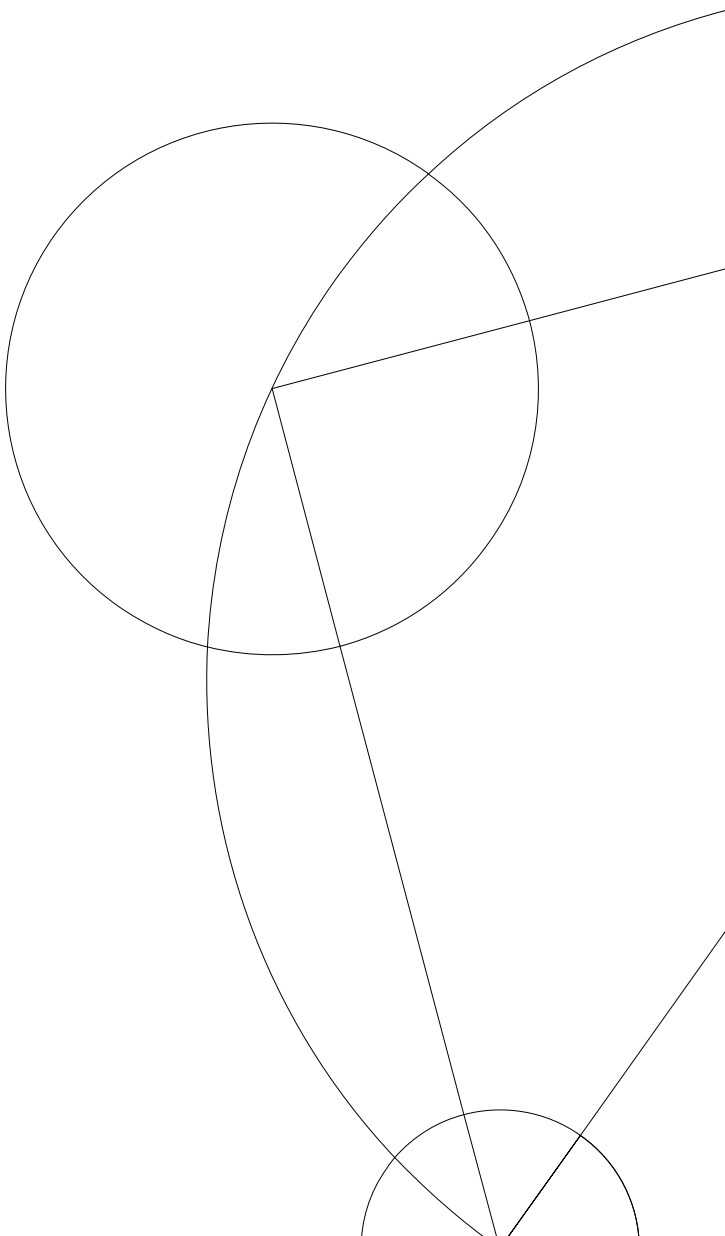
KØBENHAVNS UNIVERSITET

Bachelor thesis

Thor Steen Larsen (mvj665)

Phase Transitions In Word Embeddings

Department of Computer Science



Supervisor: Daniel Hershcovich

18. maj 2020

Abstract

McKinney-Bock and Bedrick (2019) and Hershcovich et al. [2019] found anecdotally that CBOW word embeddings with context window size 1 judge similarity better than those trained with larger windows, but then performance improves gradually until it recovers. This bachelor thesis aims to research why this is happening, and what is changing in this phase transition. We repreduce the results of Hershcovich et al. [2019] using the natural language processing model word-2-vec and look into the word embeddings using explorative data analysis.
Experiment is done using BlazingText Algortihm from AWS SageMaker.

Indhold

1 Introduction

2 Method

We take a data driven inductive approach to testing our hypthesis. We therefore want to recreate the setting where the phase transitions were found by Hershcovich et al. [2019] and Mckinney-Bock and Bedrick [2019].
To reproduce the experiment by Hershcovich et al. [2019], we use the BlazingText algorithm from Amazon SageMaker.
BlazingText is a version optimised version of the Word2Vec Algorithm and can be used for embedding as well as text classifcation [Amazon, 2020].
The Word2vec algorithm maps words to high-quality distributed vectors. The resulting vector representation of a word is the word embedding. Using these embedding we can find which words that are semantically similar. The BlazingText implementation of the Word2vec algorithm is highly optimized for multi-core CPU architectures which makes it fast when given large datasets as input. BlazingText is able to provide both Skip-gram (SGNS) and continuous bag-of-words (CBoW) models (ibid.).

2.1 Dataset and pre-processing

We use fraction of a wiki dump called text8 which has been cleaned to words of the 27 character English alphabet containing only the letters a-z and nonconsecutive spaces. The goal of the authors of the dataset was to only retain text that normally would be visible when displayed on a Wikipedia web page and read by a human. Only regular article text was retained [Mahoney, 2011]. Image captions are retained, but tables and links to foreign language versions were removed. Citations, footnotes, and markup were removed. Hypertext links were converted to ordinary text, retaining only the (visible) anchor text. Numbers are spelled out (”20”becomes ”two zero”, a common practice in speech research). Upper case letters are converted to lower case. Finally, all sequences of characters not in the range a-z are converted to a single space. The Perl script which was used to clean the text can be found here (url).

2.2 CBoW

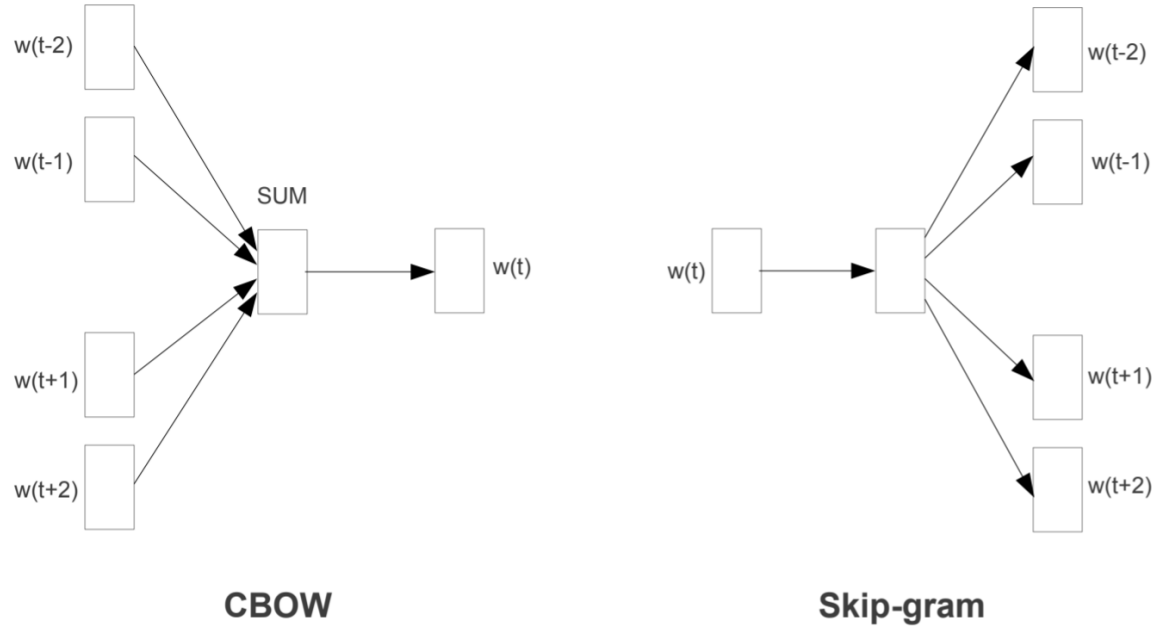


Fig 1: CBoW and Skip-gram architecture [Mikolov et al., 2013a]

In CBoW we use a similar model to a neural network language model to learn the embedding of each words context. The basic CBoW architecture is as following.

As the input layer we have an one-hot encoded input context words represented as the vectors $\{\mathbf{x}_1, \mathbf{x}_C\}$ for a word of size C and vocabulary V depending on the size of the window and vocabulary.

The hidden layer is an N-dimensional vector $\mathbf{h}$ connected to the hidden layer via a $V \times N$ weight matrix $\mathbf{W}$ and the hidden layer is connected to the output layer via a $N \times N$ weight matrix $\mathbf{W'}$.

During forward propagation in the neural network $\mathbf{h}$ is computed by

$$\mathbf{h} = \frac{1}{C}\mathbf{W} \cdot (\sum_{i=1}^{C} \mathbf{x}_i)$$

which is the average of the input vectors weighted by the matrix $\mathbf{W}$. To compute the inputs to each node in the output layer

$$u_j = {v'_{w_j}}^T \cdot \mathbf{h}$$

where v'_{w_j} is the j'th of the output matrix $\mathbf{W}$'.The output y is calculated by passing the input $\mathbf{u}_j$ through the soft max function

$$y_j = p(w_{y_j}|w_1, w_C) = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u'_j)}$$

When we have learned the weight matrices $\mathbf{W}$ and $\mathbf{W}$', we back-propagate in the neural network by using negative logarithmic in an error function Mikolov et al. [2013a]. Such

as

$$
\begin{aligned}
E &= -\log p(w_o|w_1) \\
&= -u_{*j} - \log \sum_{j'=1}^{V} \exp(U_{j'}) \\
&= -\mathbf{v}_{w_0}^T \cdot \mathbf{h} - \log \sum_{j'=1}^{V} \exp(\mathbf{v}_{w_0}^T \cdot \mathbf{h})
\end{aligned}
\tag{1}
$$

Where $*j$ is the index of the the actual output word. The next step is to derive the update equation for the hidden-output layer weights **W**', then derive the weights for the input-hidden layer weights **W** We then update the weights for the input and output hidden layer by using stocastic gradient descent or a similar method to optimize the weights. We can use the final output of a word vector $y \times \mathbf{W}$ through the softmax function to learn the probability of radomly picking a word x nearby any word in our vocabulary V.
In the NNLM, the words are averaged and the projection layer is shared for all the words. This is the reason why CBoW is a bag-of-words model as the order of words in the history does not influence the projection to the output layer and all the word vectors are averaged [Mikolov et al., 2013a]. This computation which happen in the hidden layer is the main difference compared to the Skip-gram model.

2.3 Skip-gram

As in CBoW, we use a simple NNLM to obtain the word embedding. This time using the word to infer the context insted of using the context to infer the word.
The basic Skip-gram architecture is a mirror of the one in CBoW. Given a random word from the context of the window, we built up a data set of word pairs. We then calculate the probabilities of each word pair by feeding it to the hidden layer of a simple neural network in the form of a log-linear classifier with one layer per word in the vocabulary as features. From the hidden layer, we then obtain the weights which times our vector and through a softmax give us the same result as CBoW. Or as put by the authors of Word2Vec, Mikolov et al. [2013a], we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. Furthermore, negative sampling is used to decrease the computational cost need [Mikolov et al., 2013b].

2.4 Window size

We define the windows size to go from 1 - 10 in both our CBoW and Skip-gram model. This give os a total of 20 different models.
The windows size determines what we feed into our model and model complexity due to the vectors direct relations ship with the size of the input word vectors. [Mikolov et al., 2013a] found that increasing the windows size improves quality of the resulting word vectors, but it also increases the computational complexity. To be efficiently get the best results they chose a window size of 10 in their experiment (ibid.).

2.5 Execution

The Models are executed and run in AWS SageMaker's cloud environment (url). Models can and code can be found on GitHub-url.

2.6 Evaluation

In our evaluation as well of testing of the models, we can only use input and output to find out how the model performs.
Each model is evaluated on WordSim353 for similarity and relatedness [Gabrilovich, 2009].

2.6.1 Cosine similarity

To calculate the similarity score between words, we use cosine similarity.
The cosine similarity between two vectors $\mathbf{A}$ and $\mathbf{B}$ can be calculated by

$$\begin{aligned} \mathbf{A} \cdot \mathbf{B} &= \| \mathbf{A} \| \| \mathbf{B} \| \cos\theta \\ \cos(\theta) &= \frac{\mathbf{A} \cdot \mathbf{B}}{\| \mathbf{A} \| \cdot \| \mathbf{B} \|} \end{aligned} \tag{2}$$

The resulting similarity ranges from 1 meaning high dissimilarity, in our case that the words are not related at all, to 1 meaning that the words are exactly the same. A similarity of 0 indicates orthogonality or decorrelation.
If the vectors are normalised by subtracting the vector mean, we get a metric similar to Spearman's ρ which the general correlation of all vectors. This metric is used to evaluate our models in AWS SageMaker.

To dos:

1. write more on WordSim353
2. Make plot of mean rho values of new models
3. Describe results
4. Discuss results compared to Hershcovich et al. [2019]
5. set up in theory/background, method, results, discussion section

3 Appendix

Litteratur

Amazon. Blazing documentation, 2020. URL https://docs.aws.amazon.com/sagemaker/latest/dg/blazingtext.html.

E. Gabrilovich. Wordsim353 - similarity and relatedness, 2009. URL http://alfonseca.org/eng/research/wordsim353.html.

Daniel Hershcovich, Assaf Toledo, Alon Halfon, and Noam Slonim. Syntactic interchangeability in word embedding models. pages 70–76, 2019. URL https://www.aclweb.org/anthology/W19-2009.

Matt Mahoney. About the test data, 9 2011. URL http://mattmahoney.net/dc/textdata.html. text8 can found on http://mattmahoney.net/dc/text8.zip.

Katy Mckinney-Bock and Steven Bedrick. Classification of semantic paraphasias: Optimization of a word embedding model. *Proceedings of the 3rd Workshop on Evaluating Vector Space Representations for*, 2019. doi: 10.18653/v1/w19-2007.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv.org*, 2013a. URL http://search.proquest.com/docview/2086087644/.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv.org*, 2013b. URL http://search.proquest.com/docview/2085905727/.