flc37@cornell.edu

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

Fred Callaway
Cornell University

25th April 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 problem a cognitive agent and an optimal solution to that problem from a logical viewpoint (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. These representational abstractions must ultimately 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, 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 becomes 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). Generation is thus modeled as graph traversal.

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 however, 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 a unique class of edge to represent the connection from slot to filler. For example, upon finding the patterns A X C, A Y C, and A Z C, a new pattern would be created: A E1 C, where E1 has filler edges pointing to X, Y, and Z. This demonstrates how multiple edge types can be used to represent more complex models in a graph.

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. Critically, this model employs a composition function to construct 'chunks', sequences of words that are treated as a single unit. As in ADIOS and U-MILA, the function 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 (**kanerva1988**; R. Gayler 1998; Plate 1995). Like more connectionist models, (i.e. neural networks), VSA models employ distributed representations and simple linear algebra operations to transform those representations. However, unlike neural networks, computation in VSAs is performed with defined algebraic operators. In particular, they are argued to implement recursive variable binding (R. W. Gayler 2004), an ability

that neural networks have been criticized for lacking (Jackendoff 2003). 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 (R. 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.

R. W. 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 R. 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 C Eliasmith (2013).

Label and merge have widely varying and overlapping implementations. The term *bind* is sometimes used to refer to both types of operations, although a single binding operation such as circular convolution is a only one possible implementation of *merge*. Label is the simpler of the two: it always takes two vectors, one of which only serves as a label (e.g. a variable name), and the other representing a more contentful item (e.g. a word). It is generally implemented as a single multiplicative or permutational operation.

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 (label) 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 $\circledast$ ϕ $\circledast$ 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 (Chris 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 (C 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 more complex (and hence powerful) structures and operations can be supported by neural spiking models.

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 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 a 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 method that has been used effectively in distributional models: *random indexing* (see Sahlgren 2005, for an accessible 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, the rows for similar words will be somewhat similar.

With the size of modern data sets, however, 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 unchanging *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

Here, we generalize Kanerva's technique to represent any graph. A standard representation of a graph is an adjacency matrix, $M_{N\times N}$, where each row represents the outgoing edges of one 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 id_y to row_x.

To recover an edge weight $x \to y$, we take the cosine similarity of id_y and row_x. Intuitively, this value will be higher if id_y has been added to row_x more times, that is, if the edge $x \to y$ has been bumped more times. We can also use cosine similarity to measure similarity between nodes. 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.

An interesting attribute of this representation is that edge weights behave somewhat like probabilities. That is, increasing the weight from x to y will slightly decrease the weight from x to z. Visually, adding id_y to row_x pulls row_x towards id_y, and thus away from id_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 Figure 1.

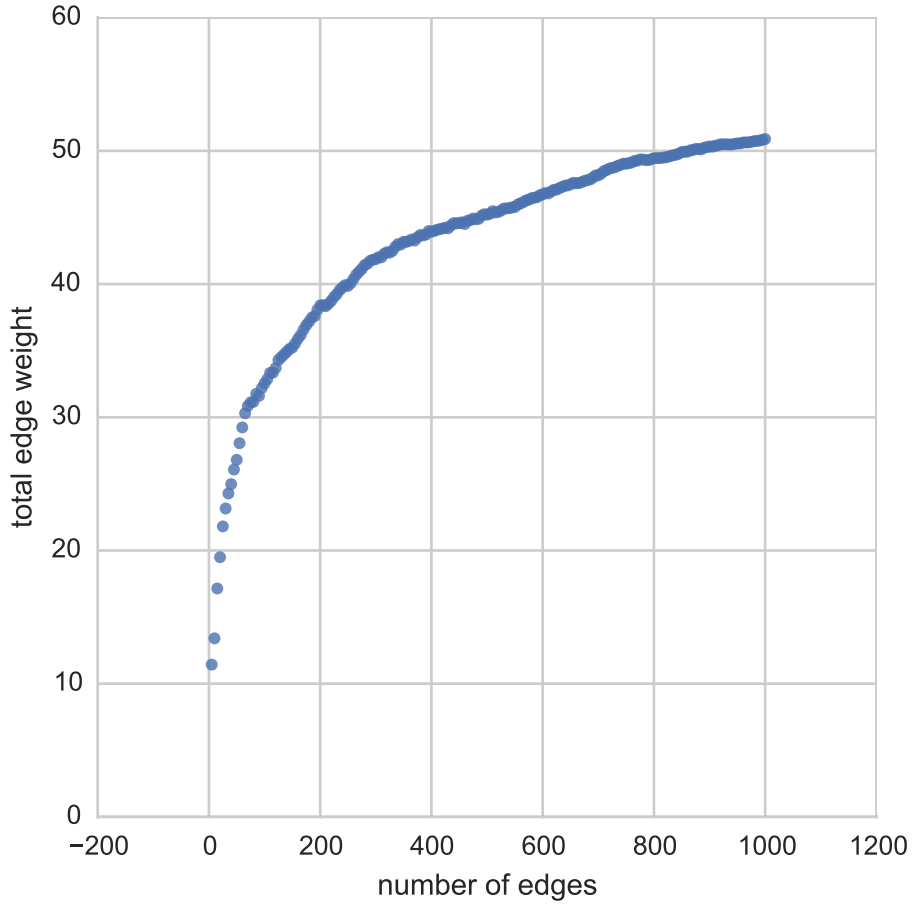

Figure 1: Total edge weight as a function of number of edges. Vectors are length 500.

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. To increase the edge from x to y with label e, xey, we add $\Pi_e id_y$ to row_x. Similarly, to recover a labeled edge weight, xey, we use $cosine(\Pi_e id_y), row_x)$.

- $bump(x, y, e) : row_x += \Pi_e id_y$
- $weight(x, ye) : cosine(row_x, \Pi_e id_y)$
- $sim(x, y) : cosine(row_x, row_y)$

TODO: This will be a table

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. Perhaps due to the impact of language on thought (at least of the academic variety), this sort of explicit categorization has been assumed to underlie generalization. Indeed, 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 discuss 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 representations. 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 to generalize based on for a given property of a given 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. Artificial neural networks are themselves graphs, many probabilistic models of categorization employ graphical representations (Tenenbaum et al. 2011), and previous graphical language models

(Kolodny, Lotem and Edelman 2015; Solan et al. 2005) have represented syntactic categories as a special class of node with edges pointing to category members. However, in addition to these extreme ends of the spectrum, the HoloGraph 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 can pull towards each other, making them 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. Intuitively, we can think of vectors as having gravity: As one vector gets close to another, it will be pulled even closer.

Formally, for a node n_0, we create a generalized row vector as the sum of the all other nodes' row vectors weighted by the similarity of each node to n_0. The generalized row vector for n_0 is thus

$$\sum sim(n_0, n_i) row_i$$

A methodological problem arises with this algorithm. It requires performing an expensive similarity calculation for every node in the graph. The brain is a massively parallel computer, and can likely preform such operations fairly quickly. However, on a digital computer, the algorithm is prohibitively time-intensive when applied to anything but a toy grammar. Thus, we propose an alternative algorithm inspired by dynamic programming that we call *dynamic association*. The basic principle of dynamic programming is to avoid computing the same thing multiple times, instead computing it just once and storing its solution. Every time we compute the generalized row vector for a given node, we will recalculate all similarities, most of which will have changed minimally. The the naive dynamic programming approach of simply caching each pair wise similarity will not do because this will ignore any information later acquired. Thus, we must go one step deeper into the algorithm to search for an efficient but well defined optimization.

Specifically, we observe that the similarity of two nodes is built up iteratively through a series of edge weight adjustments. By intervening at each of these steps, we can incrementally construct a generalized vector for each node. To do this,

each node receives two additional vectors: a *dynamic row vector* and a *dynamic id vector*. The dynamic row vector approximates the generalized row vector as defined above. This vector is the sum of the dynamic id vectors of the nodes it points to. The dynamic id vector is the sum of the (non-dynamic) row vectors of every node that has pointed to it. The effect is that when an edge $a \rightarrow b$ is increased, a's dynamic row vector becomes more similar to every other node that points to b. This is accomplished by performing two additional operations each time the edge weight from a to b is increased:

2. Add the dynamic id vector of b to the dynamic row vector vector of a.
3. Add the dynamic row vector of a to the dynamic id vector of b.

In the present implementation, we separate the dynamic generalization vector from the standard row vector, maintaining separate generalized and ungeneralized representations. However, this is not necessary. Indeed, we see it as unlikely that brains would do such a thing given the high metabolic cost of neurons.

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 individual agent's 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 $\cap$ 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) 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 (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. Baroni et al. present a recursive supervised learning alogirithm which we do not describe here. One important, and perhaps problematic, feature of this approach is that different words have different shapes (i.e. they are different order tensors). In the model of Baroni et al., the order is determined by an external syntactic model. The task of learning which words should be which order however may be very challenging.

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 cateogory label with a weighted average of all nodes. That is, upon creating of the new node [A B], we construct an initial row vector as the sum of every other chunk's row vector, weighted by the similarity of the constituents. To capture the fact that all constituents must be similar, we take the geometric mean (a multiplicative operation) of the pairwise similarities.

$$row_{ab} = \sum \sqrt{sim(a,b)sim(x,y)}row_{xy}$$

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 HoloGraph to a graph with true probabilities as edges, a ProbGraph. (Recall that HoloGraph 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 HoloGraph and a ProbGraph with the same random training data. If the HoloGraph 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 HoloGraph weights will be subject to some noise. We expect that the effect of noise will be greater for lower dimensionality vectors, and for more total nodes. As shown in figure Figure 2, the results match our expectations.

4.5.2 Generalization

To test the generalization algorithm, we create a bigram model with a HoloGraph. 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

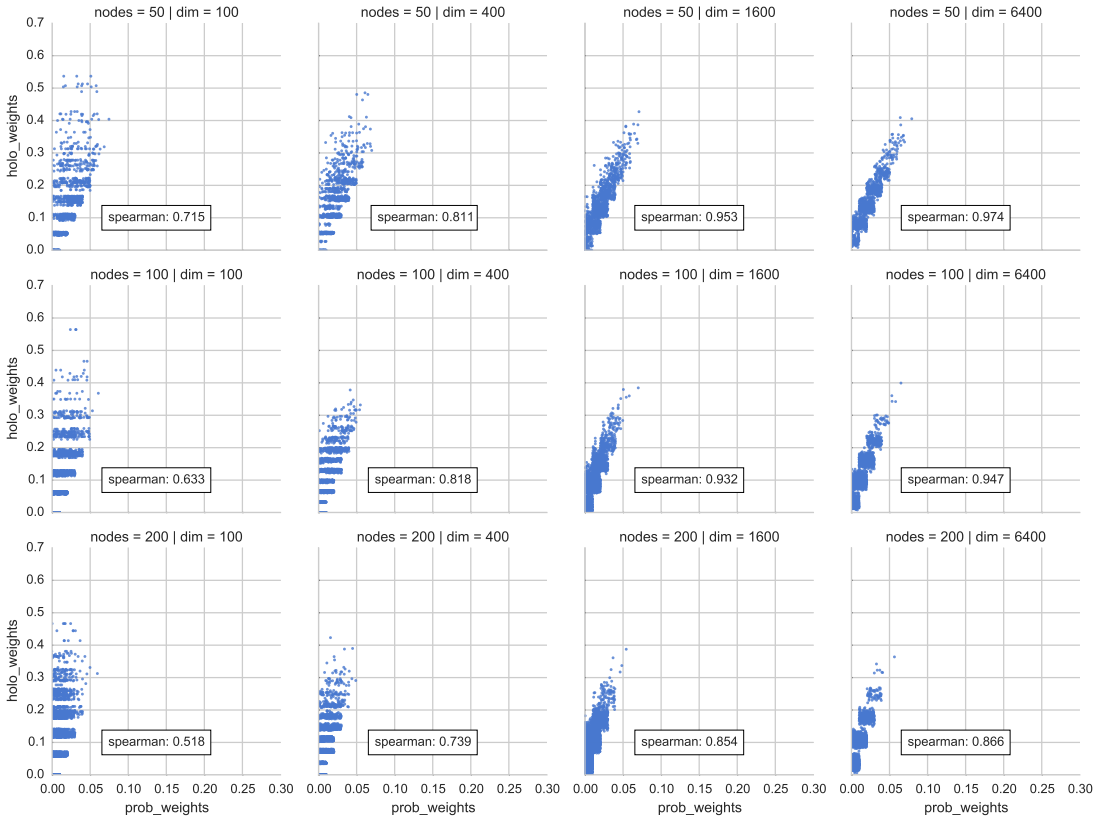

Figure 2: Correlation of sparse vector and probabilistic graph edge weights with varying number of nodes and vector dimensionality. Node count increases from top to bottom. Dimensionality increases from left to right.

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 Figure 3, both generalization algorithms are successful.

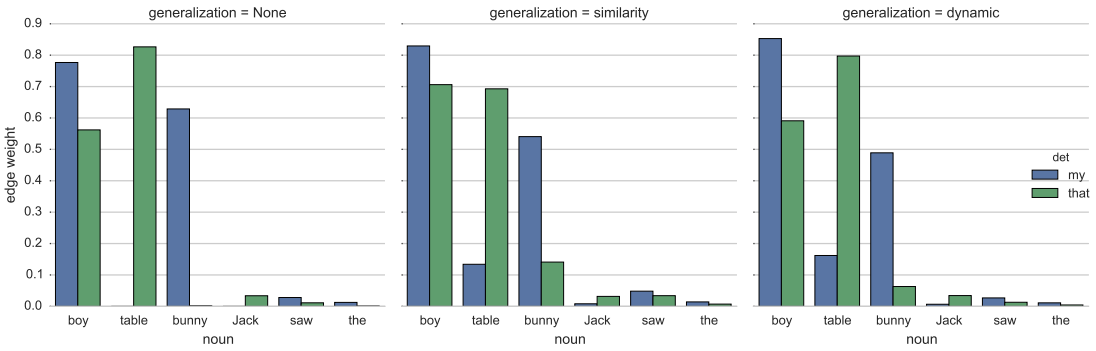

Figure 3: Generalization. A non-zero edge weight is assigned to edges that were never bumped.

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 THE and the noun BOY. For each of these phrases, 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 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 showin in figure Figure 4, the results match our expectations.

5 Nümila

This section will be dramatically shortened. I'm not confident in the validity of most of the results, so I will frame it even more as a demonstration of the graph, and not as a language model. The paper will be more cohesive this way.

Hopefully we can fix up the language part for UBL.

- bottom up chunking of U-MILA
  - no pseudo code
  - no multiple algorithms

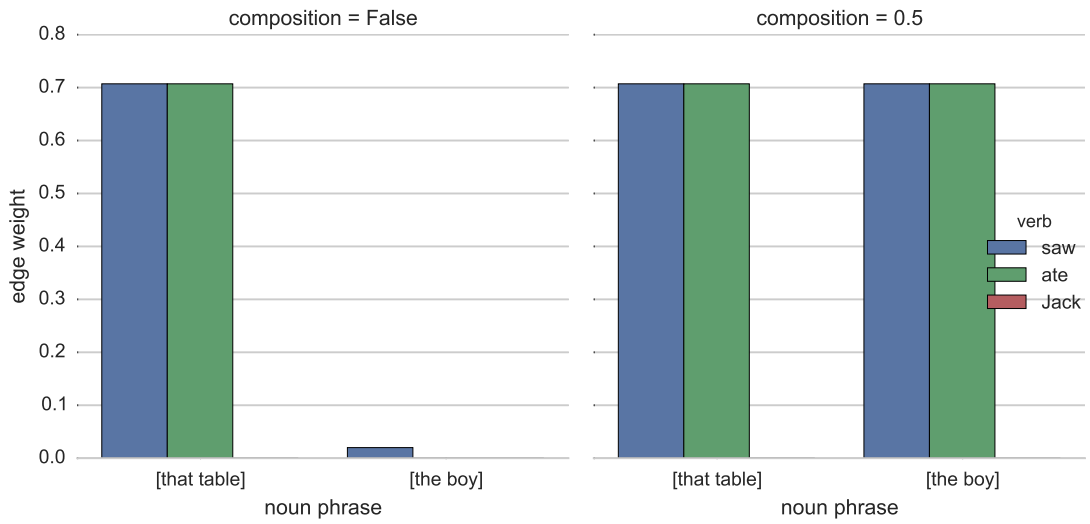

Figure 4: Composition. A newly created node has edge-weights similar to existing nodes composed of similar elements.

- ROC
  - graph of holo vs prob, with and without chunks
- BLEU
  - holo vs prob, with and without chunks
  - graph comparing FTP to BTP

6 Conclusion

Grahps, vectors, graphs, graphs, vectors, neurons, language, chalmers, and graphs.

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.*

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.

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

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.

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

Eliasmith, C. 2013. *How to build a brain: An architecture for neurobiological cognition.*

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

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

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

Gayler, Ross W. 2004. 'Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience'. *arXiv preprint cs/0412059.*

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

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

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.

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.

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