To appear in *Proceedings of the 7th Conference on Artificial General Intelligence*

Distributed Vector Representations of Words in the Sigma Cognitive Architecture

Volkan Ustun[1], Paul S. Rosenbloom[1,2], Kenji Sagae[1,2], Abram Demski[1,2]

[1] Institute for Creative Technologies, [2] Department of Computer Science
University of Southern California, Los Angeles, CA USA

Abstract. Recently reported results with distributed-vector word representations in natural language processing make them appealing for incorporation into a general cognitive architecture like Sigma. This paper describes a new algorithm for learning such word representations from large, shallow information resources, and how this algorithm can be implemented via small modifications to Sigma. The effectiveness and speed of the algorithm are evaluated via a comparison of an external simulation of it with state-of-the-art algorithms. The results from more limited experiments with Sigma are also promising, but more work is required for it to reach the effectiveness and speed of the simulation.

1 Introduction

Distributed vector representations facilitate learning word meanings from large collections of unstructured text. Each word is learned as a distinct pattern of continuous (or discrete or Boolean) values over a single large vector, with similarity among word meanings emergent in terms of distances in the resulting vector space. Vector representations are leveraged in cognitive science to model semantic memory [13,22]. They have also been used for many years in neural language models [18], where they have yielded good performance [1,2], but have not scaled well to large datasets or vocabularies. Recently, however, scalable methods for training neural language models have been proposed [10,11], handling very large datasets (up to 6 billion words) and achieving good performance on a range of word similarity tests.

The premise of distributed vector representations might have interesting repercussions for cognitive architectures as well since these architectures have naturally involved memory models and language tasks. Yet, vector representations are rare in cognitive architectures, limited to experiments with a separate module in ACT-R [20] and an effort in progress to incorporate them into LIDA to yield Vector LIDA [4].

Sigma – briefly introduced in Section 5 – is being built as a computational model of general intelligence that is based on a hybrid (discrete+continuous) mixed (symbolic+probabilistic) cognitive architecture of the same name [15]. Its development is driven by a trio of desiderata: (1) *grand unification*, uniting the requisite cognitive and non-cognitive aspects of embodied intelligent behavior; (2) *functional elegance*, yielding broad cognitive (and sub-cognitive) functionality from a simple and theoretically elegant base; and (3) *sufficient efficiency*, executing rapidly enough for antici-

pated applications. The potential utility of distributed vector representations suggests it is worth considering what they might bring to Sigma. At the same time, Sigma's approach to achieving functional elegance via a *graphical architecture* – built from graphical models [7] (in particular, factor graphs and the summary product algorithm [8]), n-dimensional piecewise linear functions [14], and gradient descent learning [16] – that sits below the cognitive architecture and implements it, suggests that it might be possible to support distributed vector representations with only small extensions to the architecture.

In other words, the goal of this paper is to evaluate whether Sigma provides a functionally elegant path towards a deep and effective integration of distributed vector representations into a cognitive architecture. The *Distributed Vector Representation in Sigma* (DVRS) model – Section 3 – is inspired primarily by BEAGLE [6], but with adaptations intended to leverage as much as possible of Sigma's existing capabilities. Section 2 provides background on BEAGLE (and on Vector LIDA, which builds on BEAGLE's approach while striving for increased efficiency).

Because Sigma is not yet completely up to implementing the full DVRS model, results are presented from two approximations to it. DVRS' is a simulation of DVRS outside of Sigma that simplifies Sigma's use of gradient descent in learning word meanings. This yields an efficient approximation to DVRS that enables large-scale experimentation. DVRS$^+$ is a partial implementation of DVRS within Sigma. It enables verifying that the core ideas work within Sigma, while requiring little modification to it, but it is incomplete and presently too slow for large-scale experimentation.

The results reported here from DVRS' (Section 4) and DVRS$^+$ (Section 6) show the potential of the DVRS algorithm itself and its incorporation into Sigma, but significant work remains for a complete and efficient implementation of DVRS in Sigma. This necessary further work is discussed in Section 7, along with the conclusion.

2 Background

BEAGLE builds a holographic lexicon − represented by distributed vectors − that captures word meanings from unsupervised experience with natural language [6]. Two types of information are utilized to learn the meaning of a word: (1) *context*, as defined by the words that co-occur in the same sentence, and (2) *word order*, as defined by the relative positions of the nearest co-occurring words.

BEAGLE assigns two vectors to each word in the vocabulary: (1) an *environmental vector* and (2) a *lexical (meaning) vector*. The word's environmental vector is ultimately intended to represent the physical characteristics of the word, such as orthography, phonology etc. and hence, should not change over time. However, in the basic model focused on here, this lower-level similarity is not included, and instead each environmental vector is simply a fixed random vector. The word's lexical vector, on the other hand, represents the memory for contexts and positions. Each time a word is encountered, its lexical vector is updated from the associated context and order information. BEAGLE uses superposition for context information – simply the sum of the environmental vectors of all of the co-occurring words. Positional infor-

mation is captured via n-grams, by *binding* together via convolution all of the words in each n-gram (of size up to 5 in [6]). The order information for a word in a sentence is the sum of all of the n-gram convolutions containing it. The word's lexical vector is then updated by adding in the context and ordering vectors.

BEAGLE uses *circular convolution* for binding to avoid the problem of *expanding dimensionality*, where the output of binding is larger than its inputs, making further binding either infeasible or intractable [12]. It furthermore uses a directed variant of circular convolution – where different inputs are permuted distinctly prior to convolution – so as to avoid losing information about their relative ordering during the otherwise symmetric convolution operation.

The approach taken in Vector LIDA [21] is similar to that in BEAGLE. The Modular Composite Representation (MCR) used in Vector LIDA also relies on capturing context and order information in high dimensional vectors, but it uses integer rather than real-valued vectors and replaces the expensive circular convolution operation – which is $O(n \log n)$ if FFTs are used [3] and $O(n^2)$ otherwise – with a faster *modular sum* operation that still avoids the expanding dimensionality problem.

3 The DVRS Model(s)

DVRS is conceptually similar to BEAGLE and MCR, but it retains BEAGLE's real-valued vectors while substituting a different fast binding operation based on pointwise product with random positional vectors. This approach still avoids the expanding dimensionality problem, but is more aligned with how Sigma works. Real-valued vectors are a natural special case of Sigma's pervasive usage of n-dimensional piecewise-linear functions; they just restrict the function to piecewise constant values over domains that are one dimensional and discrete.

The binding needed for ordering information is achieved by pointwise multiplying the environmental vector of each nearby co-occurring word with a random vector that is uniquely characteristic of the relative position between that word and the word being learned. The capture of word order information in DVRS thus maps onto skip-grams [5], which are generalizations of n-grams that allow intervening words to be skipped – a skip distance of k allows k or fewer words to be skipped in constructing n-grams. The current DVRS model employs 3-skip-bigrams, in which pairs of words are learned with at most three words skipped between them. As a result, there can be as many as 8 such skip-grams per word in a sentence. In recent work, a similar formulation was used in calculating the predicted representation of a word [11], with the excellent results reported there serving as encouragement for the potential of DVRS.

A more formal description of DVRS is as follows. Let's assume that: each sentence has n words, $l(i)$ is the lexical vector of the i^{th} word in the sentence, and $e(i)$ is the environmental vector of the i^{th} word. Each element of each environmental vector is randomly selected from the continuous span [-1,1). If the word being updated is the k^{th} word ($word_k$) in the sentence, then the context information, $c(k)$, for it is the sum of the environmental vectors for the other n-1 words in the sentence.

$$c(k) = \sum_{i=1}^{n} e(i), where\ i \neq k \qquad (1)$$

The *sequence vectors* are random vectors created for the binding operation, and like the environmental vectors are defined based on random selections from the continuous span [-1,1). *s(j)* is unique for each relative position *j* from word$_k$. The word order information, *o(k)*, is then calculated as follows (".*" is the pointwise vector multiplication operation):

$$o(k) = \sum_{j=-4}^{4} s(j) .* e(k+j), where\ j \neq 0\ \ and\ 0 < (k+j) \leq n \qquad (2)$$

DVRS uses gradient descent, where the gradient is based on the sum of the normalized context and order vectors – $\widehat{c(k)} + \widehat{o(k)}$ – to incrementally update the lexical vectors, *l(k)*, as new training sentences are processed.

The key difference between DVRS$^+$ and DVRS is that the former selects values in environmental and sequence vectors randomly from [0,1) rather than [-1,1). Sigma was originally designed to operate only with non-negative functional values because its general implementation of the summary product algorithm, which subsumes both the sum-product and max-product variants, is only guaranteed to produce correct outputs given non-negative inputs. Distributed vector computations only depend on sum-product, not on max-product, and sum-product – both in general and in Sigma – does work with negative values. However, other aspects of Sigma – such as its gradient-descent learning algorithm – also only work with non-negative values, so the use of negative values for distributed vectors has been put off to future work. This does limit the vectors in DVRS$^+$ to one quadrant of the full vector space, and as seen in Section 6, leads to somewhat degraded performance.

The key difference between DVRS' and DVRS is that the former uses a lexical vector update operation that is similar to that in BEAGLE – the lexical vector *l(k)* of word$_k$ is modified by simply adding in the normalized sum of *c(k)* and *o(k)*:

$$l(k) = l(k) + \widehat{c(k)} + \widehat{o(k)} \qquad (3)$$

By implementing DVRS' outside of Sigma, very large training datasets can be processed quickly and compared with state-of-the-art models to provide valuable insight into the overall effectiveness of DVRS.

4 Evaluating the DVRS' Simulation

The goals of this evaluation are to: (1) assess the effectiveness of DVRS'; (2) determine its robustness over random initializations of the evaluation and sequence vectors; and (3) evaluate whether replacing BEAGLE's use of expensive directed circular convolution by cheap pointwise products degrades performance. Training is performed over a corpus of ~500k sentences (12.6M words and 213K distinct words) extracted from the first 10^8 bytes of the English Wikipedia dump from March 3, 2006, provided by [9]. The text was preprocessed to contain only lowercase characters and spaces, using the script in [9]. Stop words are ignored in creating the context infor-

mation. An iMac 12,2 with 8GB RAM and a 3.4Ghz I7-2600 processor is used in the training.

One way to compare the quality of different word vectors is by examining their similarity to their closest neighbors. Table 1 depicts the top 5 neighbors of the words *language*, *film*, *business*, and *run*, ordered by vector cosine distances. Three forms of training are explored: only on context, only on ordering, and on their composite. Similar to the assessments in [6], composite training better captures word similarities for the examples shown here.

Table 1. Five nearest neighbors of four words in context, order, and composite spaces.

	language			***film***	
Context	**Order**	**Composite**	**Context**	**Order**	**Composite**
spoken	cycle	languages	director	movie	movie
languages	society	vocabulary	directed	german	documentary
speakers	islands	dialect	starring	standard	studio
linguistic	industry	dialects	films	game	films
speak	era	syntax	movie	french	movies
	business			***run***	
Context	**Order**	**Composite**	**Context**	**Order**	**Composite**
businesses	data	commercial	home	play	runs
profits	computer	public	runs	hit	running
commercial	glass	financial	running	pass	hit
company	color	private	hit	die	break
including	space	social	time	break	play

Mikolov *et al*. [10] argue that a more complex similarity test is more appropriate for the assessment of the quality of the trained word vectors. They propose for this a general word analogy test along with a specific set of test instances – termed the *Google test data* in the remainder of this article. This test simply asks, for example, the question "What is the word that is similar to *small* in the same way that *biggest* is similar to *big*?". Such questions can be answered by performing simple algebraic operations over the vector representations. To find the word that is most similar to *small* in the same way that *biggest* is similar to *big*, one can merely compute the vector $V = (l_{biggest} - l_{big}) + l_{small}$ and determine which word's lexical vector is closest to V according to cosine distance. In other words, what is added to the representation of *big* to get *biggest* should also yield *smallest* if added to the representation of *small*.[1] The Google test data includes 8,869 semantic test instances (such as determining which word is most similar to *king* in the way *wife* is similar to *husband*) and 10,675 syntactic test instances (such as determining which word is most similar to *lucky* in the way *happy* is similar to *happily*).

Table 2 shows the accuracy of DVRS' over various configurations of system settings on the Google test data. The vocabulary of the training data includes all four words in the Google test data instances for 8,185 of the 8,869 semantic test cases and 10,545 of the 10,675 syntactic test cases. Mikolov *et al*. [10] report an accuracy of

[1] As pointed out in [11], the words most similar to V in this case will actually be *biggest* and *small*, so the search results should exclude them off the top.

24% for their CBOW model with a training set of 24M words and a vector dimensionality of 600. Mnih and Kavukcuoglu [11] report an accuracy of 17.5% for a model trained on the 47M words of the Gutenberg dataset. The DVRS' co-occurrence model achieves a comparable result, 24.3%, with approximately 12.6M words in the training data and a vector dimensionality of 1024. Adding ordering information (via skip-grams) didn't improve the accuracy of the co-occurrence models in the cases tested. Increasing the vector size above 1536 also did not improve the accuracy. Overall, these results are comparable to the recently reported accuracies by [10] and [11] for comparable sizes of training data.

Table 2. Performance (% correct) on the Google test data for test instances in which all four words are in the vocabulary (and in paranthesis for all test instances).

	Vector size	Semantic	Syntactic	Overall
Co-occurrence only	1024	33.7 (31.1)	18.8 (18.6)	25.3 (24.3)
3-Skip-Bigram only	1024	2.7 (2.5)	5.0 (4.9)	4.0 (3.8)
3-Skip-bigram composite	512	29.8 (27.5)	18.5 (18.3)	23.4 (22.4)
3-Skip-bigram composite	1024	32.7 (30.2)	19.2 (18.9)	25.1 (24.0)
3-Skip-bigram composite	1536	34.6 (31.9)	20.1 (19.9)	26.4 (25.3)
3-Skip-bigram composite	2048	34.3 (31.7)	20.1 (19.9)	26.3 (25.2)

Robustness across different random initializations of environmental vectors has also been assessed for DVRS'. The model was run 5 times with different initializations of environmental vectors of size 1024, and performance was measured over a randomly selected subset (~10%) of the Google test data for composite training. The performance (% correct) was in the range [23.8, 25.0] for the best match and in the range [37.0, 37.5] when checked for a match within the 5 closest words, demonstrating the negligible effect of random initializations.

The impact of using pointwise vector multiplication instead of circular convolution has also been assessed in DVRS', with vectors of size 512. The comparison isn't directly with BEAGLE, but with a version of DVRS' in which pointwise vector multiplication is replaced with circular convolution. The achieved accuracies on the Google test data were 23.4% for pointwise multiplication and 19.9% for circular convolution; implying that, at least for this case, pointwise multiplication does not degrade performance, and in fact enhances it instead. Furthermore, training on the full training set takes a bit more than 3 hours with pointwise multiplication, but 4.5 days for circular convolution. An $O(n^2)$ variant of circular convolution is used here, but even an optimal $O(n \log n)$ implementation would be dominated by the $O(n)$ time required with pointwise multiplication. Training DVRS' on just ordering information occurs at ~1.4M words/minute with a vector dimensionality of 100, a rate that is comparable to that reported for similar configurations in [11].

5 Sigma

The Sigma cognitive architecture provides a language of *predicates* and *conditionals*. A predicate is defined via a name and a set of typed arguments, with *working memory*

containing predicate instantiations that embody the state of the system. The argument types may vary in extent and may be discrete – either symbolic or numeric – or continuous. Conditionals are defined via a set of predicate patterns and an optional function over pattern variables, providing a deep combination of rule systems and probabilistic networks. *Conditions* and *actions* are predicate patterns that behave like the respective parts of rules, pushing information in one direction from the conditions to the actions. *Condacts* are predicate patterns that support the bidirectional processing that is key to probabilistic reasoning, partial matching, constraint satisfaction and signal processing. Functions encode relationships among variables, such as joint or conditional probability distributions, although the dimensions of the functions may in general be continuous, discrete or symbolic, and the values of the functions may be arbitrary non-negative numbers (which can be limited to [0,1] for probabilities and to 0 (*false*) and 1 (*true*) for symbols).

Sigma's graphical architecture sits below its cognitive architecture, and serves to implement it. The graphical architecture is based on *factor graphs* and the *summary-product algorithm* [8], plus a function/message representation based on *n-dimensional piecewise-linear functions* [14]. Factor graphs are a general form of undirected graphical model composed of variable and factor nodes. Factor nodes embody functions – including all of those defined in the cognitive architecture plus others only relevant within the graphical architecture – and are linked to the variable nodes with which they share variables. In its simplest form, there would only be one variable per variable node, but Sigma supports variable nodes that represent sets of variables. A factor graph (Figure 1) implicitly represents the function defined by multiplying together the functions in its factor nodes. Or, equivalently, a factor graph decomposes a single complex multivariate function into a product of simpler factors.

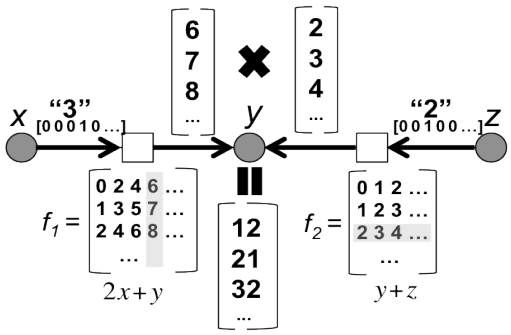

Fig 1. Factor graph for algebraic function:
$f(x,y,z) = y^2+yz+2yx+2xz = (2x+y)(y+z) = f_1(x,y)f_2(y,z)$.

The summary product algorithm computes messages at nodes and passes them along links to neighboring nodes. A message along a link represents a function over the variables in the link's variable node. Given that a variable node may embody multiple variables, functions in messages are defined in general over the cross product of their variables' domains. An output message along a link from a variable node is simply the (pointwise) product of the input messages arriving along its other links. An output message along a link from a factor node is computed by multiplying the node's function times the product of its incoming messages, and then summarizing out all of its variables that are not included in the target variable node, either by integrating the variables out to yield marginals or maximizing them out to yield maximum a posteriori (MAP) estimates.

Working memory compiles into a sector of the graphical architecture's factor graph, with conditionals compiling into more complex sectors. Functions are represented in an *n*-dimensional piecewise-linear manner and stored in factor nodes within the overall graph. Memory access in the cognitive architecture then maps onto message passing within this factor graph. As messages are sent, they are saved on links. If a new message is generated along a link, it is sent only if it is significantly different from the one already stored there. Message passing reaches *quiescence* – and thus memory access terminates – when no new messages are available to send. Once quiescence is reached, both decisions and learning occur locally – via function modification – at the appropriate factor nodes based on the messages they have received via Sigma's summary product algorithm. Based on ideas in [19] for Bayesian networks, a message into a factor node for a conditional function can be seen as providing feedback to that node from the rest of the graph that induces a local gradient for learning. Although Sigma uses undirected rather than directed graphs, the directionality found in Bayesian networks can be found at factor nodes when some of the variables are distinguished as the *children* that are conditionally dependent on the other *parent* variables. The original batch algorithm is modified to learn incrementally (online) from each message as it arrives [16].

6 The DVRS$^+$ Sigma Model

Context and *word order* information are captured by two similar predicates: (a) `Context-Vector(distributed:environment)` and (b) `Ordering-Vector(distributed:environment)`, using the discrete type `environment` – with a range equal to the vector dimensionality – for the `distributed` argument. By default, all types are continuous in Sigma, but discrete types fragment the number line into unit-length regions, each with a constant function that can take on any non-negative value. It should be clear how this directly yields the real-valued vectors needed here.

The predicate `Skip-Gram-Vector(position:position distributed:environment)` introduces a second argument, `position`, for the relative position from the word whose lexical vector is being updated. With a sufficient scope for `position`, `Skip-Gram-Vector` can store the environmental vectors of the words at each relative `position` of interest from the current word. The `Skip-Gram-Vector` predicate is used in establishing the word order information. The `Meaning-Vector(word:word distributed:environment)` predicate captures both the context and word order information for the word being updated.

Conditionals specify the rest of the DVRS$^+$ Sigma model. The conditional in Figure 2, for example, determines how context information is computed for words, with

```
CONDITIONAL Co-occurence
     Conditions: Co-occuring-Words(word:w)
     Actions: Context-Vector(distributed:d)
     Function(w,d): *environmental-vectors*
```

Fig 2. Conditional for context information.

Figure 3 explaining the computations implicitly defined by this conditional for a simplified hypothetical case where the vocabulary has only 4 words and the vector dimensionality is 5. Coming out of the condition is a message with a local vector for the other words in the sentence; that is, the vector's domain is the entire vocabulary and there is a value of 1 at every word in the sentence (and a value of 0 everywhere else). In Figure 3(a), the co-occurring words are the first and the fourth words, with a single zero region of width two sufficient to mark the second and third words as not co-occurring. Via the summary product algorithm, this vector is multiplied times the 2D function that stores the environmental vectors (Figure 3(b)) to yield a 2D function that is non-zero for only the environmental vectors of the words in the sentence (Figure 3(c)); and then the word variable is summarized out via integration (Figure 3(d)) – summing across all of the environmental vectors for words in the sentence – to generate a message for the action that is the distributed context vector. Because this message represents a distributed vector rather than a probability distribution, Sigma has been extended to do vector (or *l2*) normalization – i.e., sum of squares – over these messages rather than the normal form of probabilistic (or *l1*) normalization (Figure 3(e)). But, otherwise, a simple knowledge structure, in combination with the underlying summary product algorithm, computes what is necessary.

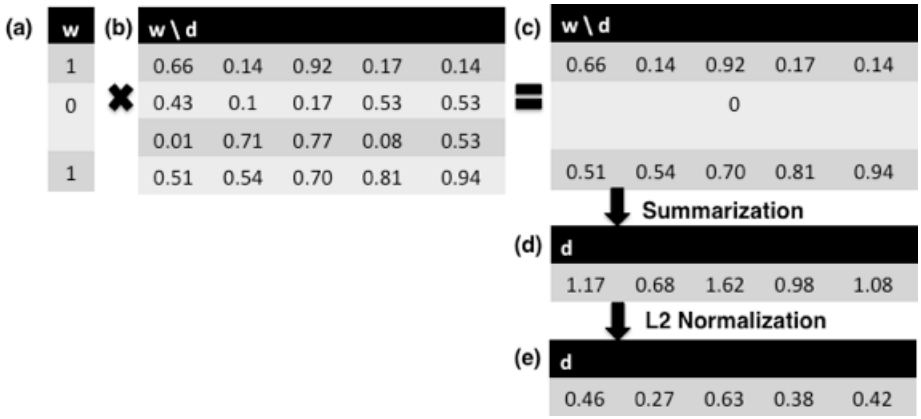

Fig. 3. Computation of context information.

Computing the ordering information is similar, albeit slightly more involved (Figure 4). The condition here yields a 2D function that captures the environmental vectors for nearby words according to their position – distance and direction – from the word being learned, while the function stores the unique sequence vectors, by position. The product yields a 2D function representing the pointwise products of the

```
CONDITIONAL Ordering
     Conditions: Skip-Gram-Vector(position:p distributed:d)
     Actions: Ordering-Vector(distributed:d)
     Function(p,d): *sequence-vectors*
```

Fig. 4. Conditional that computes the ordering vector.

corresponding environmental and sequence vectors, with summarization (and an *l2* norm) yielding the ordering vector via addition over these products.

The combination of context and ordering information occurs via a form of *action combination* that is Sigma's generalization of how multiple actions combine in parallel rule-based systems. Normal rule actions for the same predicate are combined in Sigma via *max*. For probabilistic information, multiple actions for the same predicate combine via *probabilistic or*. For distributed vectors, Sigma has been extended to use straight *addition* across multiple actions for the same predicate. For negative actions, Sigma normally inverts the message – converting, for example, 0s to 1s and 1s to 0s – and then multiplies this with the results of positive action combination; however, for vectors, negative actions simply imply *subtraction*.

The conditional in Figure 5 shows how an index for the current word is attached to the context vector – via outer product – to yield an action that influences the segment of the meaning/lexical vector that corresponds to the current word. A similar conditional with an identical action pattern also exists for the ordering vector. The results of these actions then combine additively to yield the total input to the meaning vector.

```
CONDITIONAL Context
  Conditions: Context-Vector(distributed:d)
              Current(word:w)
  Actions: Meaning-Vector(word:w distributed:d)
```

Fig. 5. Conditional for adding context information to meaning vector.

This input is then used to update, via gradient descent, the meaning/lexical function stored in the `Meaning` conditional shown in Figure 6. This differs from BEAGLE's superposition approach, but does so as to be able to leverage Sigma's existing learning algorithm in acquiring word meanings.

```
CONDITIONAL Meaning
  Condacts: Meaning-Vector(word:w distributed:d)
  Function(w,d): Uniform
```

Fig. 6. Conditional for gradient-descent learning with an initially uniform function.

The evaluation goal for DVRS$^+$ is to determine how well it performs in comparison to DVRS'. Although several new optimizations have been added to Sigma in support of distributed vector representations – including *variable tying*, so a single function could appear in multiple conditionals, and *sparse function products*, to speed up the products found in many of these conditionals [17] – efficiently processing large distributed vectors is still a challenge. So, in evaluating DVRS$^+$, lexical representations are learned only for the 46 distinct words in the *capital-common-countries* portion of the Google test data, which contains a total of 506 test instances; such as determining which word is the most similar to *Paris* in the way *Germany* is similar to *Berlin*. The training data included only the sentences containing at least one of these 46 distinct words, resulting in a training set with 65,086 distinct co-occurring words, each with a unique environmental vector, over 28,532 sentences.

When trained on the composite set of features with vectors of size 100 – and over a range of different random initializations because choice of random vectors can have

a significant effect with small vector sizes – standard DVRS' finds between 55.7% and 68.2% (median of 60.4%) of the best answers, but with only non-negative values it yields between 26.1% and 43.1% (median of 32.4%) of the best answers. DVRS$^+$ is too slow to run many random variations – it is currently 50 times slower than DVRS' (~4 hours for training rather than 5 minutes) – so only one version was run with a good, but not necessarily optimal, random initialization. This version finds 35.2% of the correct answers, placing it below the range for standard DVRS', but well within the range for non-negative DVRS'. It is above the median for the latter, but not at the maximum. There is thus a significant degradation due to the lack of negative values, plus possibly a smaller residual difference that may be due to issues in how gradient descent is operating here. Still, there is a promising positive effect with DVRS$^+$.

7 Conclusion

A new efficient algorithm has been introduced for learning distributed vector representations, with a variant of it having been implemented within Sigma in a functionally elegant manner that maximally leverages the existing mechanisms in the architecture. Although the implementation within Sigma is not yet totally complete, nor yet sufficiently efficient, it shows real promise. It also raises the possibility of pursuing other intriguing research problems. One key direction is using distributed vector representations of word meanings as a bridge between speech and language (and possibly cognition). Achieving this would yield a major demonstration of grand unification in Sigma. Pervasive use of distributed vector representations within Sigma could also yield both a native form of analogy and a form of semantic memory worth evaluating as a psychological model. However, success will require additional enhancements to Sigma. As discussed earlier, a full capability for negative values will be needed for improved effectiveness. Furthermore, DVRS$^+$ is considerably slower than the external DVRS', implying a need for significant further optimizations.

Further investigations are also worth pursuing with the DVRS' model, including: (1) training with larger data sets for more rigorous comparisons, (2) experimenting with different skip-grams rather than just 3-skip-bigrams, and (3) exploring the utility of distributed vector representations across a range of natural language tasks.

Acknowledgements. This work has been sponsored by the U.S. Army. Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government.

References

1. Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. A neural probabilistic language model. *The Journal of Machine Learning Research, 3,* 1137-1155. (2003).
2. Collobert, R., & Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning* (pp. 160-167). (2008).

3. Cox, G. E., Kachergis, G., Recchia, G., & Jones, M. N. Toward a scalable holographic word-form representation. *Behavior Research Methods*, *43*(3), 602-615. (2011).
4. Franklin, S., Madl, T., D'Mello, S., & Snaider, J. LIDA: A systems-level architecture for cognition, emotion, and learning. *IEEE Transactions on Mental Development*. (2013).
5. Guthrie, D., Allison, B., Liu, W., Guthrie, L., & Wilks, Y. A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation* (LREC-2006) (pp. 1-4). (2006).
6. Jones, M. N., & Mewhort, D. J. Representing word meaning and order information in a composite holographic lexicon. *Psychological review*, *114*(1), 1. (2007).
7. Koller, D., & Friedman, N. *Probabilistic Graphical Models: Principles and Techniques*. MIT press. (2009).
8. Kschischang, F. R., Frey, B. J., & Loeliger, H. A. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498-519. (2001).
9. http://mattmahoney.net/dc/textdata.html. Last accessed March 28th, 2014.
10. Mikolov, T., Chen, K., Corrado, G., & Dean, J. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations*. (2013).
11. Mnih, A., & Kavukcuoglu, K. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems* (pp. 2265-2273). (2013).
12. Plate, T. A. Holographic reduced representations. *IEEE Transactions on Neural Networks*, *6*(3), 623-641. (1995).
13. Riordan, B., & Jones, M. N. Redundancy in perceptual and linguistic experience: Comparing feature-based and distributional models of semantic representation. *Topics in Cognitive Science, 3*(2), 303-345. (2011).
14. Rosenbloom, P. S. Bridging dichotomies in cognitive architectures for virtual humans. In *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems*. (2011).
15. Rosenbloom, P. S. The Sigma cognitive architecture and system. *AISB Quarterly*, *136*, 4-13. (2013).
16. Rosenbloom, P. S., Demski, A., Han, T., & Ustun, V. Learning via gradient descent in Sigma. In *Proceedings of the 12th International Conference on Cognitive Modeling*. (2013).
17. Rosenbloom, P. S., Demski, A., & Ustun, V. Efficient message computation in Sigma's graphical architecture. Submitted to *BICA 2014*. (2014).
18. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. Learning representations by back-propagating errors. *Nature, 323*, 533-536. (1986).
19. Russell, S., Binder, J., Koller, D. & Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. *Proceedings of the 14th International Joint Conference on AI* (pp. 1146-1152). (1995)
20. Rutledge-Taylor, M. F., & West, R. L. MALTA: Enhancing ACT-R with a holographic persistent knowledge store. In *Proceedings of the XXIV Annual Conference of the Cognitive Science Society* (pp. 1433-1439). (2007).
21. Snaider, J., & Franklin, S. Modular composite representation. *Cognitive Computation*, 1-18. (2014).
22. Turney, P. D., & Pantel, P. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, *37*(1), 141-188. (2010).