A survey on embedding approaches for natural language processing

Yoan Dimitrov
Human-Centered Computing
Reutlingen University of Applied Sciences
Alteburgstr. 150, 72762 Reutlingen
yoan.dimitrov@student.reutlingen-university.de

ABSTRACT

Natural Language Processing, an increasingly important sub-field of artificial intelligence, deals with the interaction between computers and natural languages, such as English or German. While humans communicate with words and sentences, computers can only understand numbers. Words and sentences have to be, therefore, mapped onto fixed-length vectors of real numbers before being used as input to machine learning algorithms and neural networks. This process is usually referred to as word or sentence embedding[1]. Over the years many embedding approaches have emerged to tackle numerous natural language processing tasks such as named-entity recognition, sentence classification, or sentiment analysis. The following paper aims to present common embedding approaches and discuss their properties.

KEYWORDS

Natural Language Processings, Word Embeddings, Bag-of-Words, Tf-idf, Word2Vec, Glove, Fasttext, Doc2Vec

1 Introduction

Natural Language Processing (NLP) is an interdisciplinary field, including concepts and techniques from computer science, linguistics, cognitive psychology, and artificial intelligence. Since the 1950s, NLP has been subject to a lot of research [17, 25]. Today NLP systems take on the daunting task of understanding and generating languages in the form of speech or text. Well-engineered examples of NLP systems include dialog systems such as *Siri*[2] or *Google Home*[3] as well as the machine translation system *Google Translate*[4].

An essential key to the success of such NLP systems is a sturdy, preprocessing pipeline, which involves word and sentence embedding. Word embedding methods often help bridge the gap between computers and speech or text. In its simplest form, a word embedding approach converts words to fixed-length vectors of real numbers [43] ignoring their semantics and order [19, 24]. More sophisticated approaches are able to capture the semantic similarity of words and map them accordingly in a vector space [4, 18, 34, 27]. Other praiseworthy approaches attempt to represent whole sentences as vectors, while still considering their semantics and context [24].

This paper seeks to explain the aforementioned embedding approaches and discuss their properties. Since it is impossible to address the full range of embedding techniques in this paper, only a glimpse of the various techniques developed over the last few decades will be presented. The following word embedding methods will be discussed in the given order: Bag-of-words, Tf-idf, Word2Vec, Glove, and Fasttext. Then, Doc2Vec, an algorithm for generating sentence vectors, will be introduced. Other noteworthy approaches, which will not be further analyzed, are *BERT* by Devlin et al. [11] and the *Universal Sentence Encoder* by Cer et al. [8]. Both approaches achieved state-of-the-art performance and are currently growing in popularity.

2 Word embeddings

Word embedding approaches are essential stepping stones towards understanding and generating natural languages. There is no shortage of algorithms that can help computers understand languages by converting text to a numerical

1 Text vectorization is another common term used.

2 https://www.apple.com/de/siri/ (accessed on 10.10.19)

3 https://store.google.com/de/product/google_home (accessed on 10.10.19)

4 https://translate.google.com (accessed on 10.10.19)

representation [4, 5, 19, 24, 27, 28, 34, 35,]. The following section examines a number of common and more advanced approaches that have been established in recent years.

2.1 Bag-of-words and TF-IDF

The inituition behind the bag-of-words approach is rather trivial but still impressive:

"Text documents are represented as an unordered set of words with their position ignored, keeping only their frequency in the document." [19]

For example, consider the following sentences:

She has a best friend.
Her dog is white and black.
She has a white and black shirt.

The vocabulary of these three sentences consists of 12 unique words, although some words appear more than once. Using the bag-of-words approach, a fixed-length vector of size 12 can be used, with one position in the vector allocated to each word with a scoring [19]. The simplest scoring method is marking the presence and absence of words as a Boolean value:

She has a best friend → [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
Her dog is white and black → [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
She has a white and black shirt → [1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1]

Other scoring methods include: calculating the number of times a word appears in a document or calculating the frequency of each word in a document in proportion to all words in the document [19, 35]. The frequency word score, however, sometimes poses a problem because highly frequent words such as "the, a, and, or" receive the highest scores, although these words do not contain any relevant information. Term Frequency – Inverse Document Frequency (TF-IDF) seeks to tackle this very issue. It asks the question, whether all words in a document are equally important. Frequent words like "the, a, and, or", which are found across all documents, are penalized; simultaneously rare words across all documents receive high scores [19, 32, 35]. The formula is defined as:

$$Frequency\_Score = w_{t,d} = tf(t,d) * \log(\frac{N}{df(t)})$$

- tf (t, d) refers to the scoring of the frequency of the word (t) in the current document (d).
- N refers to all documents and df (t) is the scoring of how rare the word is across all documents.
- A log function is usually used by large numbers of documents to squash the IDF-score [19, 35].

Although it is evident that the semantics of each sentence is discarded by the use of bags-of-words combined with TF-IDF, these approaches already provide a simple way of extracting features from text data, for use in machine learning algorithms. Furthermore, they often prove to be adequate for some NLP tasks such as text classification and summarization [19, 32]. A major disadvantage of this combination is that as the vocabulary length increases, so does the vector representation. This, in turn, means that more computational resources are required, ultimately leading to slower performance [19, 24, 35].

2.2 Word2Vec

Word2Vec, a very common model for learning word embeddings, was introduced by Mikolov et al. in 2013 [27, 28]. Since then, it has been widely used in a variety of applications such as sentence classification [21], sentiment classification [3] and modeling music context [14].

Word2vec typically accepts a large text corpus[5] as input and outputs a collection of word vectors. Unlike BOW vectors that grow proportionately to the size of the given vocabulary [19], Word2vec creates far less complex word vectors ranging between 200 and 400 dimensions. Mikolov et al. proposed two closely related neural network architectures for learning word vector representations: Continous Bag-Of-Words (CBOW) and Skip-gram [27, 28]. Both models are depicted in Figure 1 and are regarded as shallow neural networks, since they only consist of one hidden layer.

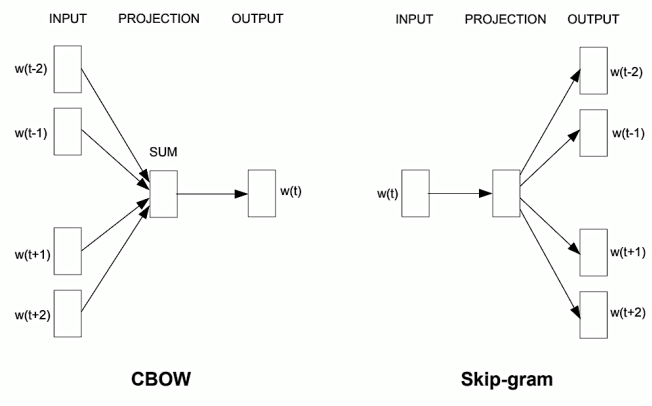

Figure 1: CBOW and Skip-gram architectures by Mikolov et al. [28]

[5] Text data usually goes through a series of preprocessing tasks such as tokenization or stemming before being inserted into a model.

CBOW and Skip-gram try to learn word embeddings by examining their neighboring word(s) and are, therefore, capable of capturing essential semantic similarities. Their parameters, in this case their weights and biases, are adjusted usuaully via stochastic gradient descent [19, 27, 28].

CBOW, as seen in figure 1, predicts a word by taking some surrounding context words into account. The input to the model could consist of preceding or/and following words of the target word (t) and is defined as [28]:

$$\boldsymbol{w(t-1), w(t-1), w(t+1), w(t+2)} \ldots$$

For example, consider the following sentence: *"The bird flies into its nest"*. If the CBOW model would attempt to find the word embeddings for the word "flies" and only consider a context window of size 2, it would output a probability score for the word "flies" given the context words "bird" and "into" or more formally: $\boldsymbol{P(target\ word | context\ words)}$. Mikolov et al. generally suggest choosing a context window between 2 and 10 [22].

Contrary to CBOW, Skip-gram predicts some corresponding context words e.g. $\boldsymbol{w(t-1), w(t-1), w(t+1), w(t+2)}$ … for a given word (t) [27, 28]. More precisely, using a softmax function, it produces a probability distribution of words that lie in close proximity to the target word. Mikolov et al. argue that the Skip-gram model performs well on small data sets and is more prone to rare words. In comparison with the Skip-gram model, the CBOW model trains quicker and is more sensitive to frequent words [19, 28].

2.3 GloVe

An equally popular word embedding algorithm named Global Vectors or GloVe was introduced by Pennington et al. in 2014 [34]. The GloVe algorithm attempts, similarly to Word2vec, to incorporate the context surrounding a word. The first step to creating word embeddings with this approach is the construction of a co-occurrence matrix based on some text corpus. A co-occurrence matrix X is simply defined as: $\boldsymbol{X_{ij}}$where each element of X represents the number of times a word *i* appears in the context of *j* [34, 19]. In order to create context-specific word vectors based on the co-occurrence matrix, the authors of GloVe defined a function which aims to calculate the difference between the dot product of the word and context vectors and the log of the co-occurrence probabilities. The equation is defined as [34]:

$$\boldsymbol{J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \widetilde{w_j} + b_i + \widetilde{b_j} - logX_{ij})^2}$$

The function $\boldsymbol{J}$ is essentially similar to Singular Value Decomposition (SVD) as it seeks to reduce the dimensionality of the matrix X, while preserving important information [34, 39, 31]. It iterates through a vocabulary $\boldsymbol{V}$; $\boldsymbol{w_i}$ and $\widetilde{\boldsymbol{w_j}}$ represent the word and context vectors and $\boldsymbol{b_i}$ and $\widetilde{\boldsymbol{b_j}}$ represent their corresponding bias terms. $\boldsymbol{X_{ij}}$ refers to the co-occurrence probabilities of words *i* and *j*. The weighting function $\boldsymbol{f(X_{ij})}$ makes sure that very common words such as "the, a, and, or" do not negatively impact the optimization. The word and context vectors $\boldsymbol{w_i}$ and $\widetilde{\boldsymbol{w_j}}$ are obtained by minimizing the function $\boldsymbol{J}$ through stochastic gradient descent [34, 37, 38]. The final words vectors are equivalent to the sum of $\boldsymbol{w_i}$ and $\widetilde{\boldsymbol{w_j}}$ or their average [34].

Although Pennington et al. argue that their model outperforms Word2Vec on similarity tasks and named entity recognition [34], in practice, it naturally depends on the NLP task and the dataset used. Still, it should be noted that GloVe does train faster than Word2Vec; its larger memory footprint due to the co-occurrence matrix is, however, a drawback [34, 38, 39]. A thorough analysis of the of the performance of the algorithms, discussed in this paper, can be found in a very recent study conducted by Wang et al. [39].

In recent years, GloVe has also been applied to diverse NLP research areas such as sentence classification [44], natural language inference [6], and image captioning [41].

2.4 Fasttext

Fasttext was developed by Bojanowski et al. in 2016 and is usually seen as an extension to the Word2Vec algorithm [5, 18]. Instead of just learning vectors on whole words of a text corpus like Word2Vec, Fasttext attempts to also learn the n-grams found within a word [5, 18, 19]. N-grams are, in this case, a sequence of N characters, where N represents the number of characters in a word [5, 18, 19, 30]. For example, the character 4-grams of the word "paper" are "pape" and "aper". Fasttext first splits words into N-grams and then feeds the N-grams to a CBOW or Skip-gram model in order to learn the embeddings. The final word vector is obtained by calculating the sum of all its N-gram embeddings [12, 18, 19, 30]. According to the authors of Fasttext:

"The full word is always included as part of the character ngrams, so that the model still learns one vector for each word." [12, 18]

A major improvement to Word2Vec is the Fasttext's sensitivity to Out-Of-Vocabulary (OOV) and rare words. Traditionally, OOV words would be represented by zero vectors [12] or the average of all word vectors [18, 5, 12]. Fasttext addresses this issue by simply learning the n-gram embeddings of a word. Rare words are sometimes regarded as insignificant to algorithms and are often discarded depending on the NLP task; however, Grave et al. have shown that rare words are

semantically important in many languages [12]. Fasttext is capable of generating embeddings for such words, as some of their n-grams are very likely to be found in other more common words [18, 5, 30].

Details on the actual performance of Fasttext can be found in comparative study by Wang et al. [39] or by Mikolov et al. [29]. Applications of Fasttext include hate speech detection [2], commonsense inference [42], and sentiment analysis [1].

2.5 Word vector similarity measures

Word2Vec, GloVe, and Fasttext, though implemented differently, all output low dimensional, context-aware word vectors, which provide valuable information about the given text corpus and form the basis for text similarity analysis. The similarity of these word vectors can be computed by using different measurement approaches such as: Consine Similarity, Euclidean Distance, Jaccard Similarity, Matching Coefficient [13], or Word Mover's Distance [23]. Such similarity measures are especially important when trying to identify ambiguity in words or phrases [17, 25].

It should be noted, that dimensionality reduction algorithms such as Principal Component Analysis (PCA) [36] and T-distributed Stochastic Neighbor Embedding (t-SNE) [15] are typically applied to word vectors as it can be computationally expensive to measure their semantic similarities.

3 Sentence embeddings

Many NLP tasks require sentence vectors instead of word vectors to conduct sentence classification, document summarization, or sentimental analysis. An algorithm to directly create sentence vectors from a text corpus has been developed by Le et al. [24] and will be explained in detail in the next section. However, some researchers tend to favor other approaches with the help of word vector algorithms such as Word2vec, GloVe, or Fasttext. For example, Liu generated word embeddings with Word2vec and simply averaged the vectors[6] of the words in a sentence to obtain sentence vectors, which he later used for citation sentiment classification [26]. Others have adopted a similar approach using GloVe [40] or Fasttext word vectors [10]. Furthermore, Chen's research proved that multiplying each word vectors with their TF-IDF-scores and then combining them to construct a sentence vector is more efficient than simply averaging the word vectors [9]. That being said, both approaches seem to work and produce significant results [9, 10, 20, 26, 40].

3.1 Doc2Vec

Doc2Vec, yet another extension of Word2Vec, was developed by Le and Mikolov in 2014. This model seeks to represent sentences and words as fixed length vectors while still capturing their semantics. The authors introduced two improvised versions of the word2vec models CBOW and Skip-gram [24]. Paragraph Vector-Distributed Memory (PV-DM), the extension of the CBOW model, is depicted in Figure 2.

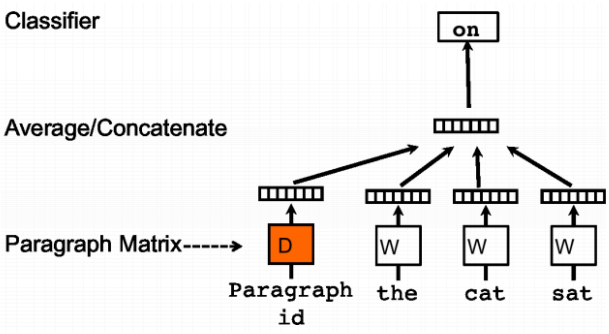

Figure 2: PV-DM by Le and Mikolov [24]

PV-DM is closely linked to the CBOW model but has an additional input called the paragraph id (or Paragraph Matrix). Le and Mikolov explain that the prediction of a target word given a set of context words strongly relies on the topic of the document. This additional information is represented by the paragraph matrix and helps the model remember what is missing from the current context. The paragraph matrix ***D*** and the context words ***W*** are either averaged or concatenated to predict a target word (***t***). The model is trained using stochastic gradient descent and not only maps words to their vector representations, but also paragraphs. After learning the appropriate representations for the paragraph matrix and context words based on a text corpus, the PV-DM model is able to generate new sentence / paragraph vectors. Le and Mikolov note that one of the main advantages of their design is the preservation of the word order in comparison to the simple word to phrase vector conversions mentioned previously. This turns out to be significant in some NLP tasks [24].

Le and Mikolov showed that their model performs well on certain NLP-Tasks such as information retrieval, text classification, and sentiment analysis [24]. In addition, their model has also played a vital role in abusive language detection [33] and multi-document summarization [7].

[6] The word vectors are usually normalized before a sentence vector is created.

Other popular sentence embedding implementations which take on a learning approach include: Skip-Thought by Kiros et al. [22] and FastSent by Hill et al. [16].

4 Conclusion

A few popular word and sentence embedding approaches are discussed in this survey. Their underlying structure is presented and a number of applications are mentioned. The study shows that Bag-of-words combined with TF-IDF scoring provides the simplest approach to converting text to a numerical representation. However, this approach does have its drawbacks, since it ignores word order and semantics. Word2Vec, Glove, Fasttext, and Doc2Vec, although to some degree built differently, all capture semantic similarities of words. This study has also found that Word2Vec acts as the basis for other embedding methods, such as Fasttext and Doc2Vec. As mentioned earlier in this paper, it is quite difficult to choose the right embedding approach since it requires a lot of information about the problem at hand. It depends, in general, on the NLP task and the desired output.

References

[1] Altowayan A. A., Lixin, T. 2016. Word embeddings for arabic sentiment analysis. *In IEEE International Conference on Big Data*.

[2] Badjatiya, P., et al. 2017. Deep learning for hate speech detection in tweets. *In Proceedings of the 26th International Conference on World Wide Web Companion*, pages 759–760.

[3] Bansal, B., Srivastava, S. 2018. Sentiment classification of online consumer reviews using word vector representations. *Procedia Computer Science,* 132, pp. 1147-1153.

[4] Bengio, Y., et al. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*.

[5] Bojanowski, P., et al. 2017. Enriching word vectors with subword information. *TACL* 5:135–146.

[6] Bowman, R. S., et al. 2015. A large annotated corpus for learning natural language inference. *In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics*.

[7] Carrillo-Mendoza, P., Calvo, H., Gelbukh, A. 2017. Intradocument and Inter-document Redundancy in Multi-document Summarization. *In: Advances in Computational Intelligence. Ed. by Grigori Sidorov and Oscar Herrera-Alcántara. Cham: Springer International Publishing*, pp. 105– 115.

[8] Cer, D., et al. 2018. Universal sentence encoder. *CoRR*, abs/1803.11175.

[9] Chen, M. 2017. Efficient vector representation for documents through corruption. *In ICLR*.

[10] Conneau, A., et al. 2018. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. *In ACL*.

[11] Devlin, J., et al. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR* abs/1810.04805.

[12] Grave, E., et al. 2018. Learning word vectors for 157 languages. *In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

[13] Gomaa, W.H., Fahmy, A.A. 2013. A survey of text similarity approaches. *Int. J. Comput. Appl.* 86 (13), 13–18.

[14] Herremans, D., Chuan, C. H. 2017. Modeling musical context with word2vec. *Proceedings of the 1st International Workshop on Deep Learning and Music 1* (05/2017 2017), Anchorage, US, 11--18.

[15] Heuer, H. 2015.Text comparison using word vector representations and dimensionality reduction. *Proc. EuroSciPy*, pp. 13-16.

[16] Hill, F., Kyunghyun, C., Korhonen, A. 2016. Learning distributed representations of sentences from unlabelled data. *In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1367–1377, San Diego, California.

[17] Joshi, A.K. 1991. Natural language processing. *Science*, 253 (5025), pp. 1242-1249.

[18] Joulin, A., et al. 2017. Bag of tricks for efficient text classification. *In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

[19] Jurafsky, D. and Martin, H. J. 2018. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. *Pearson. Prentice Hall, Third Edition draft*.

[20] Kenter, T, Borisov, A., de Rijke, M. 2016. Siamese CBOW: Optimizing Word Embeddings for Sentence Representations. *In ACL - Proceedings of the 54th Annual Meeting of the Association for. Computational Linguistics*. Berlin, Germany, pages 941–951.

[21] Kim, Y. 2014. Convolutional neural networks for sentence classification. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),* pages 1746-1751, Doha, Qatar. Association for Computational Linguistics.

[22] Kiros, R., et al. 2015. Skip-thought vectors. *In NIPS*.

[23] Kusner, M. J., et al. 2015. Weinberger. From word embeddings to document distances. *In ICML*.

[24] Le, V. Q, Mikolov, T. 2014. Distributed Represenations of Sentences and Documents. *In Proceedings of ICML 2014*.

[25] Liddy, E. 2001. Natural language processing. *2nd edn. Encyclopedia of Library and Information Science. Marcel Decker*.

[26] Liu, H. 2017. Sentiment analysis of citations using word2vec. *arXiv preprint arXiv:*1704.00177.

[27] Mikolov, T., et al. 2013a. Distributed representations of words and phrases and their compositionality. *In Advances in Neural Information Processing Systems*.

[28] Mikolov, T., et al. 2013. Efficient estimation of word representations in vector space. *ICLR Workshop*.

[29] Mikolov, T., et al. 2018. Advances in pre-training distributed word representations. *In Proceedings of LREC*. Miyazaki, Japan.

[30] Nadkarni, P.M., et al. 2011. Natural language processing: an introduction. *J Am Med Inform Assoc*., vol. 18 5(pg. 544-551).

[31] Naili, M., et al. 2017. Comparative study of word embedding methods in topic segmentation. *Procedia Computer Science*, 112:340–349.

[32] Nenkova, A. and McKeown, K. 2012. A survey of text summarization techniques. *In Aggarwal, C. C. and Zhai, C., editors, Mining Text Data*.

[33] Nobata, C., et al. 2016. Abusive language detection in online user content. *In WWW*, 145–153.

[34] Pennington, J., et al. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing*.

[35] Ramos, J. 2003. Using tf-idf to determine word relevance in document queries. *In ICML*.

[36] Raunak, V. 2017. Effective dimensionality reduction for word embeddings. *NIPS 2017, workshop on Learning with Limited Labeled Data*.

[37] Salle, A., et al. 2016b. Matrix factorization using window sampling and negative sampling for improved word representations. I*n Proceedings of ACL*.

[38] Shi, T., Liu, Z. 2014. Linking glove with word2vec. *arXiv preprint arXiv*:1411.5595.
[39] Wang, B., et al. 2019. Evaluating word embedding models: Methods and experimental results. *CoRR*, vol. abs/1901.09785.
[40] Wieting, J., et al. 2016. Towards universal paraphrastic sentence embeddings. *International Conference on Learning Representations.*
[41] You, Q., et al. 2016. Image captioning with semantic attention. *In CVPR*.
[42] Zellers, R., et al. 2018. Swag: A large-scale adversarial dataset for grounded commonsense inference. *In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
[43] Zellig S. H. 1954. Distributional Structure, WORD, 10:2-3, 146-162.
[44] Zhang, Y., Wallace, B. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv*:1510.03820.