Textual embeddings with word-type-weighted word2vec

Theodor Ladin[1], Lukáš Korel[2] and Martin Holeňa[2,3]

[1]Gymnázium Nad Štolou, Prague, Czech Republic
[2]Faculty of Information Technology, CTU, Prague, Czech Republic
[3]Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic

Abstract

The increasing use of artificial neural networks for knowledge processing often lacks precise knowledge representation. To address this issue, we propose using a word-type-weighted Word2Vec model to achieve more accurate representations of individual words within sentences. Our approach incorporates weighting vector embeddings of words based on parts-of-speech predictions generated by the spaCy library. Experimental results demonstrate that, compared to simple Word2Vec, our model enhances the accuracy of recognizing the semantics of a sentence, while maintaining significantly lower computational requirements than large language models and various variants of Transformer.

Keywords

text representation learning, text embedding, text preprocessing, word2vec,

1. Introduction

Recently, artificial intelligence (AI) and machine learning (ML) have proved to be extremely useful in most scientific fields [1, 2, 3]. Neural networks have been shown to be a very powerful tool in text analysis, predictive analytics, image recognition, and many other areas, but they lack in one respect – the processing accessibility, with most neural networks for text analysis needing supercomputers for their training [4]. This creates a problem, if we want to use a low processing cost program to determine the semantic similarity of sentences. For such situations, we tried to come up with a solution explained in this paper.

The main objective of this research is to develop a lightweight algorithm for correctly predicting sentence similarity that utilizes text representation only on the word level, i.e., word embeddings solely at the word level and parts-of-speech (POS) information. By integrating a word-type-weighted Word2Vec (W2V) [5] model with POS tagging, our approach aims to provide a low-cost alternative to large text embedding models based on transformers which often require high-performance accelerators. In our test case with processor i7-12650H and memory 2×16 GB DDR5 at 4800 MHz, we have achieved approximately 170 sentences/s with sentence transformer and 15 500 sentences/s with W2V.

The following section explains the concept of sentence embeddings and its applicability. Section 3 describes the methodology used to find the optimal weights and introduces the tools used in this task, the text pre-processing, and the weighting approach. Finally, Section 4 presents experimental results in comparison with other existing approaches.

2. Applicability of Sentence Embeddings

Textual embedding is a useful tool in NLP (natural language processing). It is a vector representation of text that helps to capture the meaning of sentences[6]. This makes it valuable for many tasks. For example, in text classification, such as sentiment analysis, sentence embeddings help determine if a sentence is positive, negative, or neutral. It is also useful in topic classification, where it helps to sort text into categories like sports, politics, or technology.

Sentence embeddings are naturally suitable for finding semantic similarities between sentences. They help in tasks such as paraphrase detection, where the goal is to find sentences with basically the same meaning. Another important application is in information retrieval, sentence embeddings improve search results by finding documents that match a query more accurately [7]. They are also used in text summarization by picking out the most important sentences. Overall, sentence embeddings make working with text easier and more effective in many applications.

ITAT'24: Information technologies – Applications and Theory, September 20–24, 2024, Javorna, Slovakia
theodor.lagin@gmail.com (T. Ladin); lukas.korel@fit.cvut.cz (L. Korel); martin@cs.cas.cz (M. Holeňa)

3. Methodology

3.1. Overview

This section outlines the methodology used to develop the word-type weighted Word2Vec model, used to predict semantic similarity of sentences. Our approach integrates word embeddings with parts-of-speech information to improve accuracy without large processing costs.

3.2. Employed Tools

The corpus we used was the Microsoft Research paraphrase corpus[8]. It contains around 5800 pairs of sentences. We trained the algorithm on the train set of this corpus and tested it on its test set.

We used the public GoogleNews-vectors-negative300[9] Word2Vec implementation, for more objective and clear results, because of how widespread this corpus is. The model utilizes 300-dimensional vectors and has been trained with around 3 million different English words. The size of this model is around 1,6 GB.

We used the spaCy library [10] for POS (Parts-of-Speech) tagging, because of its efficiency and precision, which is crucial to fine-tuning the weights correctly.

3.3. Text Preprocessing

3.3.1. Standard Preprocessing

As the first step of preprocessing, we use the spaCy library to tag each word in a sentence, which as a result also tokenizes the given sentence. SpaCy assigns tags automatically, using a neural network. Then we delete all the symbols. After deleting the symbols, we apply a standard spell-checking algorithm to correct the mistakes created by deleting the symbols. After that we employ our embedding algorithm.

This embedding algorithm starts by verifying that the word is not a stop word. If it passes this check, we clarify whether the word is present in our model. If the word is absent, we proceed to lemmatization and check again, followed by stemming and another check. If all of these steps are unsuccessful, we assign to each token the embedding based on its assigned tag. For instance, the embedding of John is assigned to every first name tagged as a proper noun because there are missing embeddings for them.

3.4. Weights

In this study, we consider weights for each word type, denoted as w_{wt}, where wt is the index of the word type. For each w_{wt}, we assume that $w_{\text{wt}} \in \mathbb{Q}$.

3.4.1. Text Preparation

We first divided the training text into two parts. First being 60 percent of the text and second being 40 percent of the text. We then used our text preprocessor to vector these parts of text. Both parts were made up of pairs of sentences, where half of them had the same semantic similarity and half did not.

3.4.2. Initial Weight Optimization

Initially, we needed to make a sufficiently accurate guess close to the global minimum. To achieve this, we used the Broyden-Fletcher-Goldfarb-Shanno algorithm, also known as the BFGS method, to minimize the mean squared error [11]. We opted for this method because, when tested, it was shown to be the most accurate for this specific type of problem.

The BFGS algorithm is an iterative method for solving unconstrained nonlinear optimization problems. It belongs to the family of quasi-Newton methods, which are used to find local maxima or minima of functions. The key idea behind BFGS is to update an approximation to the Hessian matrix (or its inverse) at each iteration to improve the convergence rate.

The BFGS update formula for the inverse Hessian matrix H_{k+1} is given by:

$$H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k} \quad (1)$$

where:

- H_k is the approximation of the inverse Hessian matrix at iteration k.
- $s_k = x_{k+1} - x_k$ is the change in the vector of variables.
- $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ is the change in the gradient of the objective function.
- I is the identity matrix.

The BFGS algorithm uses this updating formula iteratively to improve the approximation of the inverse Hessian matrix, ultimately aiding in the efficient optimization of the objective function.

This function by preconditioning the gradient determines the descent direction, towards the local minimum for each weight. The error or the loss function was computed as the difference between the target similarity, which could be either -1 or 1, and the cosine similarity between embeddings. This process polarized the weights, making them highly effective as an initial guess. We also tried iterative weight adaptation without an initial weight guess, but it would take too many iterations to produce any meaningful guess, and fewer iterations did not yield any results.

Table 1
Example output from iterations for each word type

Word type	Abbreviation	Weight - 1st iteration	Weight - 2nd iteration	Example word
Adjective	ADJ	1.000	1.000	last
Adposition	ADP	0.210	0.238	across
Adverb	ADV	0.903	1.066	separately
Auxiliary	AUX	0.415	0.396	would
Coordinating Conjunction	CCONJ	0.020	0.007	either
Determiner	DET	0.071	0.080	every
Interjection	INTJ	0.020	-0.006	oh
Noun	NOUN	-6.150	-6.651	brother
Numeral	NUM	3.470	4.467	five
Particle	PART	0.095	0.037	nt
Pronoun	PRON	0.085	0.100	somebody
Proper Noun	PROPN	-0.011	-0.585	Amrozi
Subordinating Conjunction	SCONJ	0.119	0.112	since
Verb	VERB	3.514	4.204	reported

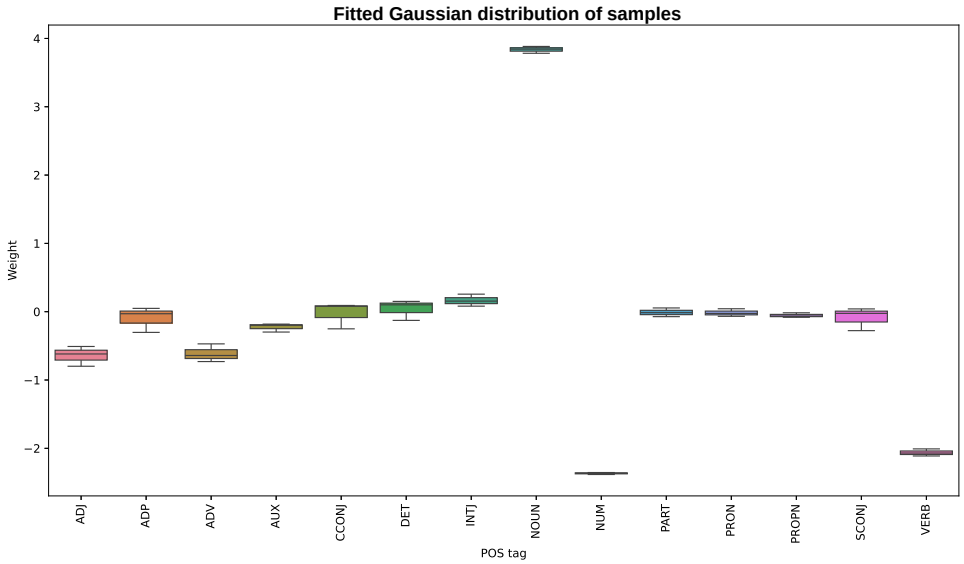

Figure 1: Fitted Gaussian distribution of samples

3.4.3. Gaussian Distribution

The BFGS method was quite dependent on the initial conditions, and hence we did a number of iterations of this function while changing the text that was supposed to be similar or not. Afterward, we fitted a Gaussian distribution on the given ratio between weights because we considered the ratio more important than the finalized weights. Figure 1 depicts the weight ratios obtained in Table 1. We normalized the overall distribution around zero.

3.4.4. Final Weights Optimization

Subsequently, we generated random samples from the obtained Gaussian distribution. These samples were generally similar (Figure 1), although there were a few exceptions, such as with nouns, created from the larger ratio differences.

Although some estimates were worse than others, all the differences could be rectified, with the method we employed at last. We refined the weights, that were different, through an iterative process, comparing them with weights derived from the Gaussian distribution with small enough differences. The refinement was

achieved by minimizing logistic loss using the Nelder-Mead method. The logistic loss was calculated based on the prediction accuracy.

The logistic loss for a binary classification problem, also known as log-loss or binary cross-entropy loss, is given by:

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \sigma(\mathbf{x}_i \cdot \mathbf{w}) + (1-y_i)\log(1-\sigma(\mathbf{x}_i \cdot \mathbf{w}))] \quad (2)$$

where:

- N is the number of samples,
- y_i is the true label (0 or 1) for the i-th sample,
- $\mathbf{x}_i$ is the feature vector for the i-th sample,
- $\mathbf{w}$ is the weight vector,
- $\sigma(z)$ is the sigmoid function defined as $\sigma(z) = \frac{1}{1+e^{-z}}$.

The Nelder-Mead algorithm minimizes the logistic loss function by iteratively refining a simplex with $n+1$ vertices in an n-dimensional space [12]. The Nelder-Mead method is particularly effective for optimizing the logistic loss function in logistic regression, especially in cases where the gradient is unavailable or the function is non-smooth. Through successive adjustments of the simplex vertices via reflection, expansion, contraction, and shrinkage, the algorithm steadily progresses toward the minimum of the logistic loss function.

3.4.5. Embedding Correction

Embeddings were too dependent on the length of their sentences. We have created a gradient-based weight creator, which modifies the embedding. It adds a corrector multiplied by the count of tokens in the sentence. We chose to use the additive weighted count of tokens in a sentence because, after many tests with different corrections, such as modification by the count of particular word types and multiplication with a weighted count of tokens, it was shown to be the most differentiating factor between different sentences.

3.5. Full experimental setup

In table 2, you can see the full experimental setup of the methodology.

Table 2
Summary of Experimental Setup

Category	Our choice
Dataset	Microsoft Research Paper
Minimization Algorithms	BFGS, Nelder-Mead
Error Functions	Logistic Loss, Mean Squared Error
Assumed Distribution	Gaussian Distribution
Evaluation Metrics	Accuracy, F1-Score, AUC
Number of Executions	10
Training-Testing Set Ratio	60 % : 40 %

4. Results

4.1. Final Weights

The resulting final weights were in some cases negative, with nouns being overly positive. Adjectives, nouns, numerals, and verbs had the largest weights, while other parts of speech, for instance determiners or adpositions had weights close to zero. This most likely happened due to these POS having such large impact on sentences. The final weights are shown in table 3.

Table 3
Example of final word type weights, other types were equal to 0.060, but this value almost does not affect results, because the occurrence of the other types is very rare. The final token-based embedding corrector $w_{ec} = -0.028$.

Word type abbreviation	Weight
ADJ	-1.330
ADP	0.341
ADV	-0.616
AUX	-0.334
CCONJ	0.126
DET	0.308
INTJ	-0.143
NOUN	4.970
NUM	-2.829
PART	-0.396
PRON	-0.060
PROPN	0.068
SCONJ	-0.011
VERB	-2.656

4.2. Classification

We have compared our approach to the BERT (Bidirectional Encoder Representations from Transformers) [13] fine-tuned for sentence embeddings, namely **all-MiniLM-L12-v2** which has good benchmark results[1] and simple averaging Word2Vec without weighting. All results in this test have been obtained from the independent testing dataset. The testing dataset is balanced to contain the same number of records for each class (the same and different descriptions). We used the Accuracy, F1 score, and AUC[14] for measuring all the statistics.

[1] benchmark results of available sentence transformers: https://www.sbert.net/docs/sentence_transformer/pretrained_models.html

Table 4
Results obtained on the balanced testing dataset. The best results have been achieved by the BERT, which is based on a neural network that has been trained on large amounts of data and requires high-power computing units to perform embedding fast. When we compare the simple Word2vec approach, the word type weighting aggregation brings much better results for sentence embedding in all considered metrics.

Quality measure	**Accuracy**	**F1 score**	**AUC**
BERT	0.975767	0.975165	0.975767
W2Vmean	0.806442	0.837831	0.806442
W2Vweighted	0.933742	0.929273	0.933742

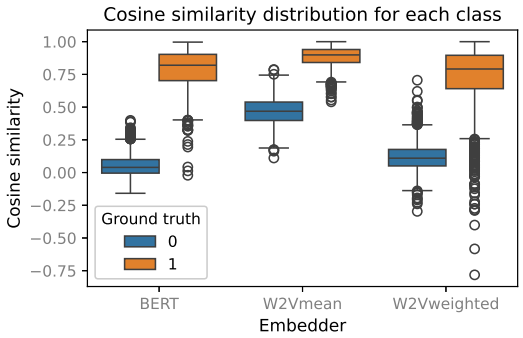

Figure 2: The distribution of results from BERT, mean aggregated Word2Vec and our solution grouped by ground truth similarity.

The results are represented in Table 4 and the box plot Figure 2. The weighted solution by word type brings much better results than simple averaging. The weighted solution has a higher margin between similar and dissimilar sentences, but not as high as the BERT. The high performance is probably caused by its architecture, training data, and contextual processing of the whole input.

The differences between the considered embedders were tested for significance by the Friedman test. The basic null hypothesis that the results for all 3 embedders coincide was strongly rejected, with the achieved significance $p = 1.39 \times 10^{-297}$. For the post-hoc analysis, we employed the Wilcoxon signed rank test with the two-sided alternative for all pairs of the compared embedders, because of the inconsistency of the more common mean ranks post-hoc test with the missing closed-world assumption in machine learning, as pointed out in [15]. For correction to multiple hypotheses testing, we used the Holm method, which yielded the following corrected results:

- BERT vs. W2Vmean: $p = 4.01 \times 10^{-156}$
- BERT vs. W2Vweighted: $p = 2.12 \times 10^{-16}$
- W2Vmean vs. W2Vweighted: $p = 1.46 \times 10^{-183}$

5. Conclusion

This paper introduces word-type-weighted Word2Vec for sentence embeddings. It is based on Word2Vec and aggregates words from a given sentence by the pre-trained weights into one numeric vector. Our weighted Word2Vec embedder was compared on testing data with average aggregation and with the BERT. The tested task was about recognizing whether the given pair of sentences is paraphrased with the same meaning or sentences with different meanings. The complex neural network architecture of the BERT outperformed our solution, but the simple averaging without weighting had a much narrower gap between target classes in our testing case. The advantage of our solution is using the simple Word2Vec model.

In future research, we would like to extend our solution to embed whole paragraphs. We also want to consider other word-based embedders.

Acknowledgments

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS23/205/OHK3/3T/18 and by the German Research Foundation (DFG) funded project 467401796.

References

[1] C. M. Bishop, Pattern recognition and machine learning, volume Information science and statistics, Springer, Oxford, 2006.

[2] T. M. Mitchell, Machine learning, volume McGraw-Hill series in computer science, international ed ed., McGraw-Hill, new York, 1997.

[3] S. Marsland, Machine learning: an algorithmic perspective, volume Chapman&Hall/CRC machine learning&pattern recognition series, second edition ed., Chapman & Hall/CRC, Boca Raton, FL, 2014.

[4] O. Suissa, A. Elmalech, M. Zhitomirsky-Geffet, Text analysis using deep neural networks in digital humanities and information science, Journal of the Association for Information Science and Technology 73 (2022) 268–287. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.24544. doi:https://doi.org/10.1002/asi.24544.

[5] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, Advances in Neural Information Processing Systems 26 (2013).

[6] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad Khasmakhi, M. Asgari-Chenaghlu, J. Gao, Deep learning based text classification: A comprehensive review, 2020.

[7] M. Zhou, D. Liu, Y. Zheng, Q. Zhu, P. Guo, A text sentiment classification model using double word embedding methods, Multimedia Tools and Applications 81 (2022) 18993–19012. URL: https://doi.org/10.1007/s11042-020-09846-x. doi:10.1007/s11042-020-09846-x.

[8] W. B. Dolan, C. Brockett, Microsoft research paraphrase corpus, Microsoft Research, 2005. URL: https://www.microsoft.com/en-us/download/details.aspx?id=52398, accessed: August 13, 2024.

[9] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781 (2013). URL: https://github.com/mmihaltz/word2vec-GoogleNews-vectors, accessed: August 13, 2024.

[10] M. Honnibal, I. Montani, S. Van Landeghem, A. Boyd, spaCy: Industrial-strength Natural Language Processing in Python (2020). doi:10.5281/zenodo.1212303.

[11] C. T. Kelley, Iterative Methods for Optimization, SIAM, 1999, pp. 71–86. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970920.ch4. doi:10.1137/1.9781611970920.ch4.

[12] J. A. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 7 (1965) 308–313. URL: https://academic.oup.com/comjnl/article/7/4/308/354237. doi:10.1093/comjnl/7.4.308.

[13] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. URL: https://aclanthology.org/N19-1423. doi:10.18653/v1/N19-1423.

[14] C. Ferri, J. Hernández-Orallo, R. Modroiu, Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation, Springer, 2009.

[15] A. Benavoli, G. Corani, F. Mangili, Should we really use post-hoc tests based on mean-ranks?, Journal of Machine Learning Research 17 (2016) 1–10. URL: http://jmlr.org/papers/v17/benavoli16a.html.