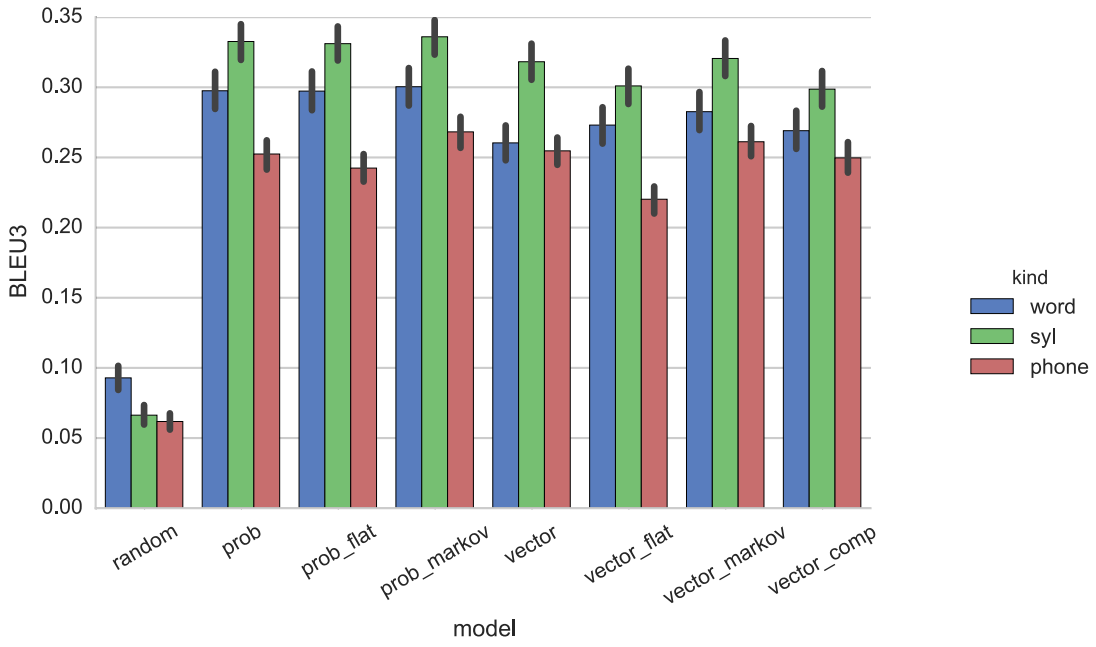

Figure 8: Production. The percentage of shared trigrams (i.e. third-order BLEU score; Papineni et al. 2002) between an adult utterance and the model's attempt to construct an utterance given an unorderd bag-of-words representation of that utterance.

To investigate the contribution of FTP and BTP, we conduct the production experiment with a second set of models. All use the ProbGraph, to eliminate possible interactions with the noise caused by the VectorGraph. We use a 2 × 2 design of (chunking, Markov) × (ftp, btp, both). Results are displayed in Figure 9. For the Markovian models, there is a clear preference for BTP, which is somewhat surprising given that the total score of a path is the same regardless of transition direction. This indicates that the FTP/BTP distinction may depend on the greedy production algorithm, which chooses the next word based on the transition from the last word. A possible explanation is that BTP prevents the algorithm from immediately concatenating a common word (which will generally have higher incoming FTP). When using BTP, the frequency of a word will not affect its probability of being chosen (at least, not in this way).

Another surprising result is the interaction between chunking and transitional

direction. The chunking models are not as sensitive to this attribute: The chunking model outperforms the Markov model when only using FTP. Given that the learning of chunks, as well as the chunking phase of production, operate entirely based on FTP, we are unsure how to explain these results. Additionally, we are unable to dissociate the effect of transitional probability direction on the learning and production phases.

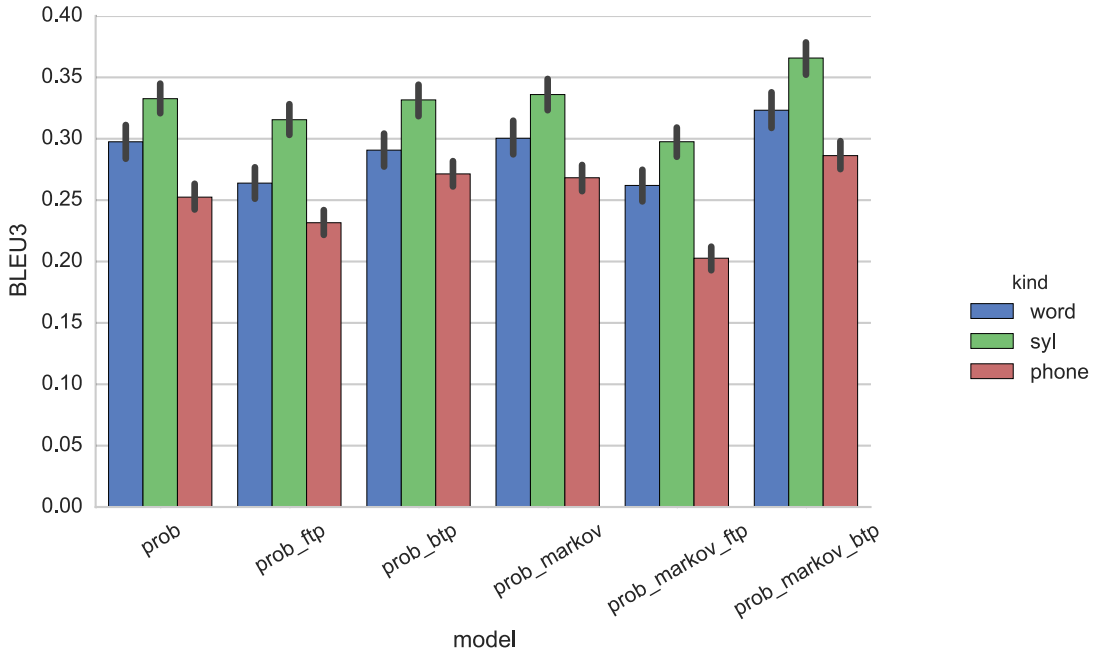

Figure 9: Relative contribution of FTP and BTP.

5.4 Discussion

The performance of Nümila is less than impressive. However there are still important lessons to be learned. First, and most relevant here, is the observation that the VectorGraph can be used effectively in language acquisition model with naturalistic input. Unfortunately, the fact that chunking does not seem to improve performance limits our ability to judge the effectiveness of the composition algorithm. The non-effect of the predictive merge function could be due to its inadequacy, or to the ineffective learning and use of chunks in general.

We may have been able to identify an effect of chunking with easier tasks. For example, we could follow McCauley, Monaghan, and Christiansen (2015) by giving the model child-produced utterances in the bag-of-words task. We could also follow Bod (2009) by using parts of speech as input. Finally, given the relatively small inventory of phonemes compared to words and syllables, we may be able to tease out an effect of chunking with a word-segmentation task.

6 Conclusion

The computational theory of mind has provided many important insights. While neuroscience explains what brains do, and classical psychology explains what people do, computational theories may serve as the best explanations for what *minds* do. However, along with new insight, these theories have brought many new methodological challenges, which remain prevalent some 40 years after the first computational psychological theories were being surfaced. One such issue, on which we have focused, is that of levels of representation. An inherent trait of a computation is multiple realizability (Turing 1950), a trait that inevitably leads to explanation at multiple levels. It is widely acknowledged at this point that these different levels each have value, but there is massive disagreement on how exactly the levels relate.

We propose that the only way to address these questions is with an earnest attempt[6] to explicitly and formally connect models at different levels of analysis. One such question that is especially interesting and controversial is that of implementation vs. approximation. Do lower level theories implement higher level theories, or do higher level theories approximate lower level theories? The first option suggests a tight fit between levels of analysis, with linking theories systematically reducing a higher level theory into smaller pieces. This is the case for digital computers: the python with which our models are implemented are directly translated to assembly, machine code, logical gates, and finally transistors. In this case, each level of description is equally true, in the sense that they make the same, accurate predictions of how the computer will behave (at least to a very good approximation). Some cognitive scientist believe that brains are similar to computers in this regard (e.g. Fodor and Pylyshyn 1988).

However, it is possible that minds are fundamentally different from computers. Under this view, functional descriptions of mind are emergent, fuzzy properties of the underlying neural computations (McClelland 2010), and it may be impossible to systematically compile these abstractions into neural processing. For example, D. J. Chalmers (1992) discusses operations on compositional, distributed representations that transform the structure holistically, arguing that these operations cannot be described in symbolic terms. This does not preclude a higher level description in non-symbolic terms. For example, *harmony* (Smolensky and Legendre 2006) is a high level description of the processing of neural networks that is not symbolic in nature.

Vector Symbolic Architectures may be a useful tool for approaching these issues. While the basic operations are, to a good approximation, implementations of symbolic operators, these could be augmented with additional operations that lack a clear symbolic interpretion. Although the simplistic generalization and composition algorithms we propose have a direct graphical interpretation, these could be replaced by holistic transformations that do not implement symbolic operations. If the functions remain modular and their purpose remains clear, we can maintain a functional description while sacrificing a direct relationship to

6. We say "attempt" because the continued failure to formally connect models may itself be indicative that cognition cannot be rigorously divided into functional and modular components.

symbols. Although symbols are powerful tools, they also limit the space of possible transformations; thus, employing such operations could lead to a more powerful system. An interesting aspect of the VectorGraph model is that the input and output from such an operation can still be understood in symbolic terms (as the edge weights to discrete nodes). However, as more of these non-symbolic operations are used, the graphical representation becomes more of an interpretation than an intrinsic trait of the system.

The relationship between levels of analysis is a major open question in the cognitive sciences. It is as of yet unclear how well the functioning of minds can be described by symbolic models, or any modular abstractions for that matter. We have suggested that graphs and Vector Symbolic Architectures are potentially profitable tools for exploring these questions. More broadly, we propose that level-spanning modeling will play an essential role in answering fundamental meta-theoretical questions in cognitive science, and in the construction of a cohesive theory of mind.

References

Anderson, John R. 1991. "The adaptive nature of human categorization." *Psychological Review* 98 (3): 409.

Anderson, John Robert. 1990. *The adaptive character of thought.* Psychology Press.

Ashby, F Gregory, and Leola A Alfonso-Reese. 1995. "Categorization as probability density estimation." *Journal of mathematical psychology* 39 (2): 216–233.

Baroni, Marco, Raffaella Bernardi, and Roberto Zamparelli. 2013. "Frege in space: A program for compositional distributional semantics." *Submitted, draft at http://clic. cimec. unitn. it/composes.*

Basile, Pierpaolo, Annalina Caputo, and Giovanni Semeraro. 2011. "Encoding syntactic dependencies by vector permutation." In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics,* 43–51. Association for Computational Linguistics.

Blouw, Peter, Eugene Solodkin, Paul Thagard, and Chris Eliasmith. 2015. "Concepts as Semantic Pointers: A Framework and Computational Model." *Cognitive science.*

Bod, Rens. 2009. "From Exemplar to Grammar: A Probabilistic Analogy-Based Model of Language Learning." *Cognitive Science* 33 (5): 752–793.

Bullmore, Edward T, and Danielle S Bassett. 2011. "Brain graphs: graphical models of the human brain connectome." *Annual review of clinical psychology* 7:113–140.

Chalmers, David. 1990. "Why Fodor and Pylyshyn were wrong: The simplest refutation." In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society, Cambridge, Mass,* 340–347.

Chalmers, David J. 1992. "Syntactic transformations on distributed representations." In *Connectionist natural language processing,* 46–55. Springer.

Chang, Franklin, Elena Lieven, and Michael Tomasello. 2008. "Automatic evaluation of syntactic learners in typologically-different languages." *Cognitive Systems Research* 9 (3): 198–213.

Chomsky, Noam. 1995. *The minimalist program.* Vol. 1765. Cambridge Univ Press.

———. 1999. *Derivation by phase.* Vol. 18. MIT, Department of Linguistics.

Christiansen, Morten H, and Nick Chater. 2015. "The Now-or-Never Bottleneck: A Fundamental Constraint on Language." *Behavioral and Brain Sciences:* 1–52.

Clark, Herbert H. 1997. "Dogmas of understanding." *Discourse Processes* 23 (3): 567–598.

Davis, Jesse, and Mark Goadrich. 2006. "The relationship between Precision-Recall and ROC curves." In *Proceedings of the 23rd international conference on Machine learning,* 233–240. ACM.

Edelman, Shimon. 2008. "On the nature of minds, or: truth and consequences." *Journal of Experimental*
Theoretical Artificial Intelligence 20 (3): 181–196.

Eliasmith, Chris. 2003. "Moving beyond metaphors: Understanding the mind for what it is." *The Journal of philosophy* 100 (10): 493–520.

———. 2013. *How to build a brain: An architecture for neurobiological cognition.*

Fodor, Jerry A. 1975. *The language of thought.* Vol. 5. Harvard University Press.

Fodor, Jerry A, and Zenon W Pylyshyn. 1988. "Connectionism and cognitive architecture: A critical analysis." *Cognition* 28 (1-2): 3–71.

Frank, Michael C, Noah D Goodman, and Joshua B Tenenbaum. 2009. "Using speakers' referential intentions to model early cross-situational word learning." *Psychological Science* 20 (5): 578–585.

Gayler, Ross. 1998. "Multiplicative binding, representation operators and analogy."

———. 2004. "Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience." *arXiv preprint cs/0412059.*

Goldstein, Michael H, Heidi R Waterfall, Arnon Lotem, Joseph Y Halpern, Jennifer A Schwade, Luca Onnis, and Shimon Edelman. 2010. "General cognitive principles for learning structure in time and space." *Trends in cognitive sciences* 14 (6): 249–258.

Goodman, Noah D, Joshua B Tenenbaum, and Tobias Gerstenberg. 2014. *Concepts in a probabilistic language of thought.* Technical report. Center for Brains, Minds and Machines (CBMM).

Griffiths, Thomas L, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. 2010. "Probabilistic models of cognition: exploring representations and inductive biases." *Trends in cognitive sciences* 14 (8): 357–364.

Griffiths, Thomas L, Mark Steyvers, and Joshua B Tenenbaum. 2007. "Topics in semantic representation." *Psychological review* 114 (2): 211.

Hagoort, Peter. 2004. "How the brain solves the binding problem for language." In *28th International Congress of Psychology.*

Hale, John T. 2011. "What a rational parser would do." *Cognitive Science* 35 (3): 399–443.

Jackendoff, Ray. 2003. "Précis of foundations of language: brain, meaning, grammar, evolution." *Behavioral and Brain Sciences* 26 (06): 651–665.

Jones, Michael N, and Douglas JK Mewhort. 2007. "Representing word meaning and order information in a composite holographic lexicon." *Psychological review* 114 (1): 1.

Kanerva, Pentti. 1988. *Sparse distributed memory.* MIT press.

———. 2009. "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors." *Cognitive Computation* 1 (2): 139–159.

Kanerva, Pentti, Jan Kristofersson, and Anders Holst. 2000. "Random indexing of text samples for latent semantic analysis." In *Proceedings of the 22nd annual conference of the cognitive science society,* vol. 1036. Citeseer.

Karlgren, Jussi, and Magnus Sahlgren. 2001. "From Words to Understanding." In *Foundations of real-world intelligence,* edited by Y Uesaka, Pentti Kanerva, and H Asoh. Stanford: CSLI Publications.

Katz, Slava M. 1987. "Estimation of probabilities from sparse data for the language model component of a speech recognizer." *Acoustics, Speech and Signal Processing, IEEE Transactions on* 35 (3): 400–401.

Kolodny, Oren, Arnon Lotem, and Shimon Edelman. 2015. "Learning a Generative Probabilistic Grammar of Experience: A Process-Level Model of Language Acquisition." *Cognitive science* 39 (2): 227–267.

Kruschke, John K. 1992. "ALCOVE: an exemplar-based connectionist model of category learning." *Psychological review* 99 (1): 22.

Landauer, Thomas K., and Susan T. Dumais. 1997. "A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge." *Psychological Review* 104 (2): 211–240.

Lund, Kevin, and Curt Burgess. 1996. "Producing high-dimensional semantic spaces from lexical co-occurrence." *Behavior Research Methods, Instruments, & Computers* 28 (2): 203–208.

Marr, David. 1982. *Vision: A computational investigation into the human representation and processing of visual information.*

McCauley, Stewart M, and Morten H Christiansen. 2011. "Learning simple statistics for language comprehension and production: The CAPPUCCINO model." In *Proceedings of the 33rd annual conference of the Cognitive Science Society,* 1619–24.

———. 2014. "Acquiring formulaic language: A computational model." *The Mental Lexicon* 9 (3): 419–436.

McCauley, Stewart M, Padraic Monaghan, and Morten H Christiansen. 2015. "Language emergence in development: A computational perspective." In *The handbook of language emergence.* Wiley-Blackwell.

McClelland, James L. 2010. "Emergence in cognitive science." *Topics in cognitive science* 2 (4): 751–770.

Nosofsky, Robert M. 1986. "Attention, similarity, and the identification–categorization relationship." *Journal of experimental psychology: General* 115 (1): 39.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. "BLEU: a method for automatic evaluation of machine translation." In *Proceedings of the 40th annual meeting on association for computational linguistics,* 311–318. Association for Computational Linguistics.

Pelucchi, Bruna, Jessica F Hay, and Jenny R Saffran. 2009. "Learning in reverse: Eight-month-old infants track backward transitional probabilities." *Cognition* 113 (2): 244–247.

Phillips, Lawrence, and Lisa Pearl. 2014. "Bayesian inference as a viable cross-linguistic word segmentation strategy: It's all about what's useful." In *Proceedings of the 36th annual conference of the cognitive science society,* 2775–2780. Citeseer.

Piantadosi, Steven Thomas. 2011. "Learning and the language of thought." PhD diss., Massachusetts Institute of Technology.

Plate, Tony, et al. 1995. "Holographic reduced representations." *Neural networks, IEEE transactions on* 6 (3): 623–641.

Pothos, Emmanuel M, and Andy J Wills. 2011. *Formal approaches in categorization.* Cambridge University Press.

Rumelhart, David E, James L McClelland, PDP Research Group, et al. 1986. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1-2." *Cambridge, MA.*

Sahlgren, Magnus. 2005. "An introduction to random indexing." In *Methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering, TKE,* vol. 5.

Sahlgren, Magnus, Anders Holst, and Pentti Kanerva. 2008. "Permutations as a means to encode order in word space."

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

Smolensky, Paul, and Géraldine Legendre. 2006. *The harmonic mind: From neural computation to optimality-theoretic grammar (Vol. 1: Cognitive architecture).* MIT Press.

Solan, Zach, David Horn, Eytan Ruppin, and Shimon Edelman. 2005. "Unsupervised learning of natural languages." *Proceedings of the National Academy of Sciences of the United States of America* 102 (33): 11629–11634.

Steedman, Mark. 2000. *The syntactic process.* Vol. 24. MIT Press.

Steedman, Mark, and Jason Baldridge. 2011. "Combinatory Categorial Grammar." In *Non- Transformational Syntax: Formal and Explicit Models of Grammar,*

edited by R. Borsley and K. Borjars, 181–224. Wiley-Blackwell, April. doi:10.1002/9781444395037.ch5. http://dx.doi.org/10.1002/9781444395037.ch5.

Tenenbaum, Joshua B, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. 2011. "How to grow a mind: Statistics, structure, and abstraction." *science* 331 (6022): 1279–1285.

Tomasello, Michael. 2003. "On the different origins of symbols and grammar." *Evolution of Language* 3:94–110.

Turing, Alan M. 1950. "Computing machinery and intelligence." *Mind* 59 (236): 433–460.

Werning, Markus, Wolfram Hinzen, and Edouard Machery. 2012. *The Oxford handbook of compositionality.* OUP Oxford.