Nazarbayev University
School of Sciences and Humanities

MATH 499 Capstone Project

Semantics- and Syntax-related Subvectors in the Skip-gram Embeddings

Submitted By:
Maxat Tezekbayev

Research Supervisor:
Zhenisbek Assylbekov

Second Reader:
Rustem Takhanov

Nur-Sultan, 2020

Abstract

This project presents that using a modified version of Skip-gram with negative sampling with weight tying proposed by Assylbekov and Takhanov (2019) word embedding of each word can be decomposed into two subvectors: $\mathbf{x}$ and $\mathbf{y}$. We empirically showed that $\mathbf{x}$-subvector is more related to the semantic role of the word and $\mathbf{y}$-subvector is more related to the syntactic role of the word.

1 Introduction

Nowadays word embeddings play a crucial role in variety of natural language processing tasks, such as part-of-speech tagging, machine translation, and question answering. Skip-gram with negative sampling (SGNS) which was proposed by Mikolov, Sutskever, et al. (2013) is one of the widespread and simple models to obtain these word embeddings. Assuming that words have already been converted into indices, let $\{1, \dots, n\}$ be a finite vocabulary of words from some corpus. Following the setups of the widely used SGNS model (Mikolov et. al, 2013), we consider *two* vectors per each word i:

- $\mathbf{w}_i$ is an embedding of the word i when i is a center word,
- $\mathbf{c}_i$ is an embedding of the word i when i is a context word.

We follow the assumptions of Assylbekov and Takhanov (2019) on the nature of word vectors, context vectors, and text generation, i.e.

1. A priori word vectors $\mathbf{w}_1$,...,$\mathbf{w}_n$ $\in$R^d are i.i.d. draws from isotropic multivariate Gaussian distribution: $\mathbf{w}_i \overset{\text{iid}}{\sim} \mathcal{N}\left(\mathbf{0},\, \frac{1}{d}\mathbf{I}\right)$, where $\mathbf{I}$ is the $d \times d$ identity matrix.
2. Context vectors $\mathbf{c}_1, \dots, \mathbf{c}_n$ are related to word vectors according to $\mathbf{c}_i = \mathbf{Q}\mathbf{w}_i$, $i = 1, \dots, n$, for some orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$.
3. Given a word i, the probability of any word j being in its context is given by

$$p(j \mid i) \propto p_j \cdot e^{\mathbf{w}_i^\top \mathbf{c}_j} \tag{1}$$

   where $p_j = p(j)$ is the unigram probability for the word j.

Assylbekov and Takhanov (2019) showed that context vectors are reflections of word vectors in approximately half the dimensions and proposed theoretically grounded way of tying weights in the SGNS model. Zobnin and Elistratova (2019) using other procedures also showed that context vectors are reflections of word vectors, but they empirically suggest using lower amount of reflections. Under the assumptions 1–3 above, Assylbekov and Takhanov (2019) showed that each word's vector $\mathbf{w}_i$ splits into two approximately equally-sized subvectors $\mathbf{x}_i$ and $\mathbf{y}_i$, and the model (1) for generating a word j in the context of a word i can be rewritten as

$$p(j \mid i) \approx p_j \cdot e^{\mathbf{x}_i^\top \mathbf{x}_j - \mathbf{y}_i^\top \mathbf{y}_j}.$$

Interestingly, embeddings of the first type ($\mathbf{x}_i$ and $\mathbf{x}_j$) are responsible for pulling the word j into the context of the word i, while embeddings of the second type ($\mathbf{y}_i$ and $\mathbf{y}_j$) are responsible for pushing the word j away from the context of the word i. We hypothesize that the $\mathbf{x}$-embeddings are more related to semantics, whereas the $\mathbf{y}$-embeddings are more related to syntax. We provide a motivating example for this hypothesis and then empirically validate it through controlled experiments.

2 Skip-gram model

In this section we will consider a toy example which will gives us an idea how word embeddings are obtained via the Skip-gram model.

2.1 Toy example

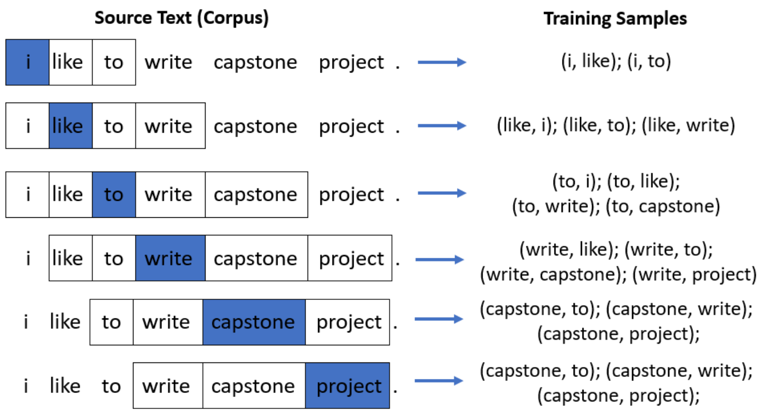
Consider a very small corpus: *"i like to write capstone project"*. In this text, we have 6 distinct words and the number of words in our vocabulary is 6, which we can denote as $\mathcal{W}$. We can represent each word in terms of one-hot vectors:

$$\begin{aligned}
\mathbf{x}_{\text{i}} &= \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \in \mathbb{R}^{|\mathcal{W}|} \\
\mathbf{x}_{\text{like}} &= \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\end{bmatrix} \in \mathbb{R}^{|\mathcal{W}|} \\
\mathbf{x}_{\text{to}} &= \begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0\end{bmatrix} \in \mathbb{R}^{|\mathcal{W}|} \\
\mathbf{x}_{\text{write}} &= \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0\end{bmatrix} \in \mathbb{R}^{|\mathcal{W}|} \\
\mathbf{x}_{\text{capstone}} &= \begin{bmatrix}0 & 0 & 0 & 0 & 1 & 0\end{bmatrix} \in \mathbb{R}^{|\mathcal{W}|} \\
\mathbf{x}_{\text{project}} &= \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 1\end{bmatrix} \in \mathbb{R}^{|\mathcal{W}|}
\end{aligned}$$

When we represent words as one-hot vectors, the size of each word's vector is equal to the number of words in our vocabulary, which in the real world can be more than 1 million words. These vectors are sparse and there is no notion of similarity between these vectors because they are orthogonal to each other. However, we know that at least some words have synonyms in the real world. To overcome this problem, we want to encode similarity in the vectors themselves and represent them in more dense way so that words that share common contexts in the corpus are located close to one another in the vector space. Let d be the size of such vectors, and usually we choose d to be from 50 to 300. In our toy example, we can choose d to be equal to 3.

2.2 How the Skip-gram works

The Skip-gram model is a one hidden layer feedforward neural network. First of all, we need to create our dataset to train our model. Considering our toy example, we can create pair of words, where the first word is a center word and the second word is a word that appears in the context of the first word within some window. We can choose window size to be equal to 2 and create our training dataset:


Source Text (Corpus)
Training Samples
i like to write capstone project .
(i, like); (i, to)
(like, i); (like, to); (like, write)
(to, i); (to, like); (to, write); (to, capstone)
(write, like); (write, to); (write, capstone); (write, project)
(capstone, to); (capstone, write); (capstone, project);
(capstone, to); (capstone, write); (capstone, project);

The Skip-gram takes any center word as input and tries to predict the surrounding window of its context words. However, in the end, we will just use the weights of the hidden layer as the word embeddings.
Consider a pair of words (i, j). We take a one-hot vector of the word i as input and one-hot vector of the word j as a desired output of the neural network and we want to maximize probability of the word j to appear in the context of the word i.
Word embeddings stacked into a matrix $\mathbf{W}$:

$$\begin{array}{c} i \\ like \\ to \\ write \\ capstone \\ project \end{array} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \\ w_{61} & w_{62} & w_{63} \end{bmatrix} \begin{array}{c} \mathbf{w_1} \\ \mathbf{w_2} \\ \mathbf{w_3} \\ \mathbf{w_4} \\ \mathbf{w_5} \\ \mathbf{w_6} \end{array}$$

Context embeddings stacked into a matrix $\mathbf{C}$:

$$\begin{array}{c} i \\ like \\ to \\ write \\ capstone \\ project \end{array} \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \\ c_{41} & c_{42} & c_{43} \\ c_{51} & c_{52} & c_{53} \\ c_{61} & c_{62} & c_{63} \end{bmatrix} \begin{array}{c} \mathbf{c_1} \\ \mathbf{c_2} \\ \mathbf{c_3} \\ \mathbf{c_4} \\ \mathbf{c_5} \\ \mathbf{c_6} \end{array}$$

$\mathbf{W}$ represents weights of the neural network between the input and hidden layers. $\mathbf{C}$ represents weights of neural network between the hidden and output layers.
How do we get output of the model? Consider a pair of words (i, j):

1. $\mathbf{W^T} \cdot \mathbf{x_i} = \mathbf{w_i^T}$(because $\mathbf{x_i}$ is one-hot vector)
2. $\mathbf{Cw_i^T} = \begin{bmatrix} \mathbf{c_1w_i^T} & \mathbf{c_2w_i^T} & \mathbf{c_3w_i^T} & \mathbf{c_4w_i^T} & \mathbf{c_5w_i^T} & \mathbf{c_6w_i^T} \end{bmatrix}$
3. Then apply $softmax$ function to obtain "probabilities" for each word to appear in the context of word i:

$$\text{softmax} \begin{bmatrix} \mathbf{c_1w_i^T} \\ \mathbf{c_2w_i^T} \\ \mathbf{c_3w_i^T} \\ \mathbf{c_4w_i^T} \\ \mathbf{c_5w_i^T} \\ \mathbf{c_6w_i^T} \end{bmatrix} = \begin{bmatrix} \frac{\exp(\mathbf{c_1w_i^T})}{\sum_{k=1}^6 \exp(\mathbf{c_kw_i^T})} \\ \frac{\exp(\mathbf{c_2w_i^T})}{\sum_{k=1}^6 \exp(\mathbf{c_kw_i^T})} \\ \frac{\exp(\mathbf{c_3w_i^T})}{\sum_{k=1}^6 \exp(\mathbf{c_kw_i^T})} \\ \frac{\exp(\mathbf{c_4w_i^T})}{\sum_{k=1}^6 \exp(\mathbf{c_kw_i^T})} \\ \frac{\exp(\mathbf{c_5w_i^T})}{\sum_{k=1}^6 \exp(\mathbf{c_kw_i^T})} \\ \frac{\exp(\mathbf{c_6w_i^T})}{\sum_{k=1}^6 \exp(\mathbf{c_kw_i^T})} \end{bmatrix} \longrightarrow \mathbf{y} = \begin{bmatrix} \mathbf{y_1} \\ \mathbf{y_2} \\ \mathbf{y_3} \\ \mathbf{y_4} \\ \mathbf{y_5} \\ \mathbf{y_6} \end{bmatrix}$$

2.3 Objective function of Skip-gram

As was mentioned previously, considering pairs of words (i, j), we want to maximize probability of word j to appear in the context of word i and therefore our objective function is:

$$\begin{aligned} \max p(j|i) &= \max y_{j^*} \\ &= \max \log y_{j^*} \\ &= \mathbf{c_{j^*}w_i^T} - \log \sum_{k=1}^{6} \exp(c_k w_i^T) := -E \end{aligned} \qquad (2)$$

Insted of maximizing $-E$, we can minimize E.

2.4 Skip-gram with negative sampling

However, calculating $\log$ term of (2) for each pair (i, j) can be computationally costly because, in a real corpus, we will have more than 100,000 words in it. Therefore, Mikolov, Sutskever, et al. (2013) proposed a modified objective function where instead of having too many context vectors that need to be updated per iteration, we only update a sample of them:

$$E = \log \sigma(c_{j^*} w_i^T) - \sum_{m \in \mathcal{W}_{neg}} \log \sigma(c_m w_i^T) \tag{3}$$

where σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

and $\mathcal{W}_{neg}$ is the set of words that are sampled based on some arbitrarily chosen probabilistic distribution.

2.5 Updating the weights

For updating weights in the neural network, we use backpropagation and gradient descent method. Let $u_j = c_j w_i^T$. To obtain the update equations of the weights, we take the derivative of E with regard to u_j:

$$\frac{\partial E}{\partial u_j} = \sigma(u_j) - t_j := e_j \tag{5}$$

where $t_j = \mathbb{1} \cdot (j = j^*)$. Then we take the derivative of E with regard to each entry k of context vector c_j:

$$\frac{\partial E}{\partial c_{kj}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial c_{kj}} = e_j \cdot w_k \tag{6}$$

Which results in the following update equations for each entry of context vector c_j:

$$c_{kj}^{(new)} = c_{kj}^{(old)} - \alpha \cdot e_j \cdot w_k. \tag{7}$$

where α is a hyperparameter of gradient descent method. Let $h_k = \sum_{i=1}^{V} x_i \cdot w_{ik}$. It represents k-th entry of the hidden layer. We take the derivative of E with regard to h_k:

$$\frac{\partial E}{\partial h_k} = \sum_{j=\{j^*\} \cup \mathcal{W}_{neg}} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_k} = \sum_{j=\{j^*\} \cup \mathcal{W}_{neg}} e_j \cdot c_{jk} := a_k \tag{8}$$

Taking the derivative of E with regard to to each entry k of word embedding w_i:

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial E}{\partial h_k} \cdot \frac{\partial h_k}{\partial w_{ik}} = a_k \cdot x_i \tag{9}$$

Which results in the following update equations for each entry of word embedding w_i:

$$w_{ik}^{(new)} = w_{ik}^{(old)} - \alpha \cdot a_k \cdot x_i. \tag{10}$$

3 Motivating Example

Consider a phrase:

he works on macbook pro

The word 'pro' appears in the context of the word 'macbook' but the word vector $\mathbf{w}_{\text{pro}}$ is not the closest to the word vector $\mathbf{w}_{\text{macbook}}$ (see Table 1). Instead, these vectors are split

$$\begin{aligned}\mathbf{w}_{\text{macbook}}^\top &= [\mathbf{x}_{\text{macbook}}^\top; \mathbf{y}_{\text{macbook}}^\top] \\ \mathbf{w}_{\text{pro}}^\top &= [\mathbf{x}_{\text{pro}}^\top; \mathbf{y}_{\text{pro}}^\top]\end{aligned}$$

in such way that the quantity $\mathbf{x}_{\text{macbook}}^\top \mathbf{x}_{\text{pro}} - \mathbf{y}_{\text{macbook}}^\top \mathbf{y}_{\text{pro}}$ is large enough. We can interpret this as follows: the word 'pro' is semantically close enough to the word 'macbook' but is not the closest one: e.g. $\mathbf{w}_{\text{macintosh}}$ is much closer to $\mathbf{w}_{\text{macbook}}$ than $\mathbf{w}_{\text{pro}}$; on the other hand the word 'pro' syntactically fits better being next to the word 'macbook' than 'macintosh', i.e. $-\mathbf{y}_{\text{macbook}}^\top \mathbf{y}_{\text{macintosh}} < -\mathbf{y}_{\text{macbook}}^\top \mathbf{y}_{\text{pro}}$. This combination of semantic proximity ($\mathbf{x}_{\text{macbook}}^\top \mathbf{x}_{\text{pro}}$) and syntactic fit ($-\mathbf{y}_{\text{macbook}}^\top \mathbf{y}_{\text{pro}}$) allows the word 'pro' to appear in the context of the word 'macbook'.

word i	$\mathbf{w}_{\text{macbook}}^\top \mathbf{c}_i$	$\mathbf{w}_{\text{macbook}}^\top \mathbf{w}_i$	$\mathbf{x}_{\text{macbook}}^\top \mathbf{x}_i$	$-\mathbf{y}_{\text{macbook}}^\top \mathbf{y}_i$
macintosh	-0.738 (26th)	18.516 (30th)	8.889 (21st)	-9.449 (59005th)
pro	2.271 (4th)	11.845 (4068th)	7.058 (95th)	-4.787 (1st)

Table 1: Dot products between vectors and their order

4 Experiments

In this section, we empirically verify our hypothesis. We train SGNS with tied weights (Assylbekov and Takhanov, 2019) on two widely-used datasets, `text8` and `enwik9`[1].

1. `http://mattmahoney.net/dc/textdata.html`. The `enwik9` data was processed with the Perl-script WIKIFIL.PL provided on the same webpage.

`Text8` is a 100 MB sample of English Wikipedia text with the total number of words of about 17 million words and 254 thousand unique words. The size of processed `enwik9` is 715 MB and it has 124 million words and 833 thousand unique words.

which gives us word embeddings as well as their partitions:

$$\mathbf{w}_i^\top := [\mathbf{x}_i^\top ; \mathbf{y}_i^\top].$$

The source code that reproduces our experiments is available at `https://github.com/MaxatTezekbayev/Semantics--and-Syntax-related-Subvectors-in-the-Skip-gram-Embeddings`.

4.1 Evaluation of word embeddings

There are two main types of tasks to evaluate "quality" of obtained word embeddings: similarity and analogy tasks. In similarity tasks, we have pairs of words and human-annotated scores for them on some scale, which depends on the annotation procedure of the dataset. We used 4 widely used datasets for similarity tasks: WordSim (Finkelstein et al. 2002), MEN (Bruni et al. 2012), M. Turk (Radinsky et al. 2011), Rare Words (Luong, Socher, and Manning 2013). The samples from the datasets are given in Table 2. For each pair of words (l, m) we can calculate cosine similarity:

$$\text{similarity} = \cos\theta = \frac{\mathbf{w_l} \cdot \mathbf{w_m}}{||\mathbf{w_l}|| \cdot ||\mathbf{w_m}||}$$

Obtaining cosine similarity scores for each pair of words, we can calculate Spearman's rank correlation coefficient between cosine similarity scores and human annotated scores. The results are given in Table 4.

WordSim	MEN	M. Turk	Rare Words
tiger cat 7.35	sun sunlight 50	funeral death 4.714	reasoning deduce 8
tiger tiger 10.00	automobile car 50	albert einstein 4.267	unicycles wheel 6.88
book paper 7.46	river water 49	asia animal 1.555	entrapping deceive 6.57
television radio 6.77	donut panda 3	life death 4.103	gibberish dutch 4
353 pairs	3000 pairs	287 pairs	2034 pairs

Table 2: Samples from similarity task datasets

In addition to similarity task datasets, there are analogy task datasets, where we have 2 pairs of words : $a : a* :: b : b*$ (a is to $a*$ as b is to $b*$). For example, $Tokyo$ is to $Japan$ as $Paris$ is to $France$. There are 2 popular datasets for analogy task: Google analogy test set (Mikolov, Chen, et al. 2013) and MSR[2]. The samples of analogy tasks are given in Table 3. Embeddings are evaluated for its ability to infer the 4th word out from the first three and we use the following well-known model $3CosAdd$:

$$\arg\max_{w \in W} cos(\ w,\ w_{a*} -\ w_a +\ w_b)$$

After choosing the 4th words by $3CosAdd$ we compare them with correct words and calculate accuracy which is a ratio of number of correct predictions to the total number of analogy samples in the dataset.

Google	MSR
athens greece berlin germany	good better rough rougher
denmark krone russia ruble	young younger quick quicker
man woman king queen	goes went feels felt
quick quickly rapid rapidly	tells tell stays stay
19544 analogies	8000 pairs

Table 3: Examples of analogy task datasets

4.2 $\mathbf{x}$-Subvectors Are Related to Semantics

We evaluate the whole vectors $\mathbf{w}_i$'s, as well as the subvectors $\mathbf{x}_i$'s and $\mathbf{y}_i$'s on standard semantic tasks — word similarity and word analogy. We used the HYPERWORDS tool of Levy, Goldberg, and Dagan (2015). The results of the evaluation are provided in Table 4 and Table 5. As one can see, the $\mathbf{x}$-subvectors outperform the whole $\mathbf{w}$-vectors in the similarity tasks and show competitive performance in the analogy tasks. However, the $\mathbf{y}$-parts demonstrate poor performance in these tasks. This shows that the $\mathbf{x}$-subvectors carry more semantic information than the $\mathbf{y}$-subvectors.

2. The dataset is available at: `https://www.microsoft.com/en-us/download/details.aspx?id=52319`

Data	Embeddings	Size	WordSim	MEN	M. Turk	Rare Words
`text8`	$\mathbf{w} := [\mathbf{x}; \mathbf{y}]$	200	.646	.650	.636	.063
	Only $\mathbf{x}$	100	.703	.693	.673	.149
	Only $\mathbf{y}$	100	.310	.102	.193	.019
`enwik9`	$\mathbf{w} := [\mathbf{x}; \mathbf{y}]$	200	.664	.697	.616	.216
	Only $\mathbf{x}$	100	.714	.729	.652	.256
	Only $\mathbf{y}$	100	.320	.188	.196	.091

Table 4: Evaluation of word vectors and subvectors on the similarity tasks

Data	Embeddings	Size	Google	MSR
`text8`	$\mathbf{w} := [\mathbf{x}; \mathbf{y}]$	200	.305	.319
	Only $\mathbf{x}$	100	.348	.213
	Only $\mathbf{y}$	100	.032	.128
`enwik9`	$\mathbf{w} := [\mathbf{x}; \mathbf{y}]$	200	.518	.423
	Only $\mathbf{x}$	100	.545	.303
	Only $\mathbf{y}$	100	.096	.251

Table 5: Evaluation of word vectors and subvectors on the analogy tasks

4.3 $\mathbf{y}$-Subvectors Are Related to Syntax

We train a softmax regression by feeding in the embedding of a current word to predict its part-of-speech (POS) tag:

$$\widehat{\mathrm{POS}}[t] = \mathrm{softmax}(\mathbf{A}\mathbf{w}[t] + \mathbf{b})$$

We evaluate the whole vectors and the subvectors on tagging the Brown corpus with the Universal POS tags from python *nltk* package. There are 12 POS tags: $NOUN$ (noun), $VERB$ (verb), ADP (adposition), DET (determiner), ADJ (adjective), ADV (adverb), $PRON$ (pronoun), $CONJ$ (conjunction), PRT (particle), NUM (numeral), X (other). Words that were not in the vocabulary of $text8$ and $enwik9$ were excluded from the dataset. The resulting accuracies are provided in Table 6.

Embeddings	Size	Trained on `text8`	Trained on `enwik9`
$\mathbf{w} := [\mathbf{x}; \mathbf{y}]$	200	0.9048	.9131
Only $\mathbf{x}$	100	.8099	.8184
Only $\mathbf{y}$	100	.8840	.8927

Table 6: Accuracies on a simplified POS-tagging task.

We can see that the $\mathbf{y}$-subvectors are more suitable for POS-tagging than the $\mathbf{x}$-subvectors, which means than the $\mathbf{y}$-parts carry more syntactic information than the $\mathbf{x}$-parts.

5 Conclusion

Although word embeddings play a crucial role in NLP tasks, there is a lack of interpretability of each dimension of the word embedding and its structure. Using theoretical analysis of word embeddings gives us a better understanding of their properties. Moreover, a theory may provide us interesting hypotheses on the nature and structure of word embeddings, and such hypotheses can be verified empirically as is done in this paper. We empirically showed that there are parts of word embeddings which are more related to the semantic role of the word and parts which are related to the syntactic role of the word. The further step can be identifying these parts of word embedding in original version of SGNS model.

References

Assylbekov, Zhenisbek, and Rustem Takhanov. 2019. “Context Vectors are Reflections of Word Vectors in Half the Dimensions.” *Journal of Artificial Intelligence Research* 66:225–242.

Bruni, Elia, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. 2012. “Distributional semantics in technicolor.” In *Proceedings of ACL,* 136–145. Association for Computational Linguistics.

Finkelstein, Lev, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2002. “Placing search in context: The concept revisited.” *ACM Transactions on information systems* 20 (1): 116–131.

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

Luong, Thang, Richard Socher, and Christopher Manning. 2013. “Better word representations with recursive neural networks for morphology.” In *Proceedings of CoNLL,* 104–113.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. *Efficient Estimation of Word Representations in Vector Space.* arXiv: `1301.3781 [cs.CL]`.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. “Distributed representations of words and phrases and their compositionality.” In *Proceedings of NeurIPS,* 3111–3119.

Radinsky, Kira, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. “A word at a time: computing word relatedness using temporal semantic analysis.” In *Proceedings of the 20th international conference on World wide web,* 337–346. ACM.