KØBENHAVNS UNIVERSITET

Bachelor thesis

Thor Steen Larsen (mvj665)

Phase Transitions In Word Embeddings

Department of Computer Science

Supervisor: Daniel Hershcovich

25. 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 Theory

1.1 Statistical estimators of Language Models

By looking at a stream of words with relations to each other, we have in general a certain number of words that fall in to a certain bin of word relations. By calculating the probability that a word fall into such a bin, we can derive a good statistical estimator of word relations. Or put in practical terms, given a training set, we wish to calculate the features of such a word. This feature is in most cases, given a word what is the probability of the next word. To simply this notation, we use n-grams. Using a word n-gram, that is a contiguous sequence of n items from a given sample of text $w_1, \dots w_n$, we are interested in the prediction task:

$$P(w_n|w_1 \dots w_{n-1}) = \frac{P(w_1 \dots w_n)}{P(w_1 \dots w_{n-1})}$$

where P is the probability function of w conditioned on the word n-gram w_{n-1}. Here we assume the Markov property that the probability of one word affects the probability of the next word.
Using a frequentist approach, we can calculate P and find the Maximum Likelihood Estimator from the relative frequencies of the training data. Relative frequencies can be found by calculating how many instances of word in the n-gram given a certain corpus. Where $C(w_1 \dots w_n)$ represents the frequency of the n-gram $w_1 \dots w_n$ in the training text or corpus, we can calculate the MLE by

$$P_{MLE}(w_n|w_1 \dots w_{n-1}) = \frac{C(w_1 \dots w_n)}{C(w_1 \dots w_{n-1})}$$

The parameters of the MLE will be the words with the highest probability of being in the context of the word of interest.

1.2 CBoW

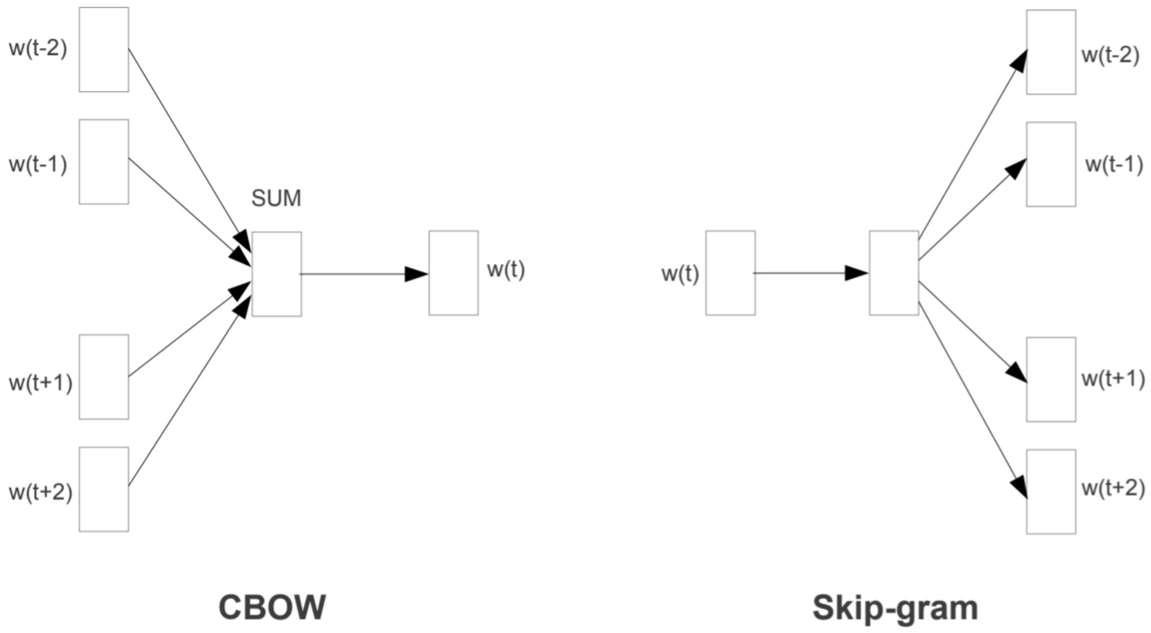


Fig 1: CBoW and Skip-gram architecture [Mikolov et al., 2013a]

In CBoW, we take some the ideas from the previous sub-section and use in a similar model to a neural network language model to learn the embedding of each word given its n-gram. More precisely, we from the middle word w_0 want to predict this words context w_1. The basic CBoW architecture is as following.
As the input layer we have an one-hot encoded input context words represented as the vectors $\{\mathbf{x}_1, \ldots \mathbf{x}_C\}$ 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 **h** connected to the hidden layer via a $V \times N$ weight matrix **W** and the hidden layer is connected to the output layer via a $N \times N$ weight matrix **W'**.
During forward propagation in the neural network **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 **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 **W**'.The output y is calculated by passing the input $\mathbf{u}_j$ through the soft max function

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

When we have learned the weight matrices **W** and **W**', we back-propagate in the neural network by using negative logarithmic in an error function Mikolov et al. [2013a]. Similar to the MLE, the objective is to maximise the conditional probability of a output word given a input context, therefore our loss function will be

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

Where $*j$ is the index of the the actual output word, w_O is the middle output word and w_I is the previous/next input 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 stochastic 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 randomly 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.

1.3 Skip-gram

As in CBoW, we use a simple NNLM to obtain the word embedding. This time using a middle input word to infer a output of previous/next words.
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.
In addition to the general softmax function Mikolov et al. [2013b] implements a hierachical softmax function to optimize the computation. This implementation uses a binary tree representation of the output layer with V words as its leaves and for each node the relative probabiliites of its child nodes. This gives a random walk that assigns probabilities to the words. Furthermore, negative sampling can be used to decrease the computational cost [Mikolov et al., 2013b].

1.4 Cosine similarity

To calculate the similarity score between words, we use cosine similarity.
The cosine similarity between two vectors **A** and **B** can be calculated by

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

Cosine similarity captures the angle of the word vectors which mean that a high similarity score of 1 is equivalent to a 0 degree difference between the two vectors and none-similarity score of 0 means a 90 degree difference between the vectors. This make us able to avoid using the magnitude as by euclidean distance to compare the word vectors.

1.5 Spearman's rank correlation coefficient

To evaluate our models on a test data-set in AWS SageMaker and Gensim, we furthermore calculate Spearman's ρ of the word rankings. In our case the words are ranked by cosine similarity from high to low where the highest cosine similarity will be of the most similar word in the n-gram.
Spearman ρ can be used as a statistical test to determine if there exists a relation between two random variables X and Y. The test can be a bilateral test or a unilateral test [Dodge, 2008].
Spearman's ρ, where ρ is Pearson correlation coefficient, and $rank_X, rank_Y$ are the ranking of the two random variables can be written as

$$\rho_{rank_X, rank_Y} = \frac{cov(rank_X, rank_Y)}{\sigma_{rank_X}, \sigma_{rank_Y}}$$

Identical values are usually each assigned fractional ranks equal to the average of their positions in the ascending order of the values, which is equivalent to averaging over all possible permutations [Dodge, 2008].

2 Method

We take a data driven inductive approach to testing our hypothesis. 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 and a similar implementation of Word2Vec from Gensim.
BlazingText is a version optimised version of the Word2Vec Algorithm and can be used for embedding as well as text classifcation [Amazon, 2020].
Gensim's Word2Vec is based on the orginal C implementation by Mikolov et al. [2013a] of Word2Vec [Rehurek, 2019]. 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 and Gensim.Word2Vec are highly optimized for multi-core CPU architectures which makes it fast when given several million of words. Furthermore, they are both 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 data-set 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 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.).

3 Results

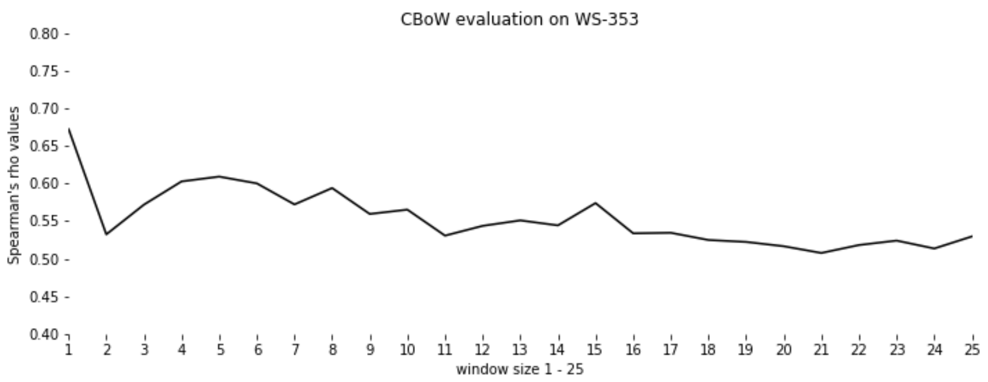
3.1 Execution

The models are trained and evaluated in AWS SageMaker's cloud environment (url) and on a personal multi-core computer. Models, data and the used code written in Python can be found on GitHub-url.

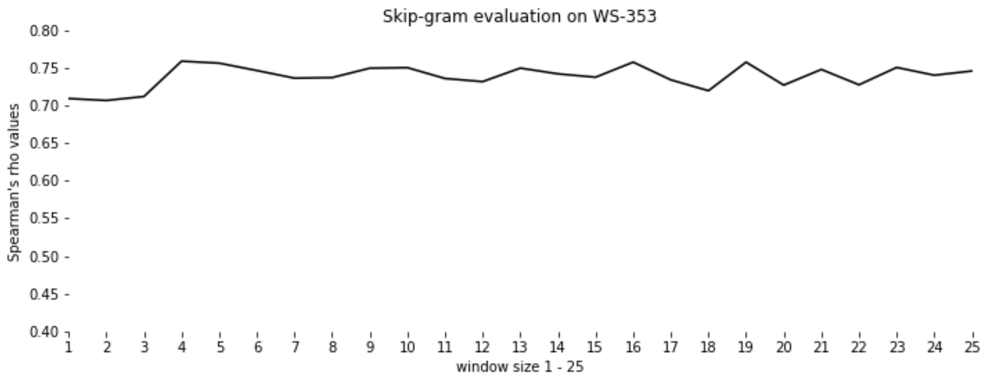
3.2 Evaluation

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 which is a dat aset of word which is human annotated [Gabrilovich, 2009]. The same data set and approach is used in Hershcovich et al. [2019] which we wish to compare with our own results.

Depending on the algorithm we use, we get different results when evaluating on WS-353. Using different parameters with BlazingText and a changing window size, we are not able to reproduce the finds of Hershcovich et al. [2019] and Mckinney-Bock and Bedrick [2019].
When using the Word2Vec implementation found in Gensim by Radimrehurek, which is based on the original implementation by Mikolov et al. [2013a] in C, we are though able to reproduce the results.



As found in Hershcovich et al. [2019] the difference between window size 1 and 2 is smaller when using the Skip-gram algorithm.



4 Appendix

Litteratur

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

Yadolah. Dodge. *The Concise Encyclopedia of Statistics.* Springer reference. Springer New York, New York, NY, 1st. ed. edition, 2008. ISBN 0-387-32833-5.

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/.

Radim Rehurek. Gensim.word2vec documentation, 2019. URL https://radimrehurek.com/gensim/models/word2vec.html.