Word2Vec

Lexical Resources

November 6^{th}, 2019

Contents of the lecture

Class Project

Class Project

In short

- The project consists in implementing NLP components for an online text editor
- You are to work in pairs. Each of the two students must come from different curricula, preferably the one from the NLP masters, the other not.
- You are to hand in both the adapted code for your components and a document (around 5pp.) detailing your implementation choices, by mail: either to `tmickus@atilf.fr` or to `ccerisara@loria.fr`, before February 12^{th}, 2020.

The code canvas is available here:
https://github.com/TimotheeMickus/lexical-resources-2019

Class Project

Online Text Editor

The general idea is to have you interact with a more or less realistic situation.

- Your work is to implement NLP components to complete the existing application.
- There already exists an application, so you don't have to code everything from scratch. Your code needs to be integrated with the existing code.
- The code was set up so that whatever function written in the relevant python module (`/lexres-site/editor/nlp/services/edits.py` and `/lexres-site/editor/nlp/services/preds.py`) will be executed and processed. Your NLP components should be written as functions in either of these two files.
- You may add supplementary modules if you wish to (eg. to add a specific tokenization module), however you must make sure that this doesn't "break the app".

Start by familiarizing yourself with the code. Make sure you can run the program and test it from your computer. Notify your teachers if you can't make the code run.

Class Project

Your work

The general idea is to have you interact with a more or less realistic situation.

- You should **at least** implement a spellchecker using the `edits.py` module, and a predictive text module using the `preds.py` module
- Optional supplementary components can be **anything**: for instance, you could implement functions in `edits.py` to create a thesaurus, and propose potential synonyms for each word, etc.
- The evaluation will be focused on what you understood from this class, how you adapted lexical resources for this task and how you solved each problem.
- Keep in mind: the document that you have to submit is a **very good place** to make explicit the work you've done.

In the end, what matters most is not how creative you are, but whether you can go from an idea to something concrete.

Word2vec: General Overview

Overview

"You shall know a word by the company it keeps"

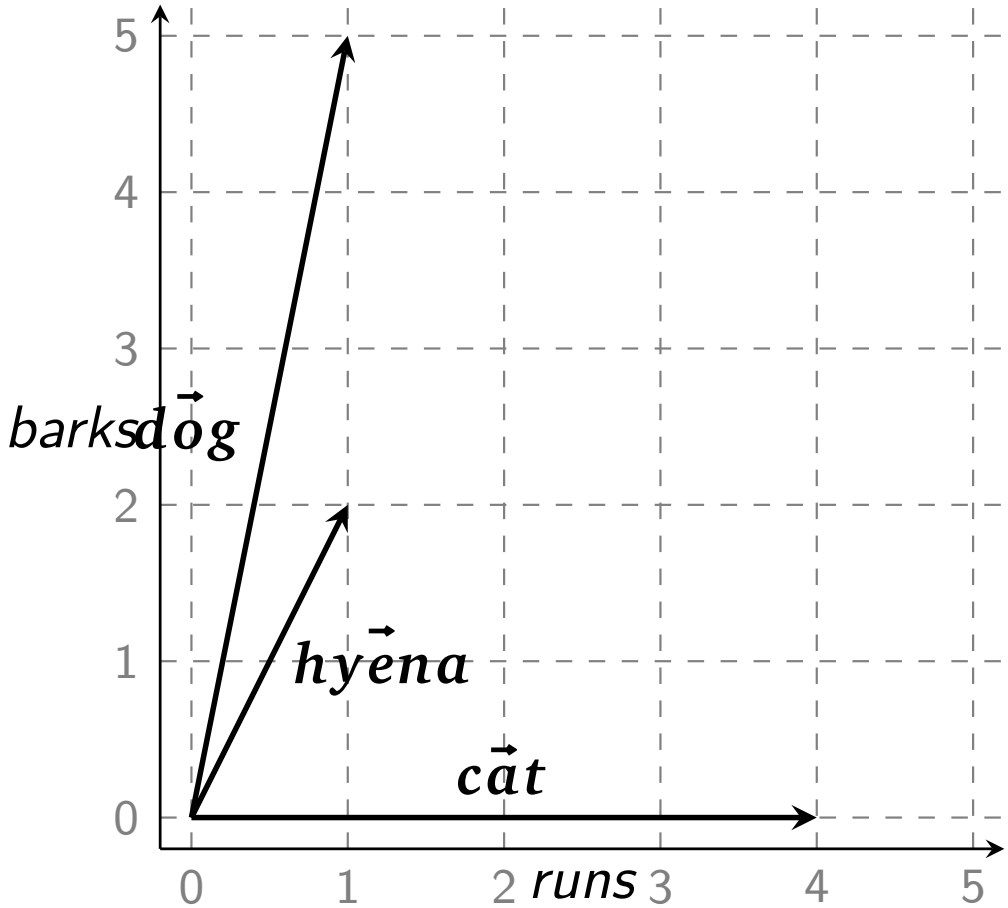
- Word embeddings correspond to the linguistic theory of "distributional semantics" (DS, or distributional semantics models, DSM)
- the general idea of both embeddings and DSM is that the meaning of a word can be known by the context in which it occurs, hence the quote from Firth (1957) : "You shall know a word by the company it keeps"
- Word embeddings are context-based vector representation of words used in machine learning, whereas DS is a semantic theory of meaning, which generally employs vectors to represent meanings.

Overview

"Word vectors"

from Baroni, Bernardi, and Zamparelli (2014)

	runs	*barks*
dog	1	5
hyena	1	2
cat	4	0



Overview

Why does word2vec matter?

Word embeddings in general, and word2vec in particular are widely used in descriptive & theoretical linguistics

- from social biases study (Bolukbasi et al., 2016) ...
- ... to theoretical morphology (Bonami and Paperno, 2018)

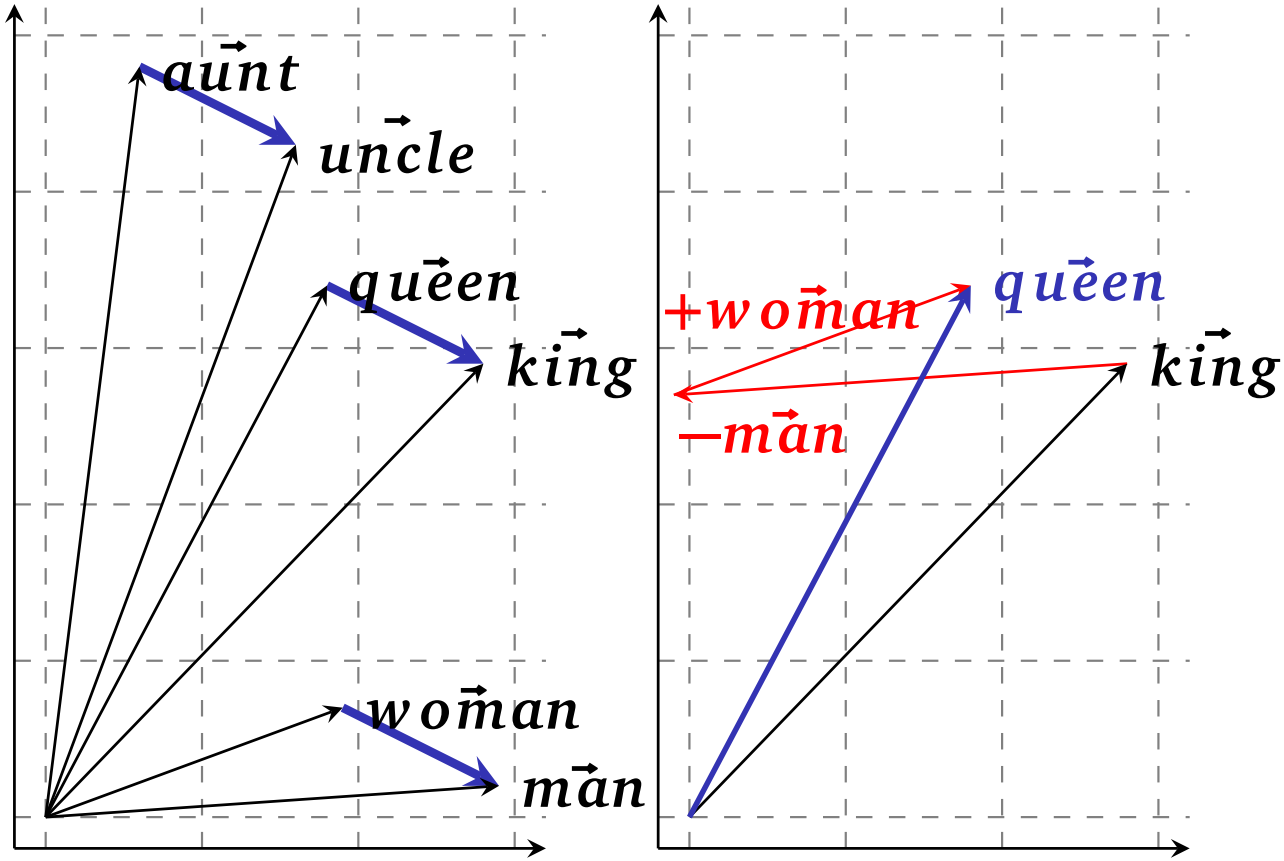
The success of word2vec comes from a multiplicity of factors:

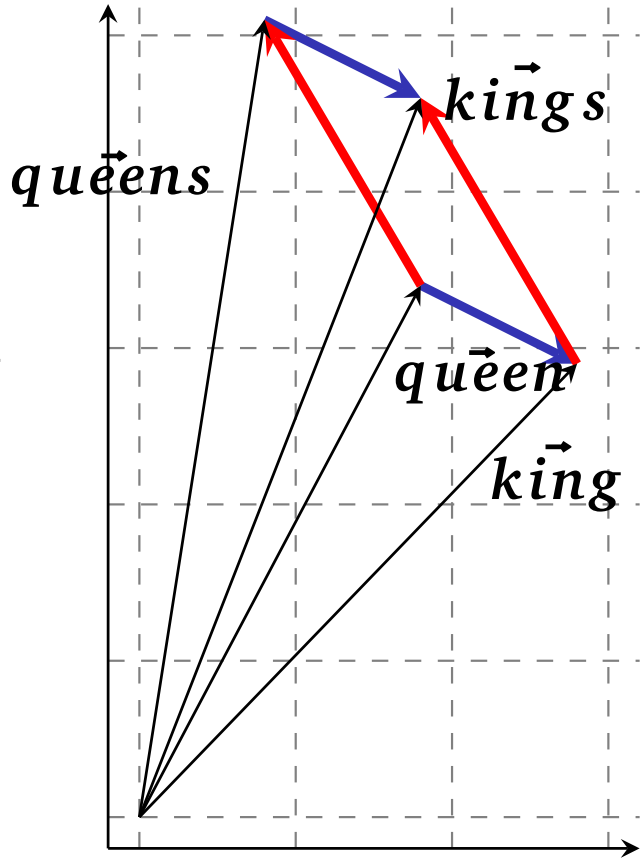
- an efficient algorithm
- useful for initializing other neural networks (Artetxe et al., 2017).
- has been shown to describe a latent code
- highlights how to combine various nifty tricks from the machine learning community

Overview

Formal analogy

Latent code, or latent semantic space of the embedding space: using vector addition, Mikolov, Yih, and Zweig (2013) operationalized formal analogy.



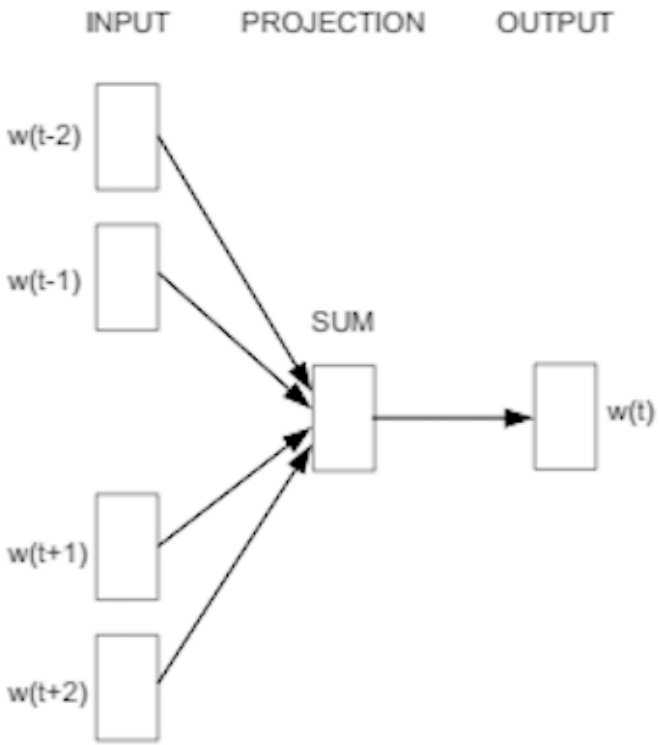


Technical definition of word2vec

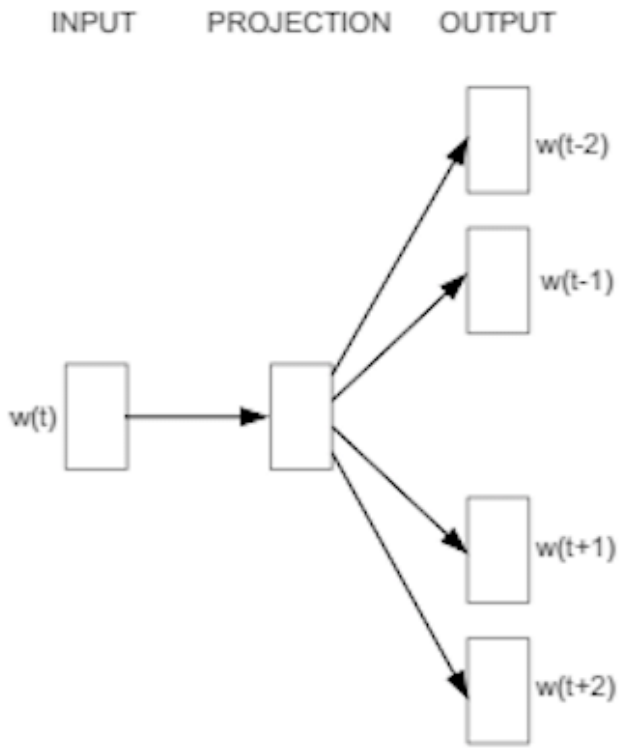
Technical Definition

word2vec comprises 2 architectures

- CBOW uses the context of a word as the input, and tries to predict the word
- Skip-gram uses a word as an input, and tries to predict each word in its context.



CBOW



Skip-gram

Technical Definition

CBOW architecture

- CBOW comprised of one linear projection $\boldsymbol{W_P} = [V \times D]$ and a log-linear classifier $\boldsymbol{W_C} = [D \times V]$,

  V is the size of the vocabulary and D is the number of dimensions.

- All context words are first transformed as one-hot vectors, then down-projected in a vector space R^D using the projection $\boldsymbol{W_P}$. The average of all projected vectors is then used as input for the log-linear classifier $\boldsymbol{W_C}$ itself.

$$\vec{h_i} = \frac{1}{2t} \sum_{j=i-t}^{i-1} \boldsymbol{W_P} \cdot \vec{w}_j + \sum_{j=i+1}^{i+t} \boldsymbol{W_P} \cdot \vec{w}_j$$

$$\hat{y}_i = \text{softmax}(\boldsymbol{W_C} \cdot \vec{h_i})$$

Technical Definition

CBOW word vectors

- The use of one-hot vectors allows us to transform a vocabulary index in a vector.
- Given a word w_i, and its index i in the vocabulary, we define

$$\vec{w}_i = (c_1, \ldots, c_d)$$

$$c_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

- Therefore the down-projection using $\boldsymbol{W_P}$ corresponds to selecting the i$^{\text{th}}$ row of $\boldsymbol{W_P}$.
- The row-vectors of the $\boldsymbol{W_P}$ matrix therefore are the actual word2vec embeddings.
- The classifier $\boldsymbol{W_C}$ only serves for training, and is to be discarded afterwards.

Technical Definition

CBOW training

- As it's a log-classifier, the training objective is to maximize the log-likelihood of the probability of predicting the correct word based on its context.
- The probability distribution is implicitly given by the softmax function:

$$\hat{y}_j = \text{softmax}(\boldsymbol{W_C} \cdot \vec{h})$$

$$= \frac{\exp(\boldsymbol{W_C}^j \cdot \vec{h})}{\sum_{j'} \exp(\boldsymbol{W_C}^{j'} \cdot \vec{h})}$$

  where $\boldsymbol{W_C}^j$ is the j$^{\text{th}}$ column vector of the matrix $\boldsymbol{W_C}$. The components of $\hat{y}$ sum to 1, and therefore define a probability distribution for each element of our vocabulary ($\hat{y}$ is a vector of dimension V).
- maximizing the probability of predicting the current word knowing the context is equivalent to minimizing the negative log-likelihood for that word.

$$\mathscr{L}(\hat{y}, w_i) = -\log \hat{y}_i$$

Technical Definition

Skip-gram architecture

In broad terms, skip gram can be thought of as a "reversed" CBOW architecture

- In skip-gram, we aim predict the context based on the current word.
- We first project the current word using a linear projection $\boldsymbol{W_P} = [V \times D]$, and use a classifier to predict each word in the context $\boldsymbol{W_C} = [D \times V]$. Like with CBOW, we derive vector from the $\boldsymbol{W_P}$ matrix
- A probability distribution is inferred by applying a softmax after the classifier's output.

$$\vec{h_i} = \boldsymbol{W_P} \cdot \vec{w}_i$$

$$\hat{y}_i = \text{softmax}(\boldsymbol{W_C} \cdot \vec{h_i})$$

where $\vec{w}_i$ is the one-hot vector for word w_i.

Technical Definition

Skip-gram Training

- As all context words are to be predicted using the same input word, we aim to maximize the joint probability of all context words knowing the current word.

$$p(w_{i-t}, \ldots, w_{i+t} | w_i)$$

- we can estimate this probability using the chain rule:

$$\prod_{j=i-t}^{i-1} p(w_j | w_i) \times \prod_{j=i+1}^{i+t} p(w_j | w_i)$$

- we can get transform the product into a sum by maximizing the log-likelihood instead
- so the model is trained by minimizing the joint negative log-likelihood of each context word.

$$\mathscr{L}(\hat{y}, \langle w_{i-t}, \ldots, w_{i+t} \rangle) = -\left(\sum_{j=i-t}^{i-1} \log \hat{y}_j + \sum_{j=i+1}^{i+t} \log \hat{y}_j \right)$$

- In practice, the loss is averaged over the full input sentence.

Technical Definition

Negative sampling—new objective

- Obtaining the multinomial distribution of the skip-gram model is computationally inefficient.
- we may instead consider to train the the classifier to distinguish whether a given context is attested for a given word.
- Let D^+ the set of all pairs of words w and contexts c that occurs in our dataset, and let D^- a set of *negative examples* (also pairs of words and contexts), such that $D^+ \cap D^- = \emptyset$. Let $p(X = 1|w, c)$ the probability that $\langle w,\ c\rangle \in D^+$.
- We can redefine the classifier's objective as maximizing $p(X = 1|w, c)$ for all $\langle w,\ c\rangle \in D^+$, and minimizing $p(X = 1|w, c)$ for all $\langle w,\ c\rangle \in D^-$.
- Minimizing $p(X = 1|w, c)$ is equivalent to maximizing $1 - p(X = 1|w, c)$.
- The objective is therefore to maximize

$$\prod_{\langle w,\ c\rangle \in D^+} p(X = 1|w, c) \prod_{\langle w,\ c\rangle \in D^-} \big(1 - p(X = 1|w, c)\big)$$

Technical Definition

Negative sampling—adapting the architecture

- We have to amend the network's architecture. We don't need a full distribution over the vocabulary, so we can replace the softmax function with a sigmoid: $\sigma(y) = \frac{1}{1+\exp(-y)}$. Vector representations for words and contexts still have to be drawn from two different matrices.
- we therefore compute the score for $\langle w_j,\ w_i \rangle$ simply using $\sigma\big(\boldsymbol{W_C}^j \cdot (\boldsymbol{W_P} w_i)\big)$ for pairs drawn from D^+, and $\sigma\big(-\boldsymbol{W_C}^j \cdot (\boldsymbol{W_P} w_i)\big)$ for pairs drawn from D^-, as $1-\sigma(y) = \sigma(-y)$
- To limit computation complexity, we estimate the second term using only k negative examples.
- To obtain the loss function, we replace the negative likelihood of predicting word w_j knowing word w_i from previous loss functions.

$$-\log p(w_j|w_i) = -\log \sigma\big(\boldsymbol{W_C}^j \cdot (\boldsymbol{W_P} w_i)\big) + \sum_{w_n \in N} \sigma\big(-\boldsymbol{W_C}^n \cdot (\boldsymbol{W_P} w_i)\big)$$

where N is a set of k negative examples sampled for w_i.

Technical Definition

Hierarchical softmax & subsampling

- *Alternatively* to negative sampling, we can use a a **hierarchical softmax** and encode probabilities using a binary tree structure. Leaves correspond to words in the vocabulary, and each node n stores the relative probabilities of its children using a dedicated weight vector $\vec{v}_n$.
- Let $\mathscr{P}(w_i) = \{n_0, \ \dots \ , n_{w_i}\}$ the path from the root node n_0 to the leaf node n_{w_i} for word w_i. We can redefine the output probability as

$$p(w_j|\vec{h}_i) = \prod_{n \in \mathscr{P}(w_j)} \sigma\left(\vec{v}_n \cdot \vec{h}_i\right)$$

- Mikolov & al also proposed to avoid issues arising with class imbalance ("Zipf's law") by dropping words from the training set based on their frequency. They define the "**subsampling**" rate:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where t is an hyperparameter (typically 10^{-5}) and $f(w)$ is the frequency of word w.

Conclusion

Conclusion

There has been and there still is an important body of research using word2vec. Here are some things to look into:

- papers introducing word2vec: (Mikolov et al., 2013b), (Mikolov, Yih, and Zweig, 2013), (Mikolov et al., 2013a)
- papers explaining word2vec: (Goldberg and Levy, 2014), (Rong, 2014), (Levy and Goldberg, 2014) ...
- original word2vec repository : https://code.google.com/archive/p/word2vec/, or on Mikolov's github : https://github.com/tmikolov/word2vec
- NLPL's repository containing many pre-trained word2vec models (as well as other embeddings) for multiple languages: http://vectors.nlpl.eu/repository/

References I

Artetxe, Mikel et al. (2017). "UNSUPERVISED NEURAL MACHINE TRANSLATION". In: *arXiv preprint arXiv:1710.11041*.

Baroni, Marco, Raffaella Bernardi, and Roberto Zamparelli (2014). "Frege in Space: A Program for Compositional Distributional Semantics Marco Baroni,1 Raffaella Bernardi1 and Roberto Zamparelli". In: *LiLT Volume 9 Perspectives on Semantic Representations for Textual Inference*.

Bolukbasi, Tolga et al. (2016). "Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., pp. 4349–4357. url: http://papers.nips.cc/paper/6228-man-is-to-computer-programmer-as-woman-is-to-homemaker-debiasing-word-embeddings.pdf.

Bonami, Olivier and Denis Paperno (2018). "A characterisation of the inflection-derivation opposition in a distributional vector space". In: *Lingua e Langaggio*.

Firth, J. R. (1957). "A synopsis of linguistic theory 1930-55.". In: *Studies in Linguistic Analysis (special volume of the Philological Society)* 1952-59, pp. 1–32.

Goldberg, Yoav and Omer Levy (2014). "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *CoRR* abs/1402.3722. arXiv: 1402.3722. url: http://arxiv.org/abs/1402.3722.

References II

Levy, Omer and Yoav Goldberg (2014). "Neural Word Embedding as Implicit Matrix Factorization". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 2177–2185. url: http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf.

Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig (2013). "Linguistic Regularities in Continuous Space Word Representations.". In: *HLT-NAACL*, pp. 746–751.

Mikolov, Tomas et al. (2013a). "Distributed Representations of Words and Phrases and Their Compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., pp. 3111–3119. url: http://dl.acm.org/citation.cfm?id=2999792.2999959.

Mikolov, Tomas et al. (2013b). "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781. arXiv: 1301.3781. url: http://arxiv.org/abs/1301.3781.

Rong, Xin (2014). "word2vec Parameter Learning Explained". In: *CoRR* abs/1411.2738. arXiv: 1411.2738. url: http://arxiv.org/abs/1411.2738.