Evaluating the performance and improving the usability of parallel and distributed Word Embeddings tools

Matheus L. da Silva, Vinícius Meyer, Dionatrã F. Kirchoff, Joaquim F. S. Neto,
Renata Vieira and César A. F. De Rose
Polytechnic School, *Pontifical Catholic University of Rio Grande do Sul (PUCRS)*
Ipiranga Avenue 6681, 90619-900 Porto Alegre, RS, Brazil
{matheus.lyra, vinicius.meyer}@edu.pucrs.br, {dionatra.kirchoff, joaquim.santos}@acad.pucrs.br,
{renata.vieira, cesar.derose}@pucrs.br

***Abstract*—The representation of words by means of vectors, also called Word Embeddings (WE), has been receiving great attention from the Natural Language Processing (NLP) field. WE models are able to express syntactic and semantic similarities, as well as relationships and contexts of words within a given corpus. Although the most popular implementations of WE algorithms present low scalability, there are new approaches that apply High-Performance Computing (HPC) techniques. This is an opportunity for an analysis of the main differences among the existing implementations, based on performance and scalability metrics. In this paper, we present a study which addresses resource utilization and performance aspects of known WE algorithms found in the literature. To improve scalability and usability we propose a wrapper library for local and remote execution environments that contains a set of optimizations such as the pWord2vec, pWord2vec_MPI, Wang2vec and the original Word2vec algorithm. Utilizing these optimizations it is possible to achieve an average performance gain of 15x for multicores and 105x for multinodes compared to the original version. There is also a big reduction in the memory footprint compared to the most popular python versions.**

***Index Terms*—Natural Language Processing, Word2vec, Performance Evaluation.**

I. Introduction

The language representation model Word Embedding (WE) has gained visibility in the natural language processing (NLP) field. This kind of model enables findings such as syntactic and semantic similarities, as well as relations and word contexts within a given corpus (input text). The neural networks algorithms identify linguistic patterns and allow algebric operations between the words' corresponding vectors of the WE models. For example:

$$[Madrid] - [Spain] + [France] \simeq [Paris] \tag{1}$$

Where $[Paris]$ would have the vector that most approximates the operation in Equation 1 [1].

These WE models can be applied in different fields, such as the forensic science by extracting texts from documents related to criminal investigations. In this case, the models can be used as an aid for investigators to identify patterns and associated words. Also, they can be applied to associate people with places, performing researches at suspicious activities.

Word2vec, introduced by Mikolov et al. [1], is a WE algorithm widely used in the NLP field. Their implementation has scalability problems, which means that it does not improve performance based on the amount of resources available. Thus, it requires several hours to complete its executions, which can be seen in Section IV. The scalability problem remains in newer versions of algorithms based on Miklov's Word2vec. However, there are opportunities for improvements by applying High-Performance Computing (HPC) solutions, which are based on parallel computing and provide performance and scalability by splitting the problem in multiple statements concurrently processed.

Similar to Word2vec there are other WE algorithms, e.g. FastText and Wang2vec. These are capable of generating vector representations from larger corpora using different training strategies and consequently producing language models with particular characteristics.

Besides the above mentioned versions, there are other optimizations that are able to achieve better results in terms of performance and resource utilization. However, suffer from limited usability since they are not in the most popular tools for the application of these algorithms, e.g., Gensim [2], NLPNET [3] and spaCy [4].

This paper presents an analysis of the main differences between the above-mentioned implementations, considering points directly related to performance and scalability. We investigate the established WE NLP implementations and the optimizations proposed by the HPC field aiming to provide insights about the resource utilization and performance issues of the different analyzed versions.

The main contributions of the current work include:

- An analysis of the most known WE algorithms and implementations concerning performance and resource usage;
- A wrapper library that contains the original Word2vec and a set of optimizations, namely, the pWord2vec, pWord2vec_MPI, and Wang2vec algorithms. This tool allows the NLP community to access the parallels and

2377-5750/20/$31.00 ©2020 IEEE
DOI 10.1109/PDP50117.2020.00038

distributed tools bringing the possibility to generate models based in bigger corpus;
- A preliminary evaluation of the wrapper library.

The rest of this paper is organized as follows. Section II summarizes the concepts used in this study. Section III lists related work. Section IV presents a performance evaluation of the WE algorithms considered in this work. Section V describes in detail how our proposed solution works and its functionalities. Finally, Section VI depicts our conclusions and future directions.

II. Background

A. WE Algorithms

In recent years, the extensive use of Language Models (LMs) led to significant results in the NLP field. A standard approach for generating LMs is the use of algorithms such as Word2vec, FastText, and Wang2vec [5].

These algorithms receive large volumes of text in a given language. From a training strategy, they construct a vocabulary and learn vector representations (in a given vector space $\mathbb{R}^n$ of dimension n) for these words, based on the context in which they are inserted.

1) Word2vec training process: Word2vec is the algorithm for embeddings generation proposed by Mikolov et al. [1]. This method attempts to predict the neighborhood of a target word within a context window, producing vectors representations for the found vocabulary.

Two architectures are used by this algorithm: the Continuous Bag-of-Words (CBOW) and the Skip-Gram. The former predicts a target word from a given context. The latter predicts the context words for a target word. Both architectures require significant processing time, which can get worse according to the data volume and parameters used. However, it is noteworthy that, due to the increase in the range of words considered for training, the Skip-gram model has greater complexity [6], therefore requires greater computational power.

The WE models show precise forms of word representation, and their use is common in NLP systems that use words as basic input units [7]. These models are capable of representing document vocabularies, capturing the word context, its syntactic and semantic meaning, and words relations in a given corpus. As pointed out by Hartmann et al. [7], different methods for embeddings generation have been developed since the publication of the most popular among them Word2vec [1]. Most of these WE learning processes require high computational power, as we detail in Section IV. Parameters such as vector dimension, window, architecture, and usage of larger datasets, directly influence resource consumption.

2) Wang2vec training process: The Wang2vec, as mentioned in previous sections, is also an algorithm for generating embeddings, and it is fundamentally based on Word2vec. Two modifications of the Word2vec algorithm give rise to Wang2vec. These modifications allow the model to capture greater detail of the syntactic features of a language. In the CBOW architecture, the input tokens are the concatenation of the one-hot vectors of the context words that appear. In the Structured Skip-gram architecture, the prediction parameters change to predict each context word, depending on the position concerning the target word [8].

3) FastText training process: The FastText is a WE algorithm that is also divided into two architectures: CBOW and Skip-gram. This type of embeddings is used with success in many NLP tasks such as Text Classification and Named Entity Recognition. One of the main differences between Word2vec and FastText is that FastText can estimate values for words that are not part of its pre-trained model. It happens because the training of the model uses n-grams instead of whole words. For example, given the token "matter" and $n = 3$, we will have the *3-grams: <ma, mat, att, tte, ter, er>* [9].

B. Usability of WE algorithms in the NLP context

The Python language offers libraries that focus on NLP. These are target on preprocessing, manipulation, and analysis. Examples include the Gensim [2], NLPNET [3] and spaCy [4] libraries.

Among the above mentioned examples, the Gensim library, firstly defined as a framework, aims to fulfill a gap among the NLP applications. This library contains some of the main word processing algorithms, allowing them to be used from a single system. Python was the programming language used for its development due to its easy learning curve, compact syntax, multiplatform nature, and easy deployment [10].

Currently, Gensim is a free Python library, designed for raw and unstructured text processing for semantic data extraction [2]. Even though Gensim is one of the most popular options from the NLP community, there are some limitations in its use, e.g., elevated memory consumption and resource utilization, which may be a barrier to large data input processing. Also, Gensim does not support the HPC optimizations proposed for WE models generation, which could lead to significant performance improvements.

III. Related Work

The first Word2vec algorithm implementation was proposed by Mikolov et al. [1]. The original version already performed the model training in parallel through the pthread library, since a sequential execution would be unfeasible for real applications. In this case, the parallelism level increment occurs in a scenario where the main memory is global for all processor cores. The input text is divided by the number of threads in order to execute the training process and update the final output file, thus ignoring the race conditions [11]. From this proposal, other implementations based on the Mikolov's et al. algorithm emerged, some examples are the FastText and Wang2vec.

The FastText algorithm uses a method where each representation is induced by the sum of the N-grams vectors with the surrounding word vectors. The N-gram is a sequence of N words used to generate estimates in the probabilities of words attribution [12]. With its method, the FastText aims to capture morphological information to induce the process of generating WE [7], [13].

TABLE I
HPC AND WE ALGORITHMS FROM STATE-OF-THE-ART

	Features		Parallelism exploration		Usability	Performance
Implementations	**Cbow**	**Skip-Gram**	**Multicore**	**Multinode**	**Gensim compatible**	**Language**
Gensim.Word2vec	✓	✓	✓	-	✓	Python
Gensim.FastText	✓	✓	✓	-	✓	Python
Word2vec	✓	✓	✓	-	-	C
pWord2vec	-	✓	✓	-	-	C
pWord2vec_MPI	-	✓	✓	✓	-	C
Wang2vec	✓	✓	✓	-	-	C

Recent work shows concern over the Word2vec algorithm scalability and performance improvement, such as the proposal of the called pWord2vec [14], an optimization for shared and distributed memory contexts. Its implementation consists of a Word2vec extension with a negative sample sharing optimization. In highlighted points, there is a mini-batching based scheme (a division of the training data in smaller loads) and shared negative samples to convert basic linear algebra subprograms (BLAS) operations of level 1 vectors to multiplication operations of level 3 matrices.

Another similar study was proposed by Rengasamy et al. [11]. The authors aimed to increase throughput by sharing positive/negative samples in several context windows using Skip-gram architecture. Similarly, Ji et al. [14] proposed an optimization also based on the Word2vec algorithm, which addresses the distribution and parallelization in environments with distributed memory using MPI and OpenMP technologies. The gains, in this case, are significant, and show scalability for up to 32 nodes with 76 cores each.

Table I shows data regarding state-of-the-art algorithm implementations for the generation of WE Word2vec and FastText available in the Gensim library. It also indicates optimizations found in the bibliography, relating to functionality, exploration of parallelism, usability, and performance. Notice that implementations developed in Python language are present in the Gensim library and provide the user with the generation of WE models on the two architectures proposed by Mikolov et al. [1]. However, they are limited to the scalability requirements, since they do not use resources that provide scale-out. The optimizations written in the C language are impaired as to their usability since they are not present in the most popular tools for the application of these algorithms, as presented in Table I. However, optimizations written in C have performance gains and scalability.

Intending to present an interface for conceptually different types of word and document embeddings, facilitating the training and distribution of state-of-the-art sequence labeling, text classification and language models, Akbik A. et al. [15] proposed present Flair. This Framework aims to abstract specif engineering challenges that different types of WE raise by presenting a unified interface for all WE and arbitrary combinations of embeddings.

IV. PERFORMANCE EVALUATION

The most widespread WE algorithms in the NLP community uses programming with multiple processing threads on computers with shared memory. However, from an HPC perspective, the implementation proposed by Mikolov [1], as well as the other versions based on it, have low scalability and low performance. Consequently, it can take several hours to finish its executions.

This section investigates the performance and efficiency levels of the established WE algorithms. We also compare the high-performance optimizations to answer specific questions about performance and memory consumption.

A. *Algorithms performance evaluation*

The word embedding algorithms have become a study object for the HPC area [11], [14], as previously mentioned. Implementations focused on performance and scalability were developed, but most popular versions among the NLP researchers area do not surpass state-of-the-art results, such as implementations of [11], [14]. Figure 1 shows the minimum requirements in terms of memory consumption from Table I versions. Additionally, Table II presents the best results regarding possessing time.

For our experiments, we used the parameter vectors' size equal to 200, window 8, negative 25, sample 1e-4, iter 15, min-count 5 and a PT-BR 4 GB file [16] as input. The computational testbed consists of a Dell EMC PowerEdge R740 server with two sockets, each one containing a 5118 Xeon Gold processor of 2.30 GHz, 12 Cores/24 Threads 12 MB L2 cache. The total number of cores is 24/48, with 16.5

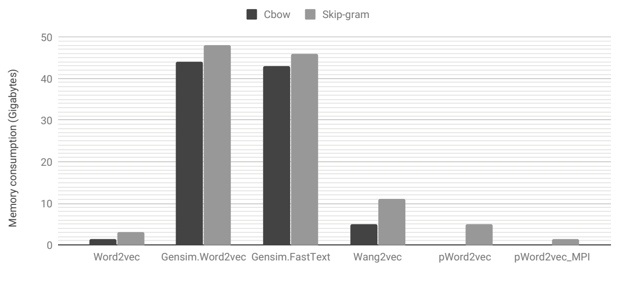

Fig. 1. Memory consumption of the analyzed algorithms.

MB of L3 cache shared by the two sockets and 322 GB of main memory.

TABLE II
FASTEST PROCESSING TIME OF THE ANALYZED IMPLEMENTATIONS.

Algorithm	Architecture	Processing time
Word2vec	Cbow	1.3 h
	Skip-Gram	22.1 h
Gensim.Word2vec	Cbow	2.0 h
	Skip-Gram	7.2 h
Gensim.FastText	Cbow	1.5 h
	Skip-Gram	3.0 h
Wang2vec	Cbow	1.3 h
	Skip-Gram	7.7 h
pWord2vec	Cbow	-
	Skip-Gram	1.4 h
pWord2vec_MPI	Cbow	-
	Skip-Gram	1.5 h

In Skip-gram architecture, Mikolov's Word2vec presents the highest execution time overall. Its fastest result is about 15 to 19 hours longer than Python implementations. However, its memory consumption, compared to others, is one of the lowest, being close to 1.5 GB. The Wang2vec algorithm presents a similar performance to Mikolov's version. Nevertheless, it performs the highest memory consumption compared to the other C versions, close to 10 GB. The Gensim's Word2vec, on the other hand, presents a faster execution time in comparison to the original version, finishing its execution about 15 hours earlier. However, it consumes much more resources, been close to 40 GB of RAM. The Gensim.FastText finishes its execution 4.2 hours faster and consumes approximately 3 GB more RAM than the Gensim.Word2vec. In general, both algorithms show similar results.

Regarding the analyzed versions which focus on CPU-parallelism, the pWord2vec is the algorithm that brings better results in terms of execution time and memory consumption, as presented in Figure 1 and Table II. This approach presents an execution time of 1.48 hours, consuming 5 GB of memory for the same workload used in all experiments. However, the pWord2vec contains only Skip-gram architecture.

Figure 2 presents the speed-up levels of the original Word2vec, pWord2vec, pWord2vec_MPI, Wang2vec, Gensim.Word2vec and Gensim.FastText over the number of threads/workers. The reference, in this case, is the original Word2vec's slowest execution time. From the speed-up levels shown in Figure 2, it is possible to identify how each algorithm deals with computational resources usage. The highlight goes to the pWord2vec, which presents a speed-up of 15x, being the fastest among the CPU-parallelism versions. For the pWord2vec's MPI version we generated projections for 4 and 8 nodes, based on the performance experienced on our testbed and the values demonstrated in S. Ji's paper [14].

The Gensim's Word2vec CBOW architecture maintains acceptable speed-up and efficiency levels until it reaches 12 cores. However, from this point on, the speed-up level stagnates and does not increase significantly when the number of processing cores is incremented from 12 to 24 and 48. The non-acceleration, even with additional resources, characterizes the low scalability of these versions.

Some implementations do not perform as expected in this scenario. Those who present a speed-up lower than 1 are slower than the reference, which means that they do not show any speed-up. For this experiment, we analyze the behavior of each version mentioned in the previous sections over the "workers" parameter (number of threads) variation.

B. Models' quality evaluation

We present an analysis of the models quality through an extrinsic evaluation of semantic similarity. The purpose of the semantic similarity task is to predict a degree of similarity (from 1 to 5) between two sentences. The corpus used in this evaluation is that of the shared task ASSIN *(Avaliação de Similaridade Semântica e Inferência Textual)*[1] proposed in PROPOR 2016. The evaluation of this task is made using two metrics: Pearson's Correlation (ρ) for semantic similarity and Mean Squared Error (MSE).

TABLE III
TRAINED MODELS' EXTRINSIC EVALUATION.

Algorithm	Architecture	ρ	MSE
Word2vec	Cbow	0.48	0.58
	Skip-Gram	0.54	0.54
Gensim.Word2vec	Cbow	0.51	0.56
	Skip-Gram	0.54	0.54
Gensim.FastText	Cbow	0.51	0.56
	Skip-Gram	0.55	0.53
Wang2vec	Cbow	0.48	0.58
	Skip-Gram	0.53	0.55
pWord2vec	Cbow	-	-
	Skip-Gram	0.53	0.55
pWord2vec_MPI	Cbow	-	-
	Skip-Gram	0.53	0.54

Table III contains the results obtained with this extrinsic evaluation. Although we perceive some variation, the results are close to each other with regard to the evaluated metrics. These evaluations may be used as a supplement when choosing which algorithm should be used in a specific scenario.

Another point to be analyzed is the variance and standard deviation of the metrics presented in table III. We present these values in table IV. With these values we can see that the generated models are quite similar, as we do not have loss in the similarity evaluation.

TABLE IV
MODELS EVALUATION

Model	s^2	s
ρ	0,000576	0,024008
MSE	0,000262	0,016181

[1]http://propor2016.di.fc.ul.pt/?page_id=381

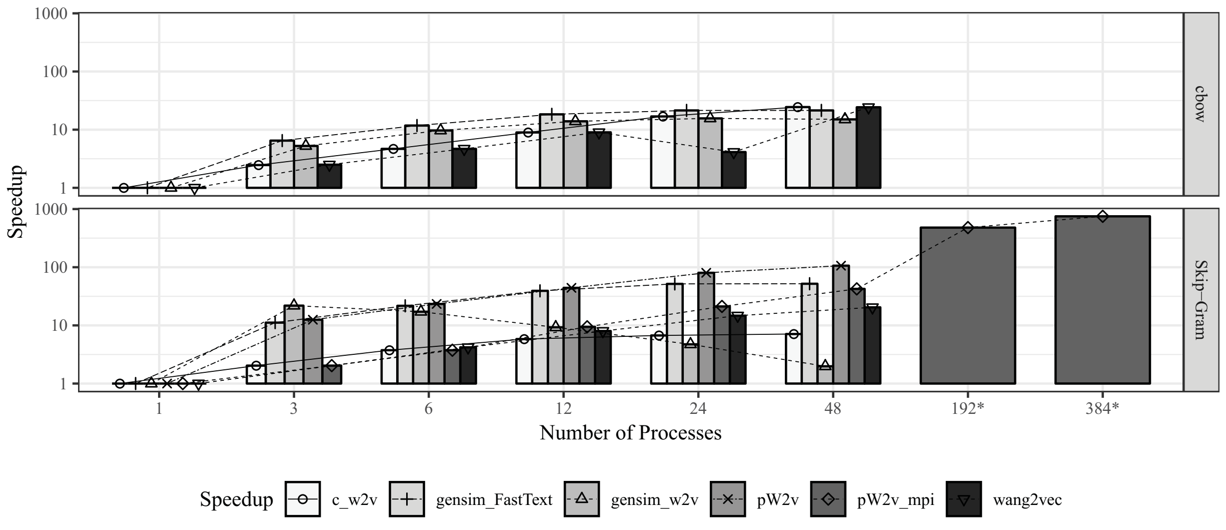

Fig. 2. Speed-up of word-embbedings algorithms generators. *Estimated values based on experiments.

V. Improving Usability

Analyzing the established optimizations proposed by the High-Performance community, we have compared performance, efficiency and memory consumption aspects. We notice that those developed in the C language present more considerable speed-up and lower memory consumption compared to the Python versions. Nevertheless, these optimizations are not commonly adopted by the NLP community. We hypothesize that this situation may be caused due to the lack of integration between these versions and the most popular NLP's tools, such as Gensim.

As shown in previous sections, each algorithm has its features and behavior to solve similar problems as well as, each parameter can increase the processing time. To minimize its scalability and performance problems, utilizing the best parameter set, demonstrated to be a relevant subject.

A. Preliminary wrapper library for optimized Word Embeddings implementations

To tackle this performance and usability issues, we have developed a wrapper [2], presented in Figure 3. The main goal is to integrate the high-performance optimizations earlier mentioned into a single library, in the most transparent way as possible regarding resource allocation and programming languages related issues. The implementations can be used for different purposes and applied to different computer architectures e.g. multicore or multinode. Utilizing this wrapper for the use of the mentioned WE algorithms, seeking the best usage of the available resources and features, can improve its usability.

We encapsulated bash scripts into python commands to run each algorithm considering specific requirements in local and remote environments. For remote executions, we used an integration with the TORQUE resource manager to manipulate job requests for the on-premise clusters. Therefore, for remote executions, the library will transfer the data input files to the on-premise cluster, make a batch job request to the TORQUE manager, and lastly bring the output back. The computational environments to be considered as a valid parameters must be previously configured, informing all access credentials.

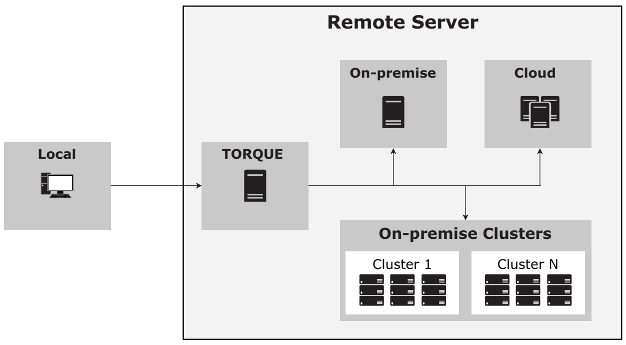

Fig. 3. Wrapper library for the optimized Word2vec implementations Call scheme

The analysis from previous sections, combined with the developed library, brings a better understanding of the analyzed WE implementations and allows users to choose the one that better suits their objectives. The algorithms composing the library are the original Word2vec, pWord2vec, pWord2vec_MPI, and Wang2vec.

The goal here is to allow researchers who do not have knowledge in high-performance ecosystems to use its resources more easily. It is worth noting that by applying the wrapper, applications' performance does not get worse,

[2]https://github.com/mmatheuslyra/Wrapper

which enforces its usability. Figure 4 summarizes the speed-up in comparison to the fastest time of the original Word2vec version, which used 48 threads. Since the testbed has 24 physical cores, it is also possible to identify how each version deals with additional resources.

Figure 4 complements the analysis, showing the best results in terms of performance and the maximum number of threads that provides speed-up. It is possible to identify that the pWord2vec reaches 15x speed-up over the fastest executions of the original Word2vec, and an estimated 105x in an 8-node cluster for its MPI version. The Gensim.Fastext, Gensim.Word2vec and Wang2vec algorithms achieved speed-up for the Skip-Gram architecture, being 7.3x, 3.07x and 2.8x respectively. For the Cbow architecture despite very close to the original the algorithms do not achieve speed-up.

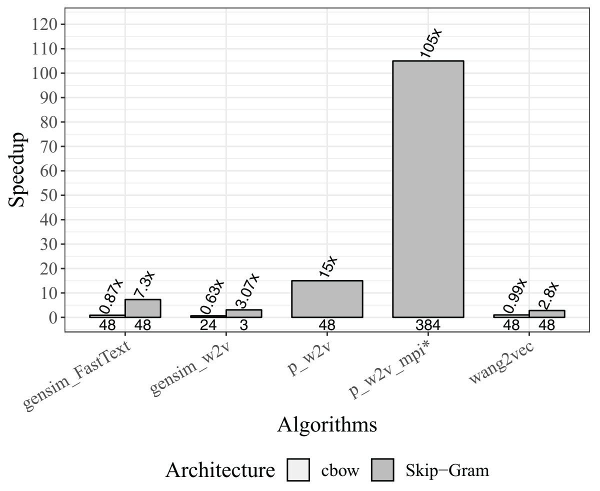

Fig. 4. Speedup comparison between wrapper library and original Word2vec executions. *Estimated values based on experiments.

VI. Conclusion

Aiming to promote a better understanding of each of the available implementations of the Word Embeddings algorithms and aid researchers to make more consistent choices regarding their applicability, we present in this paper a detailed performance evaluation that evidences their specific characteristics. We analyze performance, efficiency, memory consumption, scalability, and usability, in multicore and multinode architectures. Thus, it can become a strong ally to users and researchers, serving as an aid for choosing the most appropriate version for the desired output model, as well as the available computational resources.

We also developed a wrapper library that integrates the more optimized parallel and distributed versions of the WE algorithm with the most popular NLP tools improving their usability. This resulted in an average performance gain of 15x for multicores and 105x for multinodes compared to the original version. There is also a big reduction in the memory footprint allowing the execution of much bigger models.

In future work, we will investigate ways to automatically suggest the best hyperparameter setup to further optimize the execution of the parallel and distributed implementations presented in this work.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior – Brasil (CAPES) – Finance Code 001, Forense Science National Institute of Science and Technology (INCT) financed by CNPq Brazil.

This work has been supported by the project "GREEN-CLOUD: Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9), from FAPERGS and CNPq Brazil.

References

[1] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[2] Gensim, "Gensim: Topic modelling for humans," Apr 2019. [Online]. Available: https://radimrehurek.com/gensim/

[3] Nilc, "nlpnet — natural language processing with neural networks," Apr 2019. [Online]. Available: http://nilc.icmc.usp.br/nlpnet/

[4] spacy, "Industrial-strength natural language processing networks," Apr 2019. [Online]. Available: https://spacy.io

[5] O. Levy and Y. Goldberg, "Dependency-based word embeddings," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, 2014, pp. 302–308.

[6] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[7] N. Hartmann, E. Fonseca, C. Shulby, M. Treviso, J. Rodrigues, and S. Aluisio, "Portuguese word embeddings: Evaluating on word analogies and natural language tasks," *arXiv preprint arXiv:1708.06025*, 2017.

[8] W. Ling, C. Dyer, A. W. Black, and I. Trancoso, "Two/too simple adaptations of word2vec for syntax problems," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 1299–1304.

[9] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[10] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[11] V. Rengasamy, T.-Y. Fu, W.-C. Lee, and K. Madduri, "Optimizing word2vec performance on multicore systems," in *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*. ACM, 2017, p. 3.

[12] D. Jurafsky and J. H. Martin, *Speech and language processing*. Pearson London, 2014, vol. 3, pp. 1–3.

[13] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[14] S. Ji, N. Satish, S. Li, and P. Dubey, "Parallelizing word2vec in shared and distributed memory," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[15] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf, "Flair: An easy-to-use framework for state-of-the-art nlp," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 2019, pp. 54–59.

[16] *Repositório de Word Embeddings do NILC*, (November 29, 2018), [ONLINE]. Available: http://nilc.icmc.usp.br/embeddings.