REVIEW: WORD EMBEDDING

A PREPRINT

Haowen Xu
School of Computational Science and Engineering
Georgia Institide of Technology
Atlanta, GA 30332
haowen.xu@gatech.edu

December 13, 2019

ABSTRACT

Word embedding is a technique that maps words and phrases in natural language to vectors or tensors. With the rapid development of deep learning methods, these vectors or tensors are usually continuous values so that them can be fed into neural networks and applied to gradient based methods. In this way, word embedding is a bridge between symbolic representations and distributed representations. It makes it possible to apply deep learning methods to Natural Language Processing (NLP) tasks. In this review,

1 Introduction

Word embedding is the fundamental component in solving NLP tasks with deep learning methods. It transforms discrete, symbolic words and phrases into distributed vector representations that can be fed into neural networks. A good pretrained word embedding can greatly improve the performance of a downstream task, especially when there is not enough data for the downstream task [1, 2].

In general, word embedding methods can be classified into two groups: 1) Count-based models, 2) Prediction-based models. In count-based models, words are expressed as the frequency of word-context co-occurrence counts globally in a corpus. This idea of word representation started as early as in the 1960s when Vector Space Model (VSM) was developed by Information Retrieval community [3]. In VSM, documents are represented as vectors where each dimension corresponds to a separate term (word or phrase). If a term occurs in the document, a weight will be assign to the corresponding dimension. One popular scheme to compute the weight is if-idf. However, the dimension of vectors in this representation is too large, so people further developed Latent Semantic Analysis (LSA), which uses Singular value decomposition to reduce the number of dimensions. More recently, Dhillon et al. (2011) proposed Low Rank Multi-View Learning method [4] which uses Canonical Correlation Analysis between the left and right contexts of a given word. In 2013, Lebret and Collobert [5] proposed another Count-based model by applying Hellinger PCA transformation to the word-context matrix. Another well-known count-based model is GloVe [6] by Pennington et al. (2014). briefly describe Glove

In prediction-based models, the word embedding is learnt by improving the predictive ability of a predefined target. The history of prediction-based models started from a series of works from Bengio et al. [7]. In this work, learning of word embedding is combined with the training of a language model. First they map each word in a sentence to a distributed feature vector. These vectors are called embedding layer. Then the embedding layer will be fed into a neural network, which predicts the next word in a sentence based on its previous n words. From then on, the study of predictive-based word embedding models has became more and more popular in the NLP community. Some of these famous works include word2vec [8, 9], fasttext [10, 11], ELMo [12], BERT [10]. In the rest of this review, we will give more details about these algorithms in Section 2 (Count-based) and Section 3 (Prediction-based). In Section 4 we conclude and provide some promising further research topics.

2 Count-based models

2.1 Vector Space Model

As the name indicated, count-based models represent word by counting its appearance in a context. Vector Space Model or VSM is the most basic version of Count-based models. VSM is mainly used in methods of document classification and document similarity estimation. It contains two parts. The first part is called **Bag of Words** or BoW. For example [13], we have two simple documents:

1)*John likes to watch movies. Mary likes movies too.*

2)*Mary also likes to watch football games.*

Based on these two documents, we can construct two bags of words:

$$BoW1 = \{"John" : 1, "likes" : 2, "to" : 1, "watch" : 1, "movies" : 2, "Mary" : 1, "too" : 1\}$$

$$BoW2 = \{"Mary" : 1, "also" : 1, "likes" : 1, "to" : 1, "watch" : 1, "football" : 1, "games" : 1\}$$

After transforming the documents into Bag-of-Words, we build a vector to represent each document. The dimension of this vector is equal to the size of vocabulary of the whole corpus. We calculate a weight from the Bag-of-words for each word and assign the weight to its corresponding dimension in the vector. Some of the popular schemes to compute the weight are: 1) a binary value (with 1 indicating that the word occurred in the document, and 0 indicating that it did not); 2) term frequency (the number of times a word appears in the document); 3) term frequency-inverse document frequency (a normalized version of term frequency which takes into account the frequency of a word occurred in corpus). For example if we use term frequency, the representations for the above two documents will be:

$$(1)[1, 2, 1, 1, 2, 1, 1, 0, 0, 0]$$

$$(2)[1, 1, 1, 1, 0, 0, 0, 1, 1, 1]$$

2.2 LSA

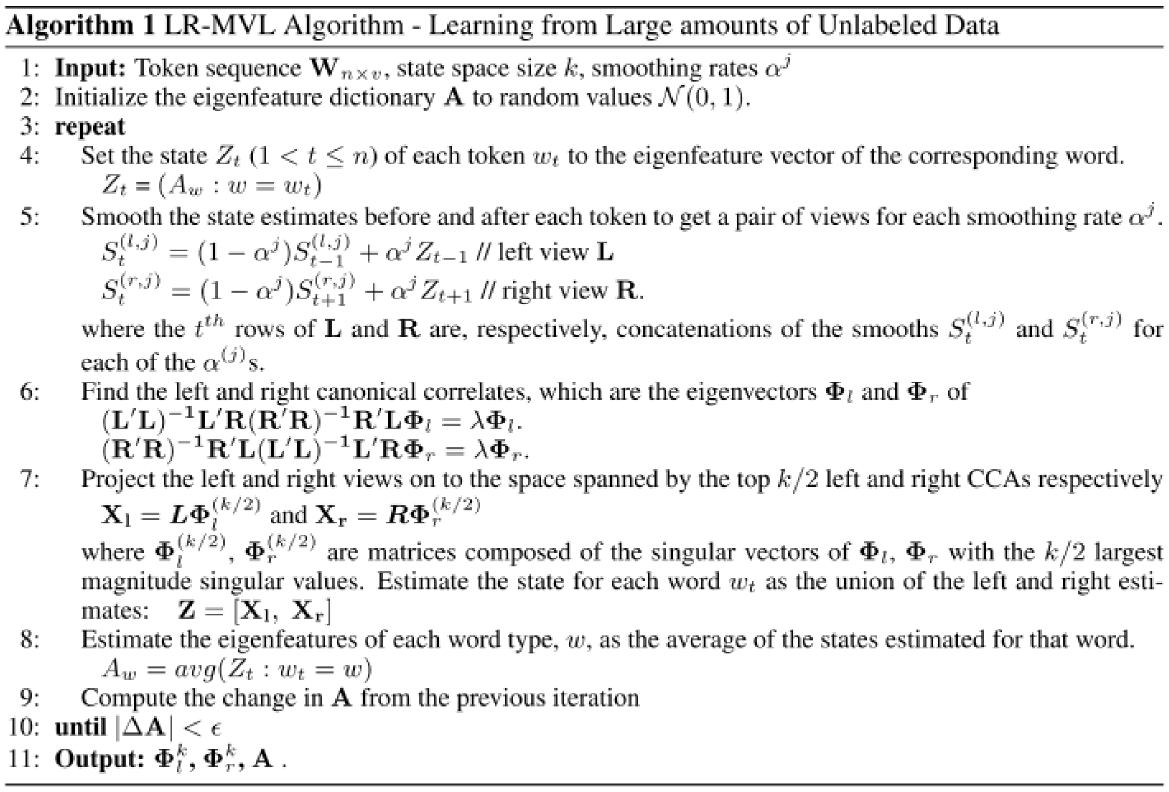
However, the dimension of the representation provided by VSM equals to the size of the vocabulary of the whole corpus, which could be very large in practice. To reduce the dimension, Deerwester et al. proposed Latent Semantic Analysis (LSA) [14] in 1990 which reduces the number of dimensions by doing Singular value decomposition on the word-context matrix.

2.3 Low Rank Multi-View Learning (LR-MVL)

In 2011, Dhillon et al. proposed LR-MVL [4] where embeddings are derived by leveraging Canonical Correlation Analysis (CCA) between the left and right contexts of a given word. To be more specific, LR-MVL has two stages. First, given a sequence of words $w = \{w_0, w_1, ..., w_n\}$, LR-MVL wants to learn a $v \times k$ matrix A that maps each of the v words in the vocabulary to a reduced rank k-dimensional state vector. Second, Induce context specific embeddings for each word from its k-dimensional state vector. The first stage of LR-MVL uses an iterative algorithm by iterating between the following four steps:

- Take the h words to the left and to the right of each target word w_t and project them each down to a k-dimension vector using A (called left contexts vector and right contexts vector).
- Take the CCA between the left contexts vector and right contexts vector to find the left and right canonical correlates.
- Project the left contexts vector and right contexts vector on to the space spanned by the top k/2 left and right CCAs respectively and concatenate them to get a new representation of w_t.
- Update A using the new representation we get in previous step.

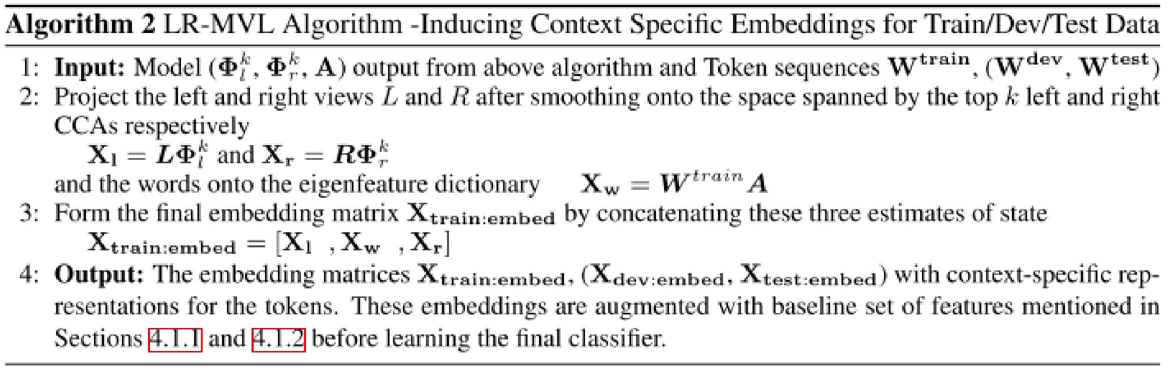
More details can be found in Algorithm 1.



Algorithm 1 LR-MVL Algorithm - Learning from Large amounts of Unlabeled Data

```
1: Input: Token sequence $\mathbf{W}_{n \times v}$, state space size $k$, smoothing rates $\alpha^j$
2: Initialize the eigenfeature dictionary $\mathbf{A}$ to random values $\mathcal{N}(0, 1)$.
3: repeat
4:     Set the state $Z_t$ $(1 < t \leq n)$ of each token $w_t$ to the eigenfeature vector of the corresponding word.
         $Z_t = (A_w : w = w_t)$
5:     Smooth the state estimates before and after each token to get a pair of views for each smoothing rate $\alpha^j$.
         $S_t^{(l,j)} = (1 - \alpha^j) S_{t-1}^{(l,j)} + \alpha^j Z_{t-1}$ // left view $\mathbf{L}$
         $S_t^{(r,j)} = (1 - \alpha^j) S_{t+1}^{(r,j)} + \alpha^j Z_{t+1}$ // right view $\mathbf{R}$.
       where the $t^{th}$ rows of $\mathbf{L}$ and $\mathbf{R}$ are, respectively, concatenations of the smooths $S_t^{(l,j)}$ and $S_t^{(r,j)}$ for
       each of the $\alpha^{(j)}$s.
6:     Find the left and right canonical correlates, which are the eigenvectors $\mathbf{\Phi}_l$ and $\mathbf{\Phi}_r$ of
         $(\mathbf{L}'\mathbf{L})^{-1}\mathbf{L}'\mathbf{R}(\mathbf{R}'\mathbf{R})^{-1}\mathbf{R}'\mathbf{L}\mathbf{\Phi}_l = \lambda \mathbf{\Phi}_l$.
         $(\mathbf{R}'\mathbf{R})^{-1}\mathbf{R}'\mathbf{L}(\mathbf{L}'\mathbf{L})^{-1}\mathbf{L}'\mathbf{R}\mathbf{\Phi}_r = \lambda \mathbf{\Phi}_r$.
7:     Project the left and right views on to the space spanned by the top $k/2$ left and right CCAs respectively
         $\mathbf{X_l} = \boldsymbol{L}\mathbf{\Phi}_l^{(k/2)}$ and $\mathbf{X_r} = \boldsymbol{R}\mathbf{\Phi}_r^{(k/2)}$
       where $\mathbf{\Phi}_l^{(k/2)}$, $\mathbf{\Phi}_r^{(k/2)}$ are matrices composed of the singular vectors of $\mathbf{\Phi}_l$, $\mathbf{\Phi}_r$ with the $k/2$ largest
       magnitude singular values. Estimate the state for each word $w_t$ as the union of the left and right esti-
       mates: $\mathbf{Z} = [\mathbf{X_l}, \mathbf{X_r}]$
8:     Estimate the eigenfeatures of each word type, $w$, as the average of the states estimated for that word.
         $A_w = avg(Z_t : w_t = w)$
9:     Compute the change in $\mathbf{A}$ from the previous iteration
10: until $|\Delta \mathbf{A}| < \epsilon$
11: Output: $\mathbf{\Phi}_l^k$, $\mathbf{\Phi}_r^k$, $\mathbf{A}$ .
```

The second stage is simply concatenating the state vector of a word and its left contexts vector and right contexts vector (see Algorithm 2). It is worth mentioning that the authors reported that the context component in embeddings not always increases the performance of downstream tasks. They found that in NER and Chunking problems, modeling the context gives decent improvements in accuracy.



Algorithm 2 LR-MVL Algorithm -Inducing Context Specific Embeddings for Train/Dev/Test Data

```
1: Input: Model ($\mathbf{\Phi}_l^k$, $\mathbf{\Phi}_r^k$, $\mathbf{A}$) output from above algorithm and Token sequences $\mathbf{W}^{\mathbf{train}}$, ($\mathbf{W}^{\mathbf{dev}}$, $\mathbf{W}^{\mathbf{test}}$)
2: Project the left and right views $L$ and $R$ after smoothing onto the space spanned by the top $k$ left and right
   CCAs respectively
     $\mathbf{X_l} = \boldsymbol{L}\mathbf{\Phi}_l^k$ and $\mathbf{X_r} = \boldsymbol{R}\mathbf{\Phi}_r^k$
   and the words onto the eigenfeature dictionary     $\mathbf{X_w} = \boldsymbol{W}^{train}\boldsymbol{A}$
3: Form the final embedding matrix $\mathbf{X_{train:embed}}$ by concatenating these three estimates of state
     $\mathbf{X_{train:embed}} = [\mathbf{X_l}\ \ , \mathbf{X_w}\ \ , \mathbf{X_r}]$
4: Output: The embedding matrices $\mathbf{X_{train:embed}}$, ($\mathbf{X_{dev:embed}}$, $\mathbf{X_{test:embed}}$) with context-specific rep-
   resentations for the tokens. These embeddings are augmented with baseline set of features mentioned in
   Sections 4.1.1 and 4.1.2 before learning the final classifier.
```

2.4 GloVe

The well-known GloVe is also a count-based model but it uses neural methods to decompose the co-occurrence matrix into more expressive and dense word vectors. To introduce GloVe, first we borrow the notation used in the original paper. Let the matrix of word-word co-occurrence counts be denoted by X, whose entries X_{ij} tabulate the number of times word j occurs in the context of word i. Let $X_i = \sum_k X_{ik}$ be the number of times any word appears in the context of word i. Finally, let $P_{ij} = P(j|i) = X_{ij}/Xi$ be the probability that word j appears in the context of word i.

With these notation, we can start the introduction of GloVe. The authors argue that the starting point for word vector learning should be with the ratios of co-occurrence probabilities, which is P_{ik}/P_{jk}, rather than the probabilities themselves (P_{ij}). Thus, they assume a general model with the form:

$$F(w_i, w_j, \hat{w_k}) = \frac{P_{ik}}{P_{jk}},$$

where w_i and w_j are word vectors and $\hat{w_k}$ are separate context word vectors.

Under the assumption that 1) vector spaces are inherently linear structures, 2) the final model should bew invariant under relabeling, the authors further simplify the model to the form:

$$w_i^T \hat{w_k} + b_i + \hat{b_k} = log(X_{ik})$$

The objective now becomes finding w to best fit this equation. Thus they propose a new weighted least squares regression model:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \hat{w_k} + b_i + \hat{b_k} - log(X_{ik}))^2,$$

where V is the size of the vocabulary. $f(X_{ij})$ is a weighting function. Learning the word embeddings now becomes solving the optimization problem.

Authors report better results than other count-based models, as well as prediction based models such as Skip-Gram [8] (one version of Word2Vec) in some tasks. However, in general we cannot state that one algorithm outperforms the other between GloVe and Word2Vec. Neither of them has been shown to provide definitively better results over all tasks.

3 Prediction-based models

3.1 Neural Network Language model (NNLM)

The pioneering work in Prediction-based models is strongly linked with the first neural network language model by Bengio et al. [7]. In this work, the word embedding is a by-product of training a neural network language model. A language model aims to predict the next word in a sentence given previous n words, which can be formed as:

$$f(w_{t-n+1}, ..., w_t) = P(w_t|w_{t-n+1}, ...w_{t-1}).$$

The authors decompose the function $f(w_{t-n+1}, ..., w_t) = P(w_t|w_{t-n+1}, ...w_{t-1})$ in two parts:

- A $|V| \times m$ matrix C which maps any word i in vocabulary V to a real vector $C(i) \in \mathbb{R}^m$. This part is also called embedding layer.
- A function g that maps an input sequence of feature vectors for words, $\{C(w_{t-n+1}), ..., C(w_{t-1})\}$, to a conditional probability distribution over words in V for the next word w_t. g is a neural network with parameters ω.

The architecture of this model is shown in Figure 1.

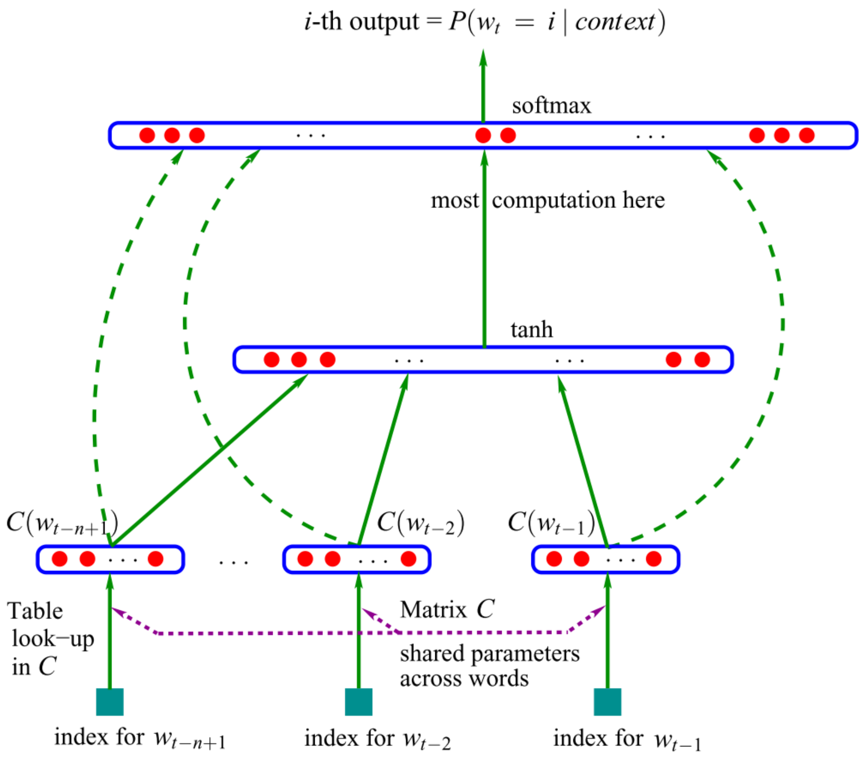


Figure 1: Neural architecture: $f(i, w_{t-n+1}, ..., w_{t-1}) = g(i, C(w_{t-n+1}), ..., C(w_{t-1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector. Figure borrowed from [7]

Training is achieved by optimizing the penalized log-likelihood function:

$$L = \frac{1}{T} \sum_t log f(w_{t-n+1}, ..., w_t; \theta) + R(\theta)$$

where $R(\theta)$ is a regularization term, $\theta = (C, \omega)$. This optimization problem can be solved by gradient based method.

3.2 Word2Vec

After Bengio's work [7], many contributions were made towards this line of research. However, among most of these works, word embedding is just a by-product of training a language model.

The first time a model was built primarily to learn the word embeddings was in 2008. Collobert and Weston [5] proposed a multi-task neural network which jointly optimizes several tasks over supervised and unsupervised data, but the target was to learn the embeddings. In 2013, Mikolov et al. [15, 8, 9] proposed Word2Vec, which finally drawn people's attention to word embeddings itself as a research topic.

Word2Vec provides two different architectures to produce word embeddings.

- **Continuous bag-of-words (CBOW)** Learning to predict the word by its context.
- **Continuous Skip-gram** Learning to predict the context given a word.

3.2.1 Continuous Bag-of-Words Model

CBOW is similar to the feedforward NNLM with three main difference, 1) the non-linear hidden layer is removed; 2) the projection layer is shared for all words; 3) future words are also used (since we no longer make the training of a language model as our primary goal). The model architecture is shown at Figure 2. Note that the weight matrix between the input and the projection layer is shared for all word positions in the same way as in the NNLM.

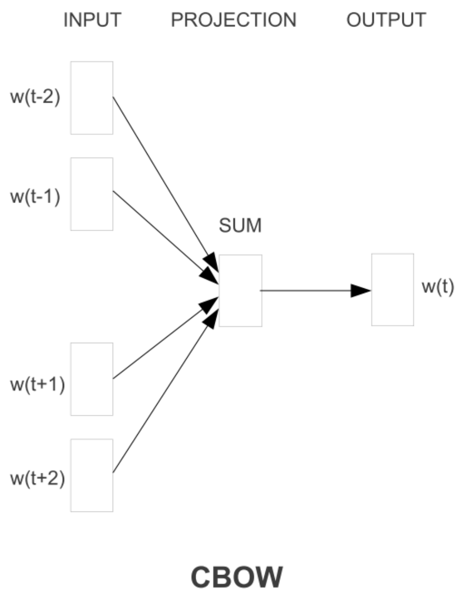


Figure 2: CBOW architecture. Figure borrowed from [8]

3.2.2 Continuous Skip-gram Model

The architecture of Skip-gram differs from CBOW in that, instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. That is, we use each current word as an input and predict its context (words within a certain range before and after the current word). The authors report that increasing the range improves quality of the resulting word vectors but increases the computational complexity at the same time. The model architecture is shown at Figure 3.

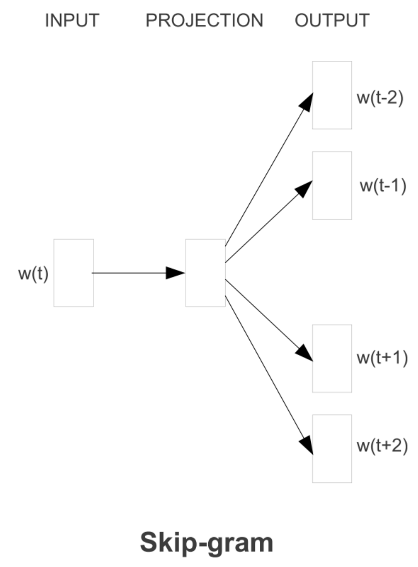


Figure 3: Skip-gram architecture. Figure borrowed from [8]

3.2.3 Variants of Word2Vec

The vocabulary in the corpus could be very large in practise, which makes the calculation of the normalization factor in normal softmax extremely time-consuming. To deal with this problem, in the first versions of Word2Vec[8], both CBOW and SG use hierarchical softmax in the last layer. This methodology could reduce the computational complexity from $O(V)$ to $O(log_2 V)$. Another methodology to alleviate this problem was introduced in a Mikolov et al. [9]. Instead of using hierarchical softmax, they use negative sampling to reduce computational complexity. The key idea in negative sampling is that, instead of update the weights for words (V in usual case), we only update a small group of negative samples (2-20 words suggested by the authors) randomly selected from the whole vocabulary. To achieve this, negative sampling reformulate the problem into a binary classification problem, which predicts whether a context-word pair is a real pair selected from the corpus, or is a "fake" pair we randomly sampled.

Another techniques introduced in [9] is subsampling of frequent words to reduce the amount of noise due to frequent words such as "the", "a", and accelerate the training at the same time.

A more recent contribution to the line of Word2Vec embedding is called FastText [10, 11, 16]. Based on Skip-gram, they took a finer granularity of language. Instead of training word embeddings, they train n-gram embeddings (sometimes called subword embeddings). The technique is helpful in 1) capturing meaning for suffixes/prefixes, 2) decomposing compound words, 3) understanding rare words.

3.3 Context dependent models

All models introduced above are context independent models, that is, each word has a fixed embedding no matter what context it is in. This can cause some problem when a word has different means in different context. For example, "He went to the prison **cell** with his **cell** phone to extract blood **cell** samples from inmates" [17]. The word **cell** has different meanings based on the context. However, context independent models simply collapse all meanings into one vector.

ELMo [12] and BERT [18] address this problem by generate different word embeddings for a word that captures the context of a word. ELMo and BERT give the state of the art results in many downstream tasks in Natural Language Processing.

4 Conclusion

Word embeddings encode words and phrases into fixed-length dense vectors, dramatically promote the research of natural language processing. In this review, we summarize some of the main works and approaches to derive word embeddings. There are two main approaches, prediction-based models, which model the probability of a word given a context, and count-based models, which leverage global co-occurrence statistics in word-context matrices. A more recent progress in word embedding is context dependent models which gives state of the art results in many NLP tasks. Examples of context dependent models are ELMo and BERT.

References

[1] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, 2013.

[2] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[3] Gerard Salton. Some experiments in the generation of word and document associations. In *Proceedings of the December 4-6, 1962, Fall Joint Computer Conference*, AFIPS '62 (Fall), pages 234–250, New York, NY, USA, 1962. ACM.

[4] Paramveer Dhillon, Dean P Foster, and Lyle H Ungar. Multi-view learning of word embeddings via cca. In *Advances in neural information processing systems*, pages 199–207, 2011.

[5] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[6] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[11] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, April 2017.

[12] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[13] `https://en.wikipedia.org/wiki/Bag-of-words_model#:~:targetText=The%20bag%2Dof%2Dwords%20model,word%20order%20but%20keeping%20multiplicity.`

[14] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[15] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[16] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[17] `https://mc.ai/brief-review-of-word-embedding-families-2019/`.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.