Metaconcepts: Isolating Context in Word Embeddings

Peter Sutor Jr.
Department of Computer Science
University of Maryland
College Park, MD 20742, USA
psutor@umd.edu

Cornelia Fermüller
Department of Computer Science
University of Maryland
College Park, MD, 20742, USA
fer@umiacs.umd.edu

Yiannis Aloimonos
Department of Computer Science
University of Maryland
College Park, MD, 20742, USA
yiannis@cs.umd.edu

Douglas Summers-Stay
U.S. Army Research Labs Adelphi
Adelphi, MD 20783, USA
douglas.a.summers-stay.civ@mail.mil

***Abstract*— Word embeddings are commonly used to measure word-level semantic similarity in text, especially in direct word-to-word comparisons. However, the relationships between words in the embedding space are often viewed as approximately linear and concepts comprised of multiple words are a sort of linear combination. In this paper, we demonstrate that this is not generally true and show how the relationships can be better captured by leveraging the topology of the embedding space. We propose a technique for directly computing new vectors representing multiple words in a way that naturally combines them into a new, more consistent space where distance better correlates to similarity. We show that this technique works well for natural language, even when it comprises multiple words, on a simple task derived from `WordNet` synset descriptions and examples of words. Thus, the generated vectors better represent complex concepts in the word embedding space.**

I. INTRODUCTION

Vector embeddings serve to embed semantic relationships into discrete points in (typically) high-dimensional spaces. Word embeddings have become more popular recently thanks to the successes of `word2vec` [13] and `GloVe` [18]. Word embeddings are attractive due to their ability to measure similarity in the semantic sense between words and phrases by measures such as cosine similarity on the vectors and linear combinations of them. Generally speaking, words that are close to each other are related in a contextual sense. In particular, these word embedding models are interesting due to how well they model analogies between words, such as the famous $king + woman - man \approx queen$ example, shown for convenience in Figure 1.

However, cosine similarity may not be the optimal measure of semantic similarity, or linear combinations of vectors the best representation for phrases. While it works well enough in practice, there is no reason to assume these vectors should be linearly nor angularly related. Although they are by no means deep learning, `word2vec` and `GloVe` both learn their vectors from corpora in self-supervised ways using neural networks, and thus it stands to reason that the vectors generated from these neural networks exist as samples on a more complex manifold.

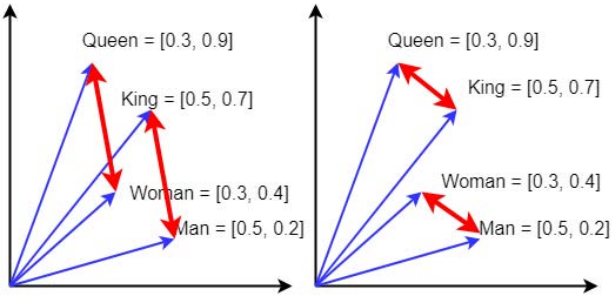

Fig. 1. The classical $king + woman - man \approx queen$ example of neural word embeddings, in 2D. It must follow that $king - man \approx queen - woman$, and we can visually see that in the red arrows. There are 4 analogies one can construct, based on the parallel red arrows and their direction. This is slightly idealized; the vectors need not be so similar to be the most similar from all word vectors. The similar direction of the red arrows indicates similar relational meaning.

Furthermore, since word vectors are static, they do not fair well in contextualization. All of the senses and contextual meaning of a word are balled up into a single, static vector, and thus it is difficult to disentangle the components of each sense of a word from its vector representation. Instead, averaging vectors for a phrase serves to try and reconcile this weakness in phrase-level contextualization by assuming the average of vectors will land near the actual average of their meaning, though it is not clear that each word should have equal weight regardless of the context in which it is used.

In this paper we investigate the idea of estimating the contextual components of word vectors given a context in which they are used. We directly utilize the word embedding

978-1-7281-1198-8/19/$31.00 ©2019 IEEE
DOI 10.1109/MIPR.2019.00110

model and its properties to create such contextual vectors, without the use of any learning mechanism. We further generalize this to handle phrases and sentences, while still confining the output to the same space. Measuring similarity on these vectors is more natural than measuring similarity on averages. To see how well the context is preserved with this technique, we devise a challenging experiment on `WordNet` synsets that is tailored to the task of context disambiguation for a given word or phrase.

II. RELATED WORK

There is much existing work in contextualization of word embeddings [12], [10], [17], though this is mostly confined to the realm of machine learning, with the goal being to adapt an existing vector set into a contextualized one, to aid in a downstream task. Other methods attempt to enrich vectors with context by learning them together in an end-to-end fashion, typically with a neural network as well [19], [20]. We differ from such methods in that we do not use any sort of learning but isolate context directly from word vectors. The improvements we observe come for free by replacing normal cosine similarity with our technique. Our method can thus be used in conjunction with such neural network based approaches as well.

More in line with our approach are those that avoid neural networks altogether, such as with latent and/or probabilistic models. For example, measuring similarity in context based on *a priori*, out of context likelihood for senses of words [5]. There have also been attempts to address the issue of singular word vectors' weaknesses at contextualizing and capturing meaning across phrases [7]. However, they differ from us by imposing a structure on word order and co-occurrence. We simply try to find a better way to *use* the existing vectors themselves. To the best of our knowledge, there is no definitive work on improving the similarity functions for word embeddings directly to take context into account, but we do take inspiration from some of the similarity measurements proposed by [11].

III. BACKGROUND INFORMATION

In this section we review the necessary background information about word embeddings and the popular neural network based models `word2vec` [13] and `GloVe` [18], and simple operations on these embeddings.

A. Word Embeddings

The basic premise behind word vectors is to form a vector representation of a word based on co-occurrence statistics. By predefining a *context window*, usually of the number of words around the target word, the co-occurrence counts are obtained by counting how often a word occurs within the context of another word. Each row in the resultant matrix forms a distribution of co-occurrence counts. The row is then a vector representation of the word. This approach is based on the famous quote "you shall know a word by the company it keeps" by linguist John Rupert Firth [9], an observation that has held true in even the modern forms of word embeddings. The generalization of this technique is often referred to as *distributional statistics* and for semantic word vectors *distributional semantics*.

The distributional technique for word vectors suffers from dimensionality issues, as the matrix is a square matrix of size equal to the vocabulary of words, which can be hundreds of thousands, or even millions, of words. To mitigate this, the vector set formed by the vocabulary is embedded into a lower dimensional, real-valued vector space. Since the vectors in the original matrix tend to be very sparse, as many words never appear in the contexts of others, little information is lost in these compressions. One of the earliest examples of this is matrix factorizing through Singular Value Decomposition (SVD) and truncating the resulting factorizations. This technique is referred to as Latent Semantic Analysis (LSA) [6].

In recent times, neural models for generating word embeddings have been shown to be more effective at capturing the semantic relationships between words [3]. In particular, these more successful models attempt to predict contexts of words from co-occurrences, as opposed to directly relying on the counts in the matrix. We will use these for our experiments, as we are particular interested in the relational properties between word vectors. However, this is not to say that count-based neural network or even non-neural network models are strictly inferior at modeling word vectors, as shown by [11] - they are better at optimizing the hyper-parameters.

Such neural models are closely tied to neural language models, which try to learn a probability distribution over words in a vocabulary and predict words in a sequence. This is done by discretizing a corpus into occurrences of N-grams, or subsequences of words of length N (mixed-gram models can incorporate multiple values of N, typically from 2 to 5), and passing these through a feed-forward network which learns to predict future words given a context. The soft-max layer that performs this optimization outputs embeddings as a result for each word, as observed by Bengio, et al. in [4].

B. Skip-Gram Neural Word Embeddings

We now describe `word2vec`, specifically focusing on the Skip-Gram model with negative sampling, as we use the popular and massive Google negative sampled 300 dimensional vector model of 3 million unique words, discussed in [14]. Loosely speaking, the Skip-Gram model [13] tries to predict the context around a given word. Given a word w_t and the sequence (context) around it:

$$C_t = w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n}$$

the Skip-Gram objective is to maximize the sums of the log probabilities across all T words:

$$\sum_{t=1}^{T} \log(P(C_t|w_t)) \tag{1}$$

The logarithmic term in Equation 1 can be further reduced:

$$\log P(C_t|w_t) = \sum_{j=-n}^{n} \log(P(w_{t+j}|w_t)) \quad (2)$$

This is a soft-max operation, and is asymptotically proportional to the number of words, as noted in [14]. The hierarchical soft-max [16] is instead performed to handle this operation efficiently, in logarithmic time.

The takeaway here is the observation in Equation 2. It's often surmised that the compositionality of `word2vec`'s vectors comes from the maximization across the sum of these log prediction probabilities. If two words occur in each other's context's often, the model will generate a larger response to that in order to maximize the log probabilities of their overlapping contexts. However, it is also clear that the context window necessitates a strict cut-off in the extent to which `word2vec` will do this. Moreover, statistics on longer n-grams also decrease rapidly, so enlarging the context window does not help the curse of dimensionality.

C. `GloVe` Neural Word Embeddings

`GloVe` [18] is a special mix of traditional matrix factoring approaches for word embeddings and `word2vec` style word embeddings. A co-occurrence matrix X is obtained in largely the usual way, and turned into word co-occurrence probabilities. Words far away relative to the context window decay in value. The `GloVe` model defines the constraint:

$$w_i^T w_j + b_i + b_j = \log(X_{ij}) \quad (3)$$

on each word pairing in X, where i is the current word, and j is a word appearing in context, with bias terms b_i and b_j for each. The bias terms absorb unrelated bias in the log probabilities of their co-occurrence, and the linear product of the word's weights preserve the linear relationship between their co-occurrence. Thus, it stands to reason that minimizing Equation 3 minus $\log(X_{ij})$ for each pair will produce a close, log-linearly related approximation to X - this is done by a neural network that optimizes the following objective:

$$\sum_{i,j} f(X_{ij})(w_i^T w_j + b_i + b_j - \log(X_{ij}))^2 \quad (4)$$

where $f(X_{ij})$ is a weighing function that down scales the importance of common words by their probability and keeps infrequent words at a constant weight. Once again, the logarithmic and linear relationship between co-occurring words is maintained. Under the right parameters, `GloVe` can perform just as well as `word2vec`, and vice versa [3].

D. Common Operations

Here we describe some of the relevant and common operations performed with word vectors, for reference.

Similarity: The most common method of measuring the semantic similarity between two vectors is with cosine:

$$s_{cos}(u, v) = 1 - d_{cos}(u, v) \quad (5)$$

where d_{cos} is the cosine distance:

$$d_{cos}(u, v) = \frac{a \cdot b}{\|a\|_2 \|b\|_2} \quad (6)$$

Analogy: Analogical reasoning can be performed on word vectors by finding the closest (non-trivial) word vector d to:

$$a + (b - c) \quad (7)$$

where a is to b, as c is to d. Furthermore, one can measure the strength of an analogical relationship by:

$$s_{cos}(a - c, d - b) \quad (8)$$

or with one of the other 3 analogy variants. Refer to Figure 1 for an example.

Average: Averaging word vectors is performed the usual way by summing over a set of word vectors and dividing by the number of vectors:

$$\overline{a} = \sum_{k \in A} a_k \quad (9)$$

Weighted versions of this also exist.

IV. METACONCEPTS OVER WORDS

In this section we describe our method of contextualizing word vectors. We will refer to contextualized versions of vectors as *metaconcepts* from here forth, as they are formed through combining self-referential projections of nearest neighbors of constituent words or phrases.

A. Contexts of Contexts

As mentioned in the Skip-Gram `word2vec` and `GloVe` sections, word embeddings typically strive to maintain linear relationships between their representations for word vectors. However, it is evident that this mostly applies up to and near the pre-defined context window and any overlapping context windows for words. As such, the word vectors closest to another vector are the most confident in similarity to that vector. Consequently, it is expected that the local topology of clusters of words is nearly linear.

Consider the set of k nearest neighbors U_k to a word vector u. Projecting u onto each element will isolate the components of u that best relate to that element. This is what happens when distance is measured between word vectors. If the cosine distance using Equation 6 is employed, the magnitude of a vector is irrelevant. Furthermore, when cosine distance is positive, there is at least some correlation in the correct direction between the projected components and the vector being projected on. Vector u is nearly guaranteed to have all nearest k neighbors to be positive, though this is not strictly true. The average of all positive vectors then tends to point in the same direction as u. Let:

$$z = \frac{1}{|S_u(v)|} \sum_{s \in S_u(v)} s \quad (10)$$

be the context average of a vector v, where $S_u(v)$ is the subset of U_k with positive cosine similarity to v. $C_u = U_k$

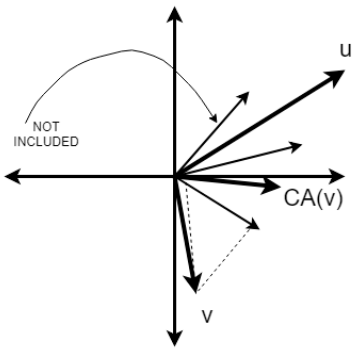

Fig. 2. An example of the context averaging described in Equation 10. The 3 thin vectors are nearest neighbors to u. The context average $CA(v)$ for a vector v excludes the indicated vector as it cannot project on it with a positive scalar value. The other two are valid, and so $CA(v)$ becomes their average. A joint context average differs only in that v is projected on its own neighbors as well, and is averaged with u's projection on v and its own neighbors. Then their vectors are averaged in a weighted fashion.

is referred to as the context of u. The average of a vector across U_k is then the context average over U_k of the vector. The result of a context average is interpreted as a good subset of nearest neighbors to average over; a best fit location of v in the local topology of u. Figure 2 shows how context averaging affects a vector v in relation to u. Note that if v is far from u, the context average can still land very close to u, perhaps even being equal to it, so long as there is at least some correlation in many of the same context elements. Now, consider the case where we have two words u and v with two different contexts C_u and C_v. We can consider their union $C_{u,v} = C_u \cup C_v$ to be a joint context. Consider the vector:

$$z = \frac{1}{|S_u(v)| + |S_v(v)|} \left(\sum_{s \in S_u(v)} s + \sum_{s \in S_v(v)} s \right) \tag{11}$$

where S_u and S_v are the set of positive cosine similar vectors to the joint context for u and v, respectively, referred to as the joint context average. This represents the best weighted context average to $C_{u,v}$, by performing a weighted average across the contexts averages of each vector to the context.

Now consider the situation where we wish to cast v into the context provided by u. For example, if v is "bank" and u is "river", then we want a new vector v_u such that it captures bank in the context of river (that is to say, the bank of a river). Our strategy is as follows: compute Equation 11, calling it v_{joint}, then Equation 10 for u with context $C_{u,v}$, calling it u_{joint}. In other words, we compute the joint context average, and then the regular context average for u, but on the joint context. Geometrically, the ideal scenario here is that the relationship between u, u_{joint}, v, and v_{joint} is anti-analogical: v is to v_{joint} as u_{joint} is to u, i.e., they point to each other. Figure 3 visualizes this relationship. We compute the following cosine similarity to measure the contextual association between v and the contextualized $v_u = v_{joint}$:

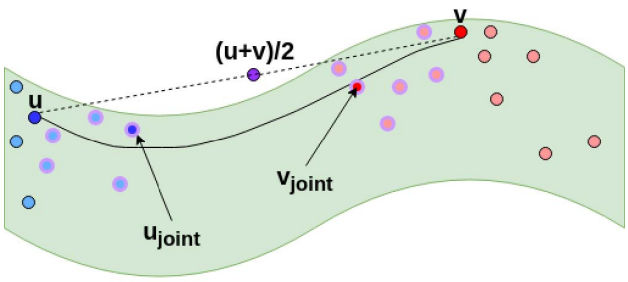

Fig. 3. An example of the relationship between u, v, u_{joint}, and v_{joint}. The orange / orange-purple points represent nearest neighbors of v, where the orange-purple ones signify their inclusion in the joint context average. The same is true of the blue / teal-purple points for u. The u_{joint} and v_{joint} vectors are the result of the joint context averages on u and v, respectively. The average and line of potential weighted averages (dotted line) of u and v is shown, along with the true geodesic (black line) along an unknown manifold (green). Averages can deviate far from the manifold.

$$s_{cos}(v - v_{joint}, u_{joint} - u) \tag{12}$$

B. Recursive Metaconcepts

We can generalize our technique to work across not only pairs of single words but multi-word sets. To do this, we perform an additional step before computing the contextual association, where we find context averages for each set of words being compared. This is done recursively by grouping words in a set together, combining their contexts, and computing Equation 12, with v_{joint} being computed the same as u_{joint}, using Equation 10 on the joint context. Whichever pair has the highest contextual similarity is voted to be merged into a single unit (a single metaconcept) by computing Equation 11. This repeats until only a single metaconcept is left, the vector and context of which is used as a word vector and its context normally would.

V. EXPERIMENTS

In this section we detail our experiments for testing the efficacy of metaconcepts.

A. The Problem Setup

Since the primary role of metaconcepts is to generate contextualized vectors from word embeddings, we design our experiment testing this aspect. We formulate our experiment in the form of a task that is designed to be difficult for word embeddings to solve with just common operations on their vectors. This task is word sense disambiguation in the form of a free-form matching problem. For example, if we have a word with 5 senses, and each sense has a definition that is available to us, we wish to match the sense definition with the correct usage of it in an example sentence. Say the word is "ride" and the sense we are considering is "using something as a form of transportation". The sentence "I hitched a ride on a friend's car this morning" must be correctly matched with this sense of "ride". The problem is made more difficult by making each decision independently, so that process-of-elimination techniques cannot be used to narrow down the

S ≥	Method	w2v (U)	GloVe (U)	w2v (W)	GloVe (W)
2	baseline	0.387	0.304	0.304	0.177
2	average	0.440	0.367	0.361	0.246
2	meta5	0.536	0.518	0.470	0.459
2	meta10	0.553	0.494	0.482	0.457
2	meta20	0.574	**0.534**	0.505	**0.536**
2	meta50	**0.590**	0.514	**0.514**	0.495
3	baseline	0.251	0.202	0.206	0.132
3	average	0.321	0.278	0.273	0.205
3	meta5	0.428	0.419	0.393	0.371
3	meta10	0.449	0.436	0.403	0.403
3	meta20	0.481	**0.538**	0.432	**0.540**
3	meta50	**0.494**	0.444	**0.436**	0.446
4	baseline	0.193	0.154	0.163	0.108
4	average	0.271	0.231	0.232	0.181
4	meta5	0.390	0.281	0.363	0.247
4	meta10	0.401	0.421	0.365	0.371
4	meta20	0.451	**0.474**	0.401	**0.498**
4	meta50	**0.468**	0.333	**0.403**	0.375
6	baseline	0.121	0.125	0.108	0.092
6	average	0.172	0.200	0.157	0.165
6	meta5	**0.419**	0.367	**0.354**	0.391
6	meta10	0.384	**0.390**	0.328	**0.395**
6	meta20	0.414	0.373	0.347	0.391
6	meta50	0.373	0.333	0.312	0.352

TABLE I

Table of results for the WordNet sense disambiguation task comparing the random baseline, benchmark (average word vectors), and metaconcepts approaches. The number after "meta" is the number of nearest neighbors used. We show results across increasing lower bounds for the minimum number of senses (S) for both `word2vec` (w2v) and `GloVe`, showing both the unweighted (U) and weighted (W) results. Underlined entries are maximal in their row and bolded entries are maximal in their column.

number of remaining sentences and senses to be matched. As examples for a set of senses of a word come in, we do not know at any given point if we've already matched a particular sense. We can only choose the highest score among all possibilities each time. If the contextualization does not work correctly, the highest score may either choose the wrong matching outright or repeatedly choose to match certain senses since it has not rid itself of the bias of other senses. We expect that word embeddings will not fare well in this regard without the use of metaconcepts.

B. Using WordNet as a Dataset

We obtain exemplar words, senses, definitions and examples from WordNet's synsets, and use WordNet's matching as the correct answer. WordNet [15], [8] is a lexical database for English and is divided into synonym sets (synsets) for words, which represent the sense of a word being used. For any given word with n senses, we select one example of each sense and the definition of a sense, and treat this as a mapping to one another.

For each pairing, we compute a score of contextual relatedness on their sentences. Since we are not supposed to know the categories we selected as we choose the best scores, we always take the top n scores in the resulting n by n matrix of scores. In this sense, we are treating the scoring as a *ranking* across all pairs. We perform this on each word, and tally the number of times we were correct and the number of times we were wrong to compute our accuracy. Since some words have more senses than others, we reflect the challenge of this by computing another, weighted score, where the number of possible senses is the weight on that correct answer. We then divide by the total number of possibilities across all answers. We also try restricting the least number of senses in various experiments to try and get a sense of how the score changes as the overall experiment becomes harder and harder.

C. Scoring the Association of Pairs

For vanilla word embeddings, we tried a mix of averaging the words in the sentences and comparing (thus, not knowing the actual word) and including the word in the average of the sentences to naïvely influence the context behind the usage of each sentence (both by averaging with the average at the end, and simply including it in the average). We found that including the word in the average always yielded inferior accuracy, so we stick to simple averaging of sentences and computing the cosine similarity between them. This serves as a benchmark for vanilla word embeddings' ability to infer the context. We also compute the score that one would get by random guessing as the baseline for the problem. Depending on the least number of senses we allow for a word, the baseline changes, getting smaller and smaller rapidly.

To benchmark our metaconcept technique, we first start by performing metaconcept recursion on each sentence in the pair, and on the WordNet word. The resulting vectors and contexts for the description and the example sentences are used with the WordNet word to compute Equation 11, or the joint context average with the word. This generates two vectors d_{joint} and e_{joint} for the description and example sentences. These have been contextualized with the WordNet word. Finally, we compute vectors d and e by via the context average on d_{joint} and e_{joint} with the union of their set of contexts. We use these values for the cosine similarity on the analogical relationship between them with Equation 12.

D. Experimental Results

As shown in Table 1 in IV-B, word embeddings only slightly outperform the baseline when simple averaging is used, while metaconcepts can hit over 50% accuracy when all words with at least two senses are considered. Because no learning mechanism was used, the improvement is solely due to the technique used to generate metaconcepts. When k is small, the results suffer, but our experimentation showed little improvement after $k = 50$; we recommend considering this many neighbors. As the number of senses increases, the difficulty of the classification problem does as well, as the number of options grow quadratically. However, our method seems to not feel the effects of this nearly as much as the average does, in relation to the baseline. This is because the contextualization via metaconcepts has better ranking properties, and does not suffer from the growing number of possible classes to choose from.

VI. DISCUSSION

It should be noted that our metaconcept technique requires careful consideration during implementation, as computing

the k nearest neighbors to a vector is expensive. A naïve implementation would run in $O(kdn)$ time, where k is the number of nearest neighbors, d is the model dimension, and n is the number of vectors. However, $O(\log(n))$ methods exist for approximating nearest neighbors [2], [1]. They can also be precomputed for a known vocabulary. We brute forced this process and computed all nearest neighbors for each word embedding model for 200 nearest neighbors.

When precomputed, the nearest neighbors become a non-issue, as our algorithm is purposefully designed to not require the nearest neighbors to a novel vector. However, our method's runtime is directly dependent on the number of nearest neighbors to consider, and the number of recursive levels needed for metaconcept recursion. Our experiments showed that 50 nearest neighbors are more than sufficient. Without using high-performance computing techniques, it can be an expensive operation, though still largely real time.

Our results indicate that metaconcepts provide good contextualized versions of word embeddings and can potentially be good features for contextualization in other learning systems. Furthermore, it works well for longer units of language. It may be possible to use metaconcepts as features in word vector based learning of document and phrase level vectors. More research is required to investigate how incorporating metaconcepts affects results on other tasks, and is a natural next step for future work. As our experiment shows that even single words can be confidently compared to more complex phrases, metaconcepts can be used as a bridge between tags obtained from meta-data or classification in images / other multi-media, and natural language. Another interesting line of future work is utilizing metaconcepts to measure association between multi-media where tags to words in natural language are available, in order to rank how relevant a piece of multi-media is to a certain topic.

VII. CONCLUSION

We have proposed a novel, non machine-learning method of using word vectors to both give context to other vectors and express more complicated units of language and their relationships. We have empirically shown that our proposed metaconcept strategy works far better than standard word vector operations for providing context to vectors by boosting the accuracy of word embeddings on a difficult sense classification problem on WordNet words. Our results increase the efficacy of contextualization from the baseline several-fold when compared to simple averaging.

ACKNOWLEDGMENT

The views, opinions, and/or findings expressed are those of the authors' and do not represent the official views or policies of the Department of Defense or the US Government. Approved for public release, distribution unlimited.

Thank you to Dr. Reza Ghanadan, Dr. Hava Siegelmann, and Boston Engineering Corporation for advice and support on this topic. Special thanks to David Shane of Boston Engineering for numerous discussions and his advice on research direction and testing. This research has been supported in part by Boston Engineering Corporation through DARPA contract number W31P4Q-13-C-0136.

REFERENCES

[1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[2] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[3] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[4] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. *Neural Probabilistic Language Models*, pages 137–186. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[5] Georgiana Dinu and Mirella Lapata. Measuring distributional similarity in context. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1162–1172, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[6] Susan T. Dumais. Latent semantic analysis. *Annual Review of Information Science and Technology*, 38(1):188–230, 2004.

[7] Katrin Erk and Sebastian Padó. A structured vector space model for word meaning in context. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 897–906, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[8] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

[9] J. R. Firth. A synopsis of linguistic theory 1930-55. 1952-59:1–32, 1957.

[10] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[11] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[12] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6297–6308, 2017.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[15] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[16] Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

[17] Siddharth Patwardhan and Ted Pedersen. Using wordnet-based context vectors to estimate the semantic relatedness of concepts. In *Proceedings of the Workshop on Making Sense of Sense: Bringing Psycholinguistics and Computational Linguistics Together*, 2006.

[18] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[19] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[20] Stefan Thater, Hagen Fürstenau, and Manfred Pinkal. Contextualizing semantic representations using syntactically enriched vector models. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 948–957. Association for Computational Linguistics, 2010.