Multimedia Tools and Applications (2024) 83:37979–38007
https://doi.org/10.1007/s11042-023-17007-z

A detailed review on word embedding techniques with emphasis on word2vec

S. Joshua Johnson[1] · M. Ramakrishna Murty[1] · I. Navakanth[2]

Received: 29 March 2022 / Revised: 19 September 2022 / Accepted: 11 September 2023 /
Published online: 3 October 2023

Abstract
Text data has been growing drastically in the present day because of digitalization. The Internet, being flooded with millions of documents every day, makes the task of text processing by human beings relatively complex, which is neither adaptable nor successful. Many machine learning algorithms cannot interpret the raw text in its original format, as these algorithms purely need numbers as inputs to accomplish any task (say, classification, regression). A better way to represent text for computers, to understand and process text efficiently and effectively is needed. Word embedding is one such technique. Word embedding, or the encoding of words as vectors, has received much interest as a feature learning technique for natural language processing in recent times. This review presents a better way of understanding and working with word embeddings. Many researchers, who are non-experts in using different text processing techniques, would not know where to start their exploration due to a lack of comprehensive material. This review provides an overview of several word embedding strategies and the entire working procedure of word2vec,both in theory and mathematical perspectives which provides researchers with detailed information so that they may rapidly get to work on their research. Research results of standard word embedding techniques have also been included to better understand how word embedding have been improved from the past years to most recent findings.

Keywords Word Embeddings · word2vec · Neural Networks · NLP

1 1 Introduction

Applications of Natural Language Processing (NLP) include Analyzing sentiments, Classifying the documents, Clustering, E-mail spam detection, chatbots, summarization, similarity, relevance judgment, and many more. All of these applications require text processing. With ample information present in the text format, extracting knowledge and turning it into applications is critical [1]. For machines to process text, the vocabulary's free-form text terms must be transformed to numeric values (vectors). Different machine learning

✉ S. Joshua Johnson

[1] CSE, Anil Neerukonda Institute of Technology and Sciences(Autonomous), Visakhapatnam, India

[2] CSE, Maturi Venkata Subba Rao (MVSR) Engineering College(Autonomous), Hyderabad, India

techniques are more commonly used by different enterprises to exploit text data for automating processes, predictive modelling, knowledge discovery, and understanding. One of the significant issues is converting free-form text into a representation that machine learning algorithms can efficiently utilize.

One of these transformations is one-hot encoding, which converts every word in the vocabulary into a unique index in a vector consisting of V different elements. As a result, a word is designated by a vector of zeros (0)except one (1) in the appropriate position. The degree of dimensions is determined by the number of words in the source vocabulary. The size of the vector in this method is equal to the total words in the vocabulary. In contrast to the simplicity of the one-hot encoding technique, It cannot convey the meaning of a certain word and the difference between words. As the current scenario demands extensive data processing, one-hot encoding has become infeasible due to its colossal memory requirements to represent data in thousands of dimensions. Besides, this technique does not hold any hidden relationship among words in the vocabulary. Overcoming this shortfall needs different word representation methods.

Word embedding differs from the previously stated conventional embedding technique. The one-hot encoded vectors are mapped to dense representations using word embedding methods. Word embedding approaches, which capture the semantics of linguistic data, often have a lower dimensionality than vocabulary size. The theory is that acquiring more dense word representations allows for better word prediction. In n-dimensional vector space of words, for instance, "Doctor," "Tomato," and "Physician" are dispersed. The spacing in between the words "Doctor" and "Physician" is less than the distance in comparison to "Doctor" and "Tomato," demonstrating that the words "Doctor" and "Physician" are comparable. This technique establishes a hidden semantic connection between the words.

The 'distributional theory' [2] posits that words with comparable or related meanings occur in the same context. As a result, embeddings for syntactically or semantically connected words will be nearer together in the vector space when compared to terms that are not related. This relationship is completely dependent on the textual data, or corpus, from which these embeddings are produced.

Implementing NLP tasks using these simple techniques has their own limitations, as in the speech recognition, limited by the relevant inter-domain data availability where the performance has its dominance with respect to quality in speech data transcription. Availability of corpora for different languages is comparatively less and scaling up of available techniques in not just sufficient. Hence, there is a need to develop more advanced techniques.

To alleviate the above challenges, this research has its major contribution to review the research on improvised word embeddings, which helps perform different NLP tasks more efficiently.

The contributions in this research are summarized as follows:

- Advancements in machine learning techniques paved a path to build more complex models, and NLP tasks require a sophisticated technique to learn word vectors from datasets with millions of words in vocabulary. Aim of this research is to review the advanced techniques made available to generate high-quality word vectors for efficient word embedding generation.
- Previously, feature engineering in Natural Language Processing (NLP) required the creation of appropriate numerical functions to capture significant textual properties like the noun-to-pronoun ratio. This strategy, on the other hand, needs domain knowledge as well as time and effort to find crucial and meaningful traits. Word embeddings, on the other hand, are learned directly from the text corpus without the need for man-

ual labelling or feature extraction/engineering. Thus, this technique helps in achieving word embeddings in an unsupervised manner.

- A model has been proposed that aims to maximise the accuracy of generating appropriate word vectors by preserving linear regularities in terms of both syntactic and semantic space.

Frequency word embeddings, static word embeddings, contextual word embedding, and sentiment-aware embedding methods are the several techniques used for word embedding.

The count vector, TF-IDF, and co-occurrence matrix are examples of frequency-based embeddings. Co-occurrence methods are further classified into Latent Semantic Indexing (LSI), Probabilistic LSI (PLSI), and Latent Dirichlet Allocation (LDA).

word2vec, Glove, and FastText are examples of static word embedding approaches. Elmo, GPT-2, and BERT are the three types of contextualized word embeddings. Enhancement of contextualized embedding methods is carried out in Sentiment aware methods. The techniques are represented in Fig. 1.

This review focuses on several word embedding approaches, how different techniques are used to represent text, and how different techniques are used to determine text similarity, with an emphasis on word2vec.

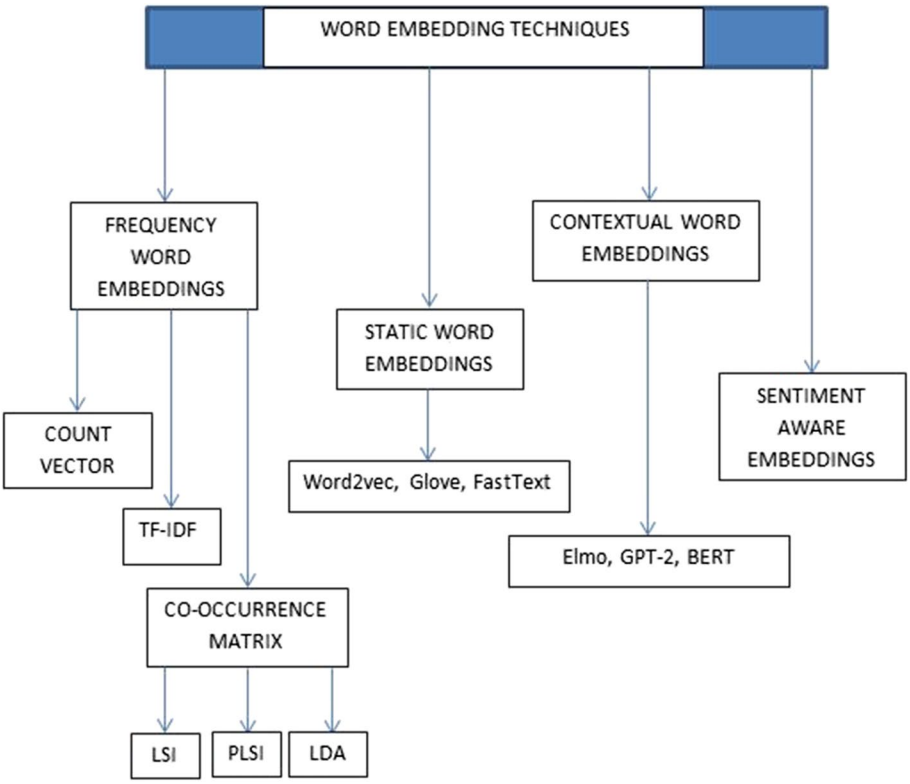

Fig. 1 Diagrammatic representation of different word embedding techniques

The remainder of the paper is organised as follows. Section 2 reviews the literature on various word embedding approaches, as well as their advantages and disadvantages. Section 3 provides a thorough knowledge of the word2vec model, as well as a precise mathematical interpretation. Section 4 discusses the benefits and drawbacks of various word embedding strategies. Section 5 provides a comparison of word embedding strategies utilising various data sets. Section 6 discusses the applications and limitations of word2vec. Lastly, Section 7 summarizes the work and outlines future research directions.

2 Literature survey

Following a thorough review of the literature, it was discovered that there are four main word embedding techniques: Frequency-based embeddings, static word embeddings, Contextualised word embeddings, and sentiment-aware methods. A survey on these techniques is presented in the following subsections.

2.1 Frequency-based word embeddings

Frequency-based or traditional word embedding methods are based according to the prevalence of words occurring in the document. These techniques determine the significance of unique terms in a document and tally their occurrence and co-occurrence.

Embedding is a technique in which vectors are created from textual information. The BoW, TF-IDF are two popularly used embedding techniques for text transformation into numeric vectors. In Bag of Words representation, every sentence in the document is represented by word vectors. The vocabulary is made up of all the unique terms present in the document, with the total number of dimensions equal to the total number of unique words. Every dimension in the vector space corresponds to the frequency of the terms for every unique word within the given text corpus, within a hash-table [32] or a dictionary. The BoW is easy to implement, but it faces serious challenges. If this method generates vectors for large documents, the resultant vector space will be a high dimensional sparse vector. Around 600,000 distinct words are present in the Oxford English dictionary. It is sufficient to assert that the majority of written material can be expressed using a small vocabulary (e.g., 5000–10000 distinct words). For example, the adjectives appealing, charming, dazzling, fascinating, fine, graceful, majestic, and fantastic are all synonyms for "beautiful," but BoW is incapable of capturing synonymy. A document containing the term "fascinating" will be differentiated from one containing the phrase "dazzling." As a result, a classifier trained to predict document sentiment needs to be exposed to a large number of labeled samples to learn that all of these terms predict positive sentiment. Apart from this, information regarding the grammar and word orderings cannot be retained, alongside giving similar vector representations to the sentences carrying different meanings, making poor performance on identifying the context and sense of the given data.

TF-IDF is the following method. In this popular scheme [33], a vocabulary of "terms" or "words" is chosen, and then every word occurrence is counted for each document in the text corpus. Term Frequency is a metric that indicates how often a term t appears in a document d [34], which is given by,

$$TF_{t,d} = \frac{n_{t,d}}{Total\ number\ of\ terms\ in\ the\ document} \tag{2.1}$$

$$TF = Term\ Frequency \tag{2.2}$$

$n-$ Count of appearance of term "t" in the document "d".

Following suitable normalisation, the term frequency count was compared to the inverse document frequency count, which is given by,

$$IDF_t = \log \frac{number\ of\ documents}{number\ of\ documents\ with\ term\ 't'} \tag{2.3}$$

$$IDF = Inverse\ Document\ Frequency \tag{2.4}$$

The IDF count is a log scale measure of the total number of times a term appears in a corpus, with the values suitably normalised.

Finally, the TF-IDF score is computed for every word in the corpus using:

$$(TF\_IDF)_{t,d} = TF_{t,d} * IDF_t \tag{2.5}$$

As a result, term-by-document matrix X is obtained, where the columns of the matrix contain TF-IDF values for the documents in the corpus. Words with higher TF-IDF scores are regarded as more significant, while those with lower scores are considered less critical. An observation is that the TF-IDF scheme helps us reduce the documents of varying lengths to fixed-length lists of numbers.

Despite its simplicity and efficiency, TF-IDF uncovers only a little of the connections between terms, thereby failing to catch some fundamental linguistic ideas like synonymy and polysemy. Also, it gives only a small amount of reduction in the description length. Several dimensionality reduction techniques have been developed by the researchers, like, LSI, PLSI, and LDA to alleviate these issues.

2.1.1 Methods based on matrix factorization

Matrix factorization methods have always been essential in many NLP tasks [3]. Factorizing word Co-occurrence matrix is essential for two reasons. First, it creates word-embeddings [4, 5], which are low-dimensional representations of vocabulary that contain topical and semantic meaning. Second, it contains representations of meaning that extend beyond single words to latent topics.

The objective of matrix factorization is to compress the most relevant information from a high-dimensional input matrix to several low-dimensional factor matrices, each having either estimated or accurate input reconstruction.

Factorization techniques are useful in high-dimensional and sparse data frameworks for lowering sparsity, the total amount of features, and noise. These techniques can also retrieve data's hidden features.

Latent semantic indexing (LSI) One of the most accepted approaches to matrix factorization is the LSI method [6], which is also called Latent Semantic Analysis (LSA). This technique tries to overcome the problems suffered by TF-IDF. When Singular Value Decomposition (SVD) [35] is applied to TF-IDF, latent semantic space is created, which retains the majority of the variances in the corpus. Each feature in the new space is a linear combination of the actual TF-IDF features, resolving the synonymy issue. This kind of approach achieves significant compression in a large corpus. LSI is a technique used to

analyze a given set of documents to identify statistical word co-occurrences that appear together, providing insights into the topics of those words and documents. The two major problems that LSI solves are synonymy and polysemy. Synonymy refers to how many ways through which the same object can be referred. Searching for "Most Beautiful woman of the year 2021" is similar to searching "Most Stunning woman of the year 2021". The next is polysemy. The fact that most words have more than one different meaning is referred to as polysemy. The word "bank," when used by different people in different contexts, takes varying meanings. When the word "bank" is accompanied by the word "river" in a document, it is statistically probable that the word "bank" is referring to a river, where a poet is thinking of writing some poems. Also, as already specified, [6] contends that LSI features are linear combinations of the original TF-IDF features. These features can capture minor aspects of basic linguistic ideas like synonymy and polysemy. A computer can give the correct answer for a user query if it can understand how words occur together. In addition to noise reduction, LSI factorises the term-document matrix to uncover latent characteristics of data. The term-document matrix is essentially large, with Co-occurrence data in the matrix being noisy. As a result, reducing the dimensionality of data can also aid in the reduction of noise.

LSI is quick and efficient to implement, but it has a few major limitations. One notable disadvantage of LSI [31] is that once the semantic set has been formed, it cannot handle freshly added documents. SVD employs all linear relations in the dimensions supplied to it to generate vectors that forecast every text sample in which the word appears, so changing any cell value changes the coefficient in every other word vector. It has interpretable embeddings (which means that components can be arbitrarily positive or negative). To get accurate results, it requires a large number of documents and vocabulary. Its representation is inefficient. LSI is a more traditional indexing approach designed for smaller static databases. A significant limitation to LSI is that if a corpus is added with the updated(new) content, indexing the entire corpus is required, thereby making it of limited use for dynamically changing corpus.

Probabilistic LSI (PLSI) Using LSI as a foundation, probabilistic LSI (PLSI) [7] has been proposed to improve the expressiveness of LSI. This technique is another widely used document model. Given an unobserved topic z, the PLSI model proposes that the document label d and a word w_n are conditionally independent:

$$p(d, w_n) = p(d) \sum_z p(w_n|z)p(z|d) \tag{2.6}$$

PLSI solves the problem utilising a probabilistic method rather than the Singular Value Decomposition (SVD). The key idea is to construct a probabilistic model with hidden themes that generates the data seen in the document-term matrix [8].Each PLSI document contains several subjects, and each word is assigned to one of them. PLSI adds a probabilistic approach of topics and words to LSI.

PLSI, despite being a significantly more flexible model, nevertheless has several issues. It suffers from a lack of probabilistic models to represent related topics. Another issue to consider is that the total number of PLSI parameters grows linearly with the number of documents, making overfitting possible.

Latent dirichlet allocation (LDA) The Latent Dirichlet Allocation (LDA) approach was proposed in 2003[9] to solve this problem. In contrast to PLSI, each topic is a probabilistic

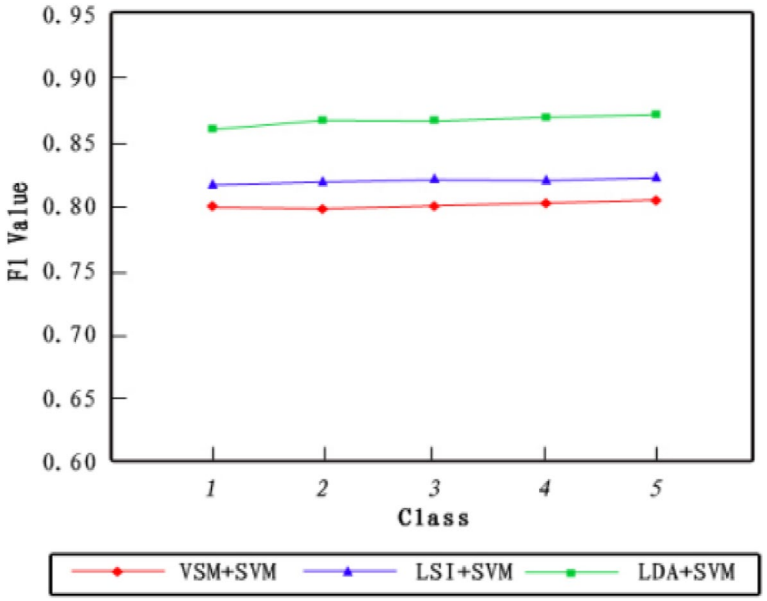

Fig. 2 Classification methods-Comparison. Source [37]

Table 1 Macro_average and micro_average comparison. Source [37]

Methods	Macro_P	Macro_R	Macro_Fl	Micro_Fl
VSM + SVM	0.804153	0.805347	0.805023	0.80574
LSI + SVM	0.825966	0.822337	0.82398	0.828164
LDA + SVM	0.865325	0.86968	0.864919	0.868738

distribution over words. LDA is, in fact, a Bayesian version of PLSI. The use of Dirichlet priors for document-topic and word-topic distribution, in particular, increases generalisation. Human-interpretable topics can be extracted from a document corpus using LDA, with each topic being defined by the words with which it is most likely associated.

Another contender for generating word embeddings is Principal Component Analysis (PCA)[10].In this approach, dimensionality reduction of word co-occurrence matrix is made by employing PCA. However, Instead of Euclidean distance, Hellinger distance has been used for error correction. Another matrix factorization-based method, Global Vector (Glove) [11], is frequently used for word embeddings generation using a global word co-occurrence matrix. This method is based on utilising an optimization procedure to anticipate the co-occurrence matrix entries. Glove varies from other matrix factorization-based methods in that it is more frequently known as a predictive method. Comparative analysis of all the three techniques is given in the Fig. 2 and Table 1.

2.2 Static word embeddings

A static word embedding function converts each word into a vector. In comparison to vocabulary size, these vectors are dense and have a lower dimensionality. These embeddings are static in a way such that the embedding assumes a fixed size vocabulary.

Another group of researchers employs Neural Networks for word embeddings generation. Neural Network approaches are predictive, which means that they forecast an acceptable output for a given input data.word2vec [12] is a very well-known model which uses two-layer feedforward neural network architecture. The vector representation of words that the word2vec model learns holds semantic meanings and plays a vital role in various NLP tasks. Work done by [12] is summarized in Table 2.

Many researchers who want to use word2vec or similar techniques do not have detailed information explaining the parameter learning process. Though [13] clearly explains the parameter learning process of word2vec, it is pretty tricky for a person with basic knowledge to understand the process.

This review mainly focuses on a detailed, step-by-step explanation of the word2vec model, which would help beginners understand even more clearly. This discussion will be the basis for understanding all other advanced techniques to word2vec and will be discussed in the next section.word2vec is applied in two ways: a continuous bag of words (CBOW) and skip-gram models. The CBOW model predicts the target word based on context words, whereas the skip-gram model predicts context words based on the target word.

CBOW and skip-gram both produce dense and low-dimensional word vector representations that can be used to solve diversified Natural Language Processing challenges. These approaches produce vectors with a significant flaw: they only have local word co-occurrence information and no global co-occurrence information. Local and global information are merged as input to a two-layer neural network in [14]. Graph embedding was used in [15] to combine local and global co-occurrence counts. The nodes in the graph represent words, and the weight of each edge is determined by a combination of local and global co-occurrence counts. When compared to word2vec and GloVe, the proposed graph outperformed the others

Table 2 Semantic and syntactic questions in the Semantic-Syntactic word relationship test set. Source [12]

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	Kwanza	Iran	Rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	grand daug hter
Adjective to adverb	Apparent	Apparently	Rapid	Rapidly
Opposite	possibly	impossibly	Ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	Walking	Walked	Swimming	Swam
Plural nouns	Mouse	Mice	Dollar	Dollars
Plural verbs	work	works	speak	speaks

in terms of word similarity and word. It is worth noting that the GloVe model was created to address the constraints of word2vec. In [16–18], clustering is applied in the context of a word, using a cluster-based technique. In this case, the word sense is represented by the centroid of each cluster. [19, 20], and [21] make an attempt to ameliorate on the original word2vec model for producing multi-sense word embeddings.

2.3 Contextual methods

Many researchers have recently focused on contextualised word embedding algorithms, such as Elmo[22], Bert[23], and Xlnet[24]. These techniques are predicated on the premise that the essence of a word is provided in context, necessitating a unique embedding for each word per context. Fine-tuning can be used in count-based and neural network-based approaches for creating high-quality word embeddings for Natural Language Processing jobs to address the problem of polysemy.

2.4 Sentiment aware methods

Techniques that produce word embeddings based on context cannot tell the difference between words that have the same context but distinct polarity. Take, for example, the words "good" and "bad." Because these two terms are commonly used in similar situations, they are assigned vectors in the vector space that are close together. However, in the context of sentiment analysis, the words "good" and "bad" have opposite polarity and must thus be separated in vector space. Various strategies have been developed to overcome the difficulties of word embedding models. [25] To integrate sentiment information, a hybrid approach of supervised and unsupervised learning is proposed. Two models are employed in [27] to improve the skip-gram model by accounting for sentiment information. A pre-trained word vector was redefined [28] based on polarity. [29] Presents a strategy for upgrading pre-trained GloVe and word2vec word vectors for sentiment analysis. To improve pre-trained word vectors, a vector was developed and incorporated for each part of speech tag, word location, and polarity. [30] Developed two unsupervised models that integrate word polarity information and word co-occurrences into a tensor, and then build word embeddings using tensor factorization, which yielded a greater performance.

3 Word2vec: parameter learning

As stated in section 2.2, the continuous bag of words (CBOW) model and skip-gram model are the two ways word2vec can be applied. This section goes over each of these models in depth (for beginners) to understand parameter learning and gives an overview of advanced optimization techniques like hierarchical softmax and negative sampling. The detailed interpretation of every equation has been added along with their mathematical derivations. The base for discussion in this section, including mathematical expressions, is taken from [13] and explained in a detailed manner for the beginners to understand effectively.

3.1 Continuous bag-of-words model

3.1.1 One-word context

First, we look at the most straightforward variation of the Continuous bag-of-words model (CBOW) introduced in [26]. Here, the assumption is that only one word per given context is considered, meaning that, given one context word, the model predicts one target word, similar to a bigram model. Before going deeper into the model, let us first discuss the preliminary information about the model.word2vec uses a fully connected neural network with a single hidden layer. The neurons in the hidden layer are Linear Neurons. The count of neurons in the input layer is equal to the count of words in the vocabulary used for training purposes. The size of the hidden layer is equal to the dimensionality of word vectors, denoted by N, implying that N is the total number of dimensions utilised to represent our word. This is an arbitrary value and a hyperparameter for the Neural Network. The total number of neurons in the output layer equals the total number of neurons in the input layer Fig. 3.

Visualization:

Input Layer:

Here, x1=word 1, x2=word 2, and xv=word v.

V=Total number of words in corpus.

Hidden Layer:

Here, the Input-to-Hidden-Layer Fig. 4 connection is represented by: WV×N

$$W_{V\times N} = \begin{bmatrix} w_{11} & w_{12} \cdots & w_{1N} \\ w_{21} & w_{22} \cdots & w_{2N} \\ \vdots & \vdots\vdots & \vdots \\ w_{V1} & w_{V2} \cdots & w_{VN} \end{bmatrix}$$

Here,

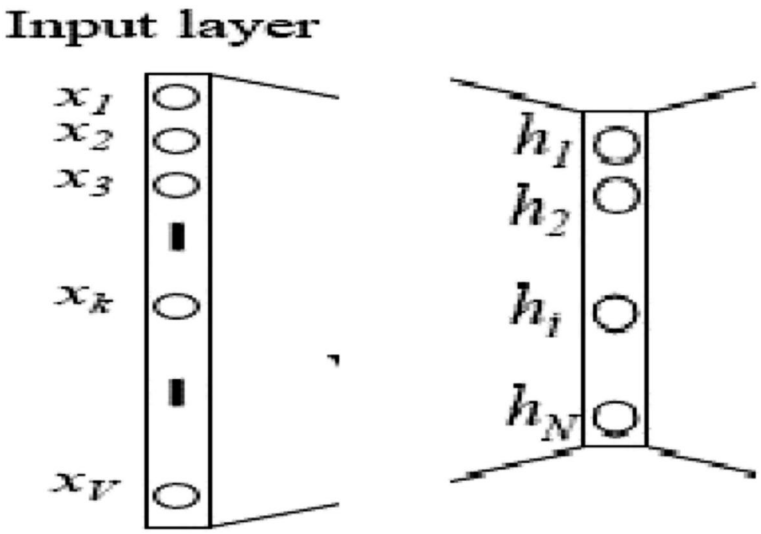

Fig. 3 Input layer in the CBOW model with one word context

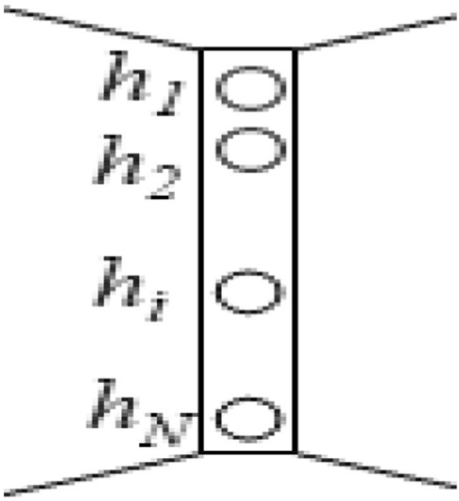

Fig. 4 Hidden layer in the CBOW model with one word context

V=Number of words in corpus, which is the input size.

N=Number of Neurons in Hidden Layer, which is arbitrary.

Hidden layer size determines the word vectors size.

Denoting the matrix $W_{V\times N}$ as W_1.

The weight matrix W_1 stores the information or patterns that the neural network has learned when trained. The weight matrix W_1 is called Embedding Layer. During training, multiplication of input layer (input vector) with first weight matrix (W_1) occurs. So, when multiplying the input vector (one-hot encoded vector) with the weight matrix,

i.e.,

$$[\text{Input vector}] \times W_1 = \text{Hidden Layer Values}$$

Here, the value of hidden layer will be the value which corresponds to 1 in the weight matrix. When a weight matrix is multiplied by a one hot encoded vector, only one row in the weight matrix retains its value, while all other rows become 0.This is illustrated in Fig. 5.

As a result, when a specific word is used, such as the n^{th} word in the vocabulary, the effect is limited to the n^{th} row in the weight matrix. This is because all other rows become 0.

As a result, the n^{th} row of the weight matrix contains all the trained information about the n^{th} word in the vocabulary. This is how the word vectors from a trained word-2vec Neural Network are retrieved.

Network model for one-word context definition The Fig. 6 shows the network model, with simplified definition:

The above architecture has Single Input Layer, single hidden layer (a=1, means having a linear activation function), one output layer(having softmax as activation function).

The size of the Vocabulary is V, and the size of the Hidden Layer N. Adjacent layer units are fully connected layers. Input is one-hot encoded vector, means, out of V units,

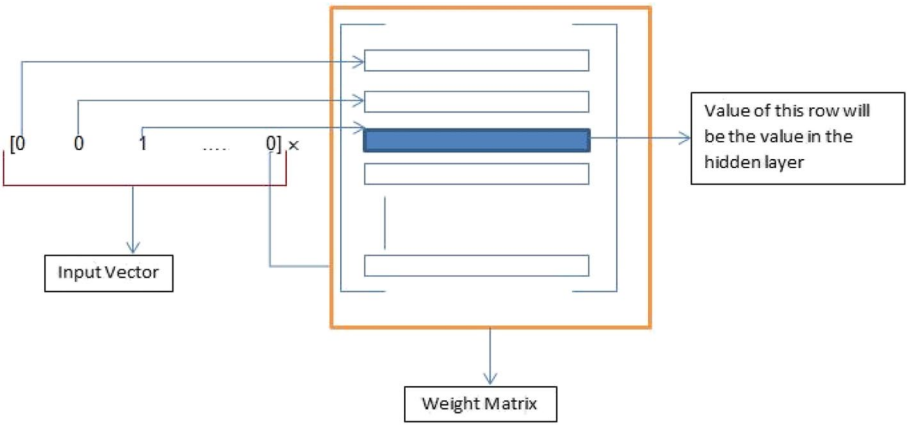

Fig. 5 Visual Representation of Hidden layer computation

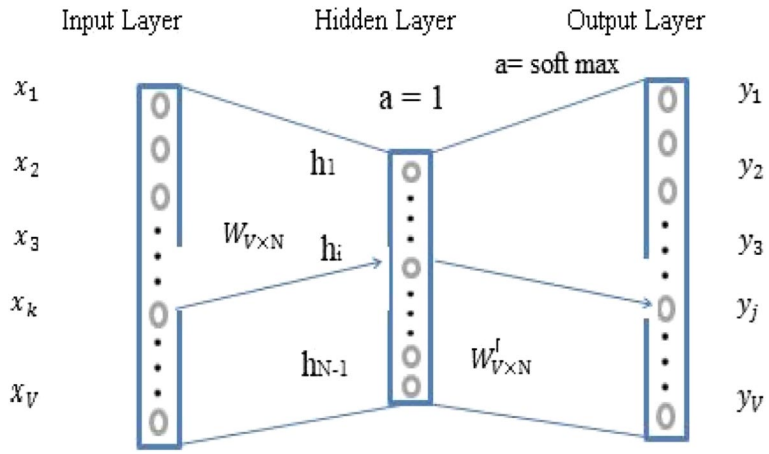

Fig. 6 CBOW Model – Single word context

$\{x_1 \ldots xv\}$, only one unit will be 1, and remaining units are 0.The output layer has a V-dimensional vector.

The V × N matrix can be used to represent the weights between the input and hidden layers, which is denoted by W. Every row of W matrix is N-dimensional vector. v_w is the N-Dimensional vector representation of a particular(associated) word of Input Layer. More formally, i^{th} row of W can be written as v_w^T.

Given a context (context word), assuming x_k=1 and $x_{k'}$=0 for $k' \neq k$, then

$$h = W^T x = W^T_{(k,.)}$$

This means, k^{th} row of W-matrix is copied to the hidden layer matrix.

As the k^{th} row is being copied to h, and, as only one word is considered, that is the reason vector representation of the word is used as v.

$$\text{So}, h = W^T \text{x} = W^T_{(k,.)} = v^T_{w_I} \tag{1}$$

The vector representation of the input word w_I is v_{w_I}.
From Eq (1),

$$h = v^T_{w_I}$$

This means that the hidden layer's activation function is linear, as the weighed sum of inputs is directly passed to the hidden layer.

$W^{'}$ Represents a different weight matrix from the hidden layer to the output layer.
Where,

$$W^{'} = \{w_{ij}{}'\}, \text{ which is aN } \times \text{ V matrix}$$

Now, using these weights from $W^{'}$, the score u_j can be computed for every word in vocabulary,

$$u_j = (v_{w_j}{}')^T (h) \tag{2}$$

where, $v_{w_j}{}'$ is j$^{\text{th}}$column of weight matrix $W^{'}$.The softmax function, which is a log-linear classification model, is utilised to obtain the posterior distribution of words.
i.e.,

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_j{}')} \tag{3}$$

Here, y_j is the output of j$^{\text{th}}$ unit in output layer.
$p(w_j|w_I)$: Given input wordw_I, find the probability of word w_j, to be nearer to input word w_I
.
Substituting (1) and (2) in Eq (3),

$$p\left(w_j|w_I\right) = \frac{\exp((v_{w_j}{}')^T v_{w_I})}{\sum_{j'=1}^{V} \exp((v_{w_{j'}}{}')^T v_{w_I})} \tag{4}$$

A point to note here is, the word w has two representations, namely, v_w and $v_w{}'$
v_w : It is derived from rows of W (Input layer to hidden layer weight matrix).
$v_w{}'$: It is derived from columns of $W^{'}$(Hidden layer to output layer weight matrix.)
Also,
v_w : Termed as **input vector** of word$w$.
$v_w{}'$: Termed as **output vector** of word$w$.
Equation for hidden → output weights Updation
The objective of training (for single training sample) is to maximize Eq. (4). It means, given the input context word w_I, The goal of training is to maximize the conditional probability of observing the actual output wordw_o.
i.e.

$$\max p\left(w_o|w_I\right) = \max y_{j^*} \tag{5}$$

y_{j^*} = Denoting the output word w_o index in the output layer as j^*

From (5),

$$\max p\left(w_o|w_I\right) = \max y_{j^*}$$

$$\max p\left(w_o|w_I\right) = \max \log y_{j^*} \tag{6}$$

Equation (5) is modified to Eq. (6) by adding log. Why to take log in Eq. (6)? Generally, in optimizing problems,*max*log probability is taken instead of simple probability.

This is due to the following reasons:

In most machine learning tasks, while formulating some probability p, which should be maximized, the log probability (log p) is optimized instead of simple probability.

Generally, gradient methods work better optimizing log p(x), rather than simple p(x), because, the gradient of log p(x) is more well scaled. It aids the objective function in selecting an appropriate step size and achieving the best results in fewer steps.

From Eq (6),

$$\max p\left(w_o|w_I\right) = u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) = -L \tag{7}$$

Here,$L = -\log p\left(w_o|w_I\right)$ is the loss function and, minimizing this loss function L is the objective.

In the output layer, the index of the actual output word is j^*.

Now see, how the Eq (7) is obtained, from Eq (6) with detailed analysis.

From the above discussion,

Input data: X

$$\text{Hiddenlayer} h = W^T X \tag{8}$$

u = Value of output before applying softmax function.

$$\text{i.e.} u = W^{'T} h = W^{'T} W^T X$$

$$y = softmax(u) = softmax(W^{'T} W^T X)$$

Assuming that the model is trained with (target, context) word pairs, represented by (w_o, w_I)

Here, w_o : Target word, represented by one-hot encoded vector.

Assuming w_o is having a value 1 at a position j^*, then it will have value 0 at all other positions.

What should loss function do now? The loss function should evaluate the output layer at position j^*, which means y_{j^*}.

Values in softmax are known to be interpreted as conditional probabilities of the target word, given the context word w_I.

Loss function can now be defined as,

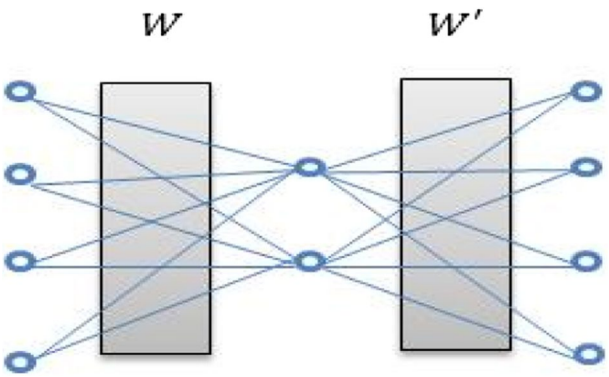

Fig. 7 Visualizing the weight matrices W and $W^{'}$

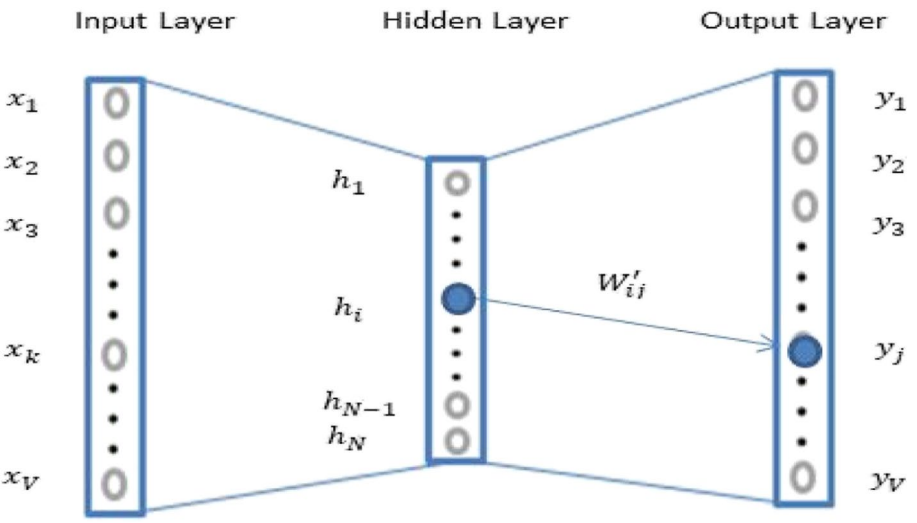

Fig. 8 Hidden-Output layer connection

$$\begin{aligned} L &= -\log p(w_o|w_I) \\ &= -\log y_{j^*} \\ &= -\log[softmax(u_{j^*})] \\ &= -\log\left(\frac{\exp u_{j^*}}{\sum_i \exp u_i}\right) \\ &= -\left[\left(\log\exp\left(u_{j^*}\right)\right) - \log\left(\sum_i \exp u_i\right)\right] \\ &= -u_{j^*} + \log\sum_i \exp u_i \end{aligned} \tag{9}$$

The objective is to:

i) find the weight matrices Fig. 7 and Fig. 8 W and $W^{'}$ such that, Loss function is minimized,

ii) Given a context word, the models probability to predict the target word should be maximized.

The model should learn the weights. Using gradient descent approach to tackle this situation is appreciated.

Here,how much change in W and $W^{'}$ effects the total loss has to be identified.

This in turn means, finding the derivatives.

$\frac{\partial L}{\partial W}$ (Gradient w.r.t W) and $\frac{\partial L}{\partial W'}$ (Gradient w.r.t $W^{'}$)

From chain rule of multivariate functions,

$$\begin{aligned}\frac{\partial L}{\partial W_{ij}{}'} &= \frac{\partial L}{\partial u_1}.\frac{\partial u_1}{\partial W_{ij}{}'} + \frac{\partial L}{\partial u_2}.\frac{\partial u_2}{\partial W_{ij}{}'} + \cdots + \frac{\partial L}{\partial u_k}.\frac{\partial u_k}{\partial W_{ij}{}'} \\ \frac{\partial L}{\partial W_{ij}{}'} &= \sum_{k=1}^{V} \frac{\partial L}{\partial u_k}.\frac{\partial u_k}{\partial W_{ij}{}'}\end{aligned} \tag{10}$$

And,

$$\frac{\partial L}{\partial W_{ij}} = \sum_{k=1}^{V} \frac{\partial L}{\partial u_k}.\frac{\partial u_k}{\partial W_{ij}} \tag{11}$$

Consider, Eq (10)

$$\frac{\partial L}{\partial W_{ij}{}'} = \sum_{k=1}^{V} \frac{\partial L}{\partial u_k}.\frac{\partial u_k}{\partial W_{ij}{}'}$$

In the above Figure, the weight $W_{ij}{}'$ is the element of the matrix $W^{'}$

$W_{ij}{}'$: Connects hidden layer node i and output layer node j.

This affects the output score u_j and also y_j. Remember, y_j is obtained after applying $softmax(u_j)$.

The only thing that is being effected here is the output score u_j, so, from all the derivative values of $\frac{\partial u_k}{\partial W_{ij}{}'}$, only at the point$k = j$, a value other than zero(0) is found.

All the other values of $k \neq j$ will be zero.

So, from among the derivatives of Eq (10), derivative value remains only at the point $= j$,

So,

$$\frac{\partial L}{\partial W_{ij}{}'} = \frac{\partial L}{\partial u_j}.\frac{\partial u_j}{\partial W_{ij}{}'} \tag{12}$$

Considering the first term from the Eq (12).

Calculate the value of $\frac{\partial L}{\partial u_j}$

$$\frac{\partial L}{\partial u_j} = -\delta_{jj^*} + y_j \tag{13}$$

where, δ_{jj^*} is kronecker delta.

From Eq (13), $\delta_{jj^*} = 1$ (only when $j = j^*$).

Which means, if j^{th} unit is the actual output word, that is, if the j value referred in Eq (12) is.

equal to j^* (our actual output word),

Then,

$$\delta_{jj^*} = \delta_{j^*j^*} = 1$$

If this is not the case, then,$\delta_{jj^*} = 0$ when $j \neq j^*$

Revisiting the Eq (13), $\frac{\partial L}{\partial u_j} = -\delta_{jj^*} + y_j$,
Can be rewritten as, $\frac{\partial L}{\partial u_j} = y_j - \delta_{jj^*}$
This derivative is the output layer's prediction error,e_j.
So, the above equation can be rewritten as,

$$\frac{\partial L}{\partial u_j} = y_j - \delta_{jj^*} = e_j \tag{14}$$

The difference between the target and the predicted output, which is the prediction error vector, is represented by e_j.
Taking the Second term from the Eq (12).

$$\frac{\partial u_j}{\partial W_{ij}'} = \sum_{K=1}^{V} W_{ki} x_k$$

$\frac{\partial u_j}{\partial W_{ij}'}$: Change in u_j w.r.t change in W_{ij}'
$\frac{\partial u_j}{\partial W_{ij}'} = h_i (\because h_i = \sum_{K=1}^{V} W_{ki} x_k)$.
Substituting Eq (14) and (hi) in Eq (12), we get,

$$\frac{\partial L}{\partial W_{ij}'} = e_j \cdot h_i \tag{15}$$

Finally, using stochastic gradient descent, the weight updation equations for hidden → output weights W_{ij}' are as follows:

$$w_{ij}^{'(new)} = w_{ij}^{'(old)} - \eta \cdot e_j \cdot h_i \tag{16}$$

Or

$$v_{w_j}^{'(new)} = v_{w_j}^{'(old)} - \eta \cdot e_j \cdot h \text{ for j} = 1, 2, \cdots, \text{V} \tag{17}$$

where,
η is the learning rate parameter ($\eta > 0$),

$$e_j = y_j - \delta_{jj^*}$$

$= y_j - t_j$ (Renaming δ_{jj^*} as t_j for simplicity)
h_i = Hidden layer's i$^{\text{th}}$ unit.
v_{w_j}'=w_j's output vector.
Points to note:
Iterating through every possible word in vocabulary is needed, and then, check the output probability of the word y_j and compare y_j with expected output t_j(which is 0 or 1).
i) If $y_j > t_j$(overestimating), then,
$v_{w_j}' - v_{w_I}$ (i.e., $v_{w_j}' - h$) is done
ii) If $y_j < t_j$ (underestimating), then,
$v_{w_o}' + h$ (to make v_{w_o}' closer to v_{w_I}) is done
iii) If $y_j \cong t_j$, then,
According to update Eq (16), only a little change is done to weights.
Input → hidden weights Updation:

After obtaining the update equations for W', moving further to get the update equation for W,

Step 1: Taking derivative of Loss function L, with respect to, output of hidden layer,

$$\begin{aligned}\frac{\partial L}{\partial h_i} &= \sum_{j=1}^{V} \frac{\partial L}{\partial u_j}.\frac{\partial u_j}{\partial h_i} \\ &= \sum_{j=1}^{V} e_j.\frac{\partial u_j}{\partial h_i} \text{fromEq(14)} \\ &= \sum_{j=1}^{V} e_j.w_{ij}' \\ &= LH_i\end{aligned} \tag{18}$$

h_i –hidden layer's ith unit output.

u_j –jth unit net input in the output layer

$$e_j = y_j - t_j$$

The jth word in the output layer has a prediction error, given by e_j.

LH: Vector of N-dimensions, that represents the output vectors sum, of all words in vocabulary, which is weighted by their prediction errore_j.

Step 2: Calculate the derivative of L on W. It is critical to note that the linear computation is performed by the hidden layer on the values collected from the input layer.

From Eq (8), the vector notation can be expanded as,

$$h_i = \sum_{k=1}^{V} x_k.w_{ki} \tag{19}$$

Obtaining the derivative of L, with respect to every element of W,

$$\begin{aligned}\frac{\partial L}{\partial w_{ki}} &= \sum_{j=1}^{V} \frac{\partial L}{\partial h_i}.\frac{\partial h_i}{\partial w_{ki}} \\ &= LH_i.x_k(\text{fromEq(18), Eq(19)})\end{aligned}$$

Which is same as the tensor product of x and LH,

i.e.

$$\frac{\partial L}{\partial W} = \text{x} \otimes LH = \text{x}LH^T \tag{20}$$

Which gives us the $V \times N$ matrix.

As there is only one Non-Zero component of x, there is only one row in $\frac{\partial L}{\partial W}$,which is a Non-Zero row whose value is LH^T, a N-dimensional vector.

Update equation for W is now obtained as,

$$v_{w_I}^{(new)} = v_{w_I}^{(old)} - \eta LH^T \tag{21}$$

Here, v_{w_I} is a row of W that is the input vector of the only context word taken into account, and it is the only row of W with a non-zero derivative. Because their derivatives are zero, all of the other rows of W will remain unchanged after this iteration.

The vector LH is the sum of all output vectors of words in the vocabulary, weighted by their prediction error $e_j = y_j - t_j$,so the eq 21 can be understood as adding a fraction of each output vector in the vocabulary to the input vector of the chosen context word.

Points to note:

1. Considering the Output Layer,
If$p(w_j)$, to become the output word is highly estimated,i.e.,
If $(y_j > t_j)$, then the input vector belonging to the context word w_I, moves far from the output vector of w_j.
2. If$p(w_j)$, to become the output word is underestimated,i.e.,

If $(y_j < t_j)$, then, the input vector of the context word w_I, moves towards the output vector of w_j.

If the probability of the word w_j can be predicted accurately, the movement of the input vector of w_I will be less affected. The prediction error of all the vectors in the vocabulary can be used to calculate the w_I input vector moment. The effect of a word on the movement of the input vector of the context word increases as the prediction error increases.

As the model's parameters are iteratively updated by going through context-target word pairings generated from a training corpus, the effects on the vector will build. After many iterations, the relative positions of the input and output vectors will eventually stabilise.

3.1.2 Multi-word context

Figure 9 depicts the CBOW model with multi-word context. To compute the hidden layer output in a single-word context, the input vector of the input context word is directly copied to the hidden layer; however, in a multi-word context, the average of all vectors of the input context words is taken, input product to hidden weight matrix, and average vector as output.

$$\begin{aligned} h &= \frac{1}{C} W^T \left(x_1 + x_2 + \cdots + x_C\right) \\ &= \frac{1}{C}(v_{w_1} + v_{w_2} + \cdots + v_{w_C})^T \end{aligned} \tag{22}$$

Here, C is the total context words.

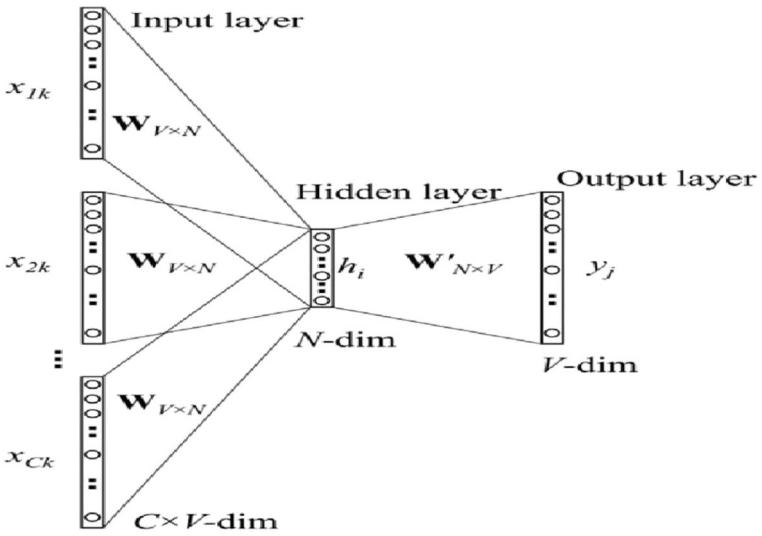

Fig. 9 CBOW Model

$w_1, w_2, \ldots, w_C$ are the context words.
Word w has the input vector v_w
The loss function can now be defined as,

$$\begin{aligned} L &= -\log p(w_O|w_{I,1}, \ldots, w_{I,C}) \\ &= -u_{j^*} + \log \sum_{j'=1}^{V} \exp(u_{j'}) \\ &= -{v_{w_o}}'^T.h + \log \sum_{j'=1}^{V} \exp({v_{w_j}}'^T.h) \end{aligned} \tag{23}$$

The hidden → output weights update equation is same as the equation for one-word-context, which is given by

$$v_{w_j}^{\prime(new)} = v_{w_j}^{\prime(old)} - \eta \cdot e_j \cdot h \text{for} j = 1, 2, \ldots, V \tag{24}$$

It's important to remember that for each training instance, the above equation must be applied to every element of the hidden → output weight matrix.

For the update equation of input → hidden weights, for every word $w_{I,C}$ in the context, apply the following equation.

$$v_{w_{I,c}}^{(new)} = v_{w_{I,c}}^{(old)} - \frac{1}{C}.\eta \cdot LH^T for c = 1, 2, \ldots, C \tag{25}$$

$v_{w_{I,c}}$ – Input vector of the word c in the input context.
η – Learning Rate Parameter(> 0).

$$LH = \frac{\partial L}{\partial h_i} \tag{26}$$

3.2 Skip-gram model

Mikolov et al. first proposed the skip-gram model (2013a,b). The skip-gram model is depicted in Fig. 10. In the skip-gram model, unlike the CBOW model, the target word will be on the input layer, while the context words will be on the output layer.

Using the same notations here also, where V_{w_I} denotes the input vector of one and only existing word in the input layer, hidden layer output is h.

The definition of h is defined as,

$$h = W_{(k,.)}^T = v_{w_I}^T$$

Remember, context words are present in the output layer, so, in this layer, there will be C multinomial distributions. The computation of each output is done using the same hidden → output matrix.

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_j')} \tag{27}$$

Here,
$w_{c,j}$ is the word j on the panel c of output layer.
$w_{O,c}$ is the actual word c in the output context words.
w_I is one and only input(target) word.

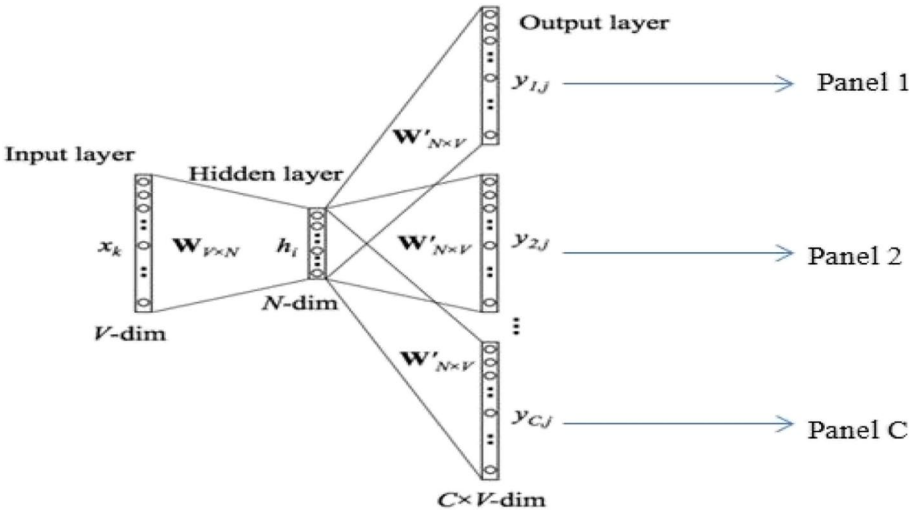

Fig. 10 Skip-gram model

$y_{c,j}$ is the output of the unit j on panel c of output layer.

is the net input of the unit j on the panel c of output layer.

As $u_{c,j}$ the panels of the output layer share common weights, and hence.

$u_{c,j} = u_j = {v_{w_j}}'^T.h$, for c = 1,2,…,C

Here, ${v_{w_j}}'$ is the output vector of the word j in vocabulary, w_j, taken from the column of hidden → output weight matrix, W'.

The derivation of parameter update equations is very similar to the one-word-context model derivations.

The loss function is given as,

$$L = -\sum_{c=1}^{C} u_{j_c^*} + C.log \sum_{j'=1}^{V} \exp(u_{j'}) \tag{28}$$

Here, j_c^* is the index of actual c-th output context word in vocabulary.

All the other derivations are similar to that of one-word-context model, where the final update equation is given by,

$$v_{w_I}^{(new)} = v_{w_I}^{(old)} - \eta \cdot LH^T$$

where, LH is N-dimensional vector, where the individual component is defined by,

$$LH_i = \sum_{j=1}^{V} LI_j.w_{ij}' \tag{29}$$

3.3 Computational efficiency optimization

The efficiency of the models discussed (the bigram model, CBOW, and the skip-gram models) can be improved by optimizing the computation of the updates for the output vectors. This optimization is done by limiting the total number of output vectors to be updated per training instance. Two approaches are widely used to achieve optimization, namely,

Hierarchical softmax and Negative Sampling. Hierarchical softmax techniques bring down [13] the computational complexity from O(V) to O(log(V)), per training instance per context word. Negative Sampling deals with the problem by updating only a sample of the output vectors per iteration.

4 Comparison of word embedding techniques

Technique	Description	Merits	Demerits
Unique numbers	Converting words into unique numbers based on the vocabulary	Easy To Implement	As random numbers are assigned, they cannot capture relationships between word pairs
One-hot encoding	Every word in the vocabulary, which is made up of V different elements, is mapped to a unique index in a vector	Simplicity in representing words as vectors	1.Computationally Inefficient 2. Relative closeness of the words are not preserved
WORD EMBEDDINGS			
Bag-of-Words (BoW)	Sentences in the document are represented by word vectors	Easy to implement	Cannot capture the order and context of the words
Count vector	Counts the total number of times(frequency representation) each word occurs in a document	Specifies frequency count, thereby helping in understanding the text	1. Incapable of differentiation more important words from less important words 2. Words with more number of occurrences in the vocabulary are considered as the significant words 3. Does not capture relationship between words
TF-IDF(term frequency-inverse document frequency)	Measure of appearance of term t, in document d	1. Helps reduce the documents of varying lengths to fixed-length lists of numbers 2.Simple and efficient	1. Uncovers only little connections between terms 2. Fails to solve synonymy and polysemy problems
Co-occurrence matrix	Similar words has a tendency to occur together	The semantic relationship between the words is maintained	The co-occurrence matrix requires a lot of memory to be stored
word2vec	Generates word embeddings using dense representations	1.Context information is preserved 2.Embedding vector size is small in contrast to huge sparse vector representations in bag of words and TF-IDF techniques	1. Cannot handle out of vocabulary(OOV) words 2. Representation at subword level cannot be captured

Technique	Description	Merits	Demerits
Glove	Unsupervised learning algorithm, training performed on aggregated global word-word co-occurrence statistics	Sub-linear relationships are captured	Model trains on co-occurrence matrix, hence need lot of memory
FastText	Derived from word-2vec.each word is represented as a bag of character n-grams	Representation for rare words	Contextual information is not added

5 Comparative analysis of word embeddings

This section gives an overview of the comparison of word embeddings in different aspects. The comparative analysis is done using three data sets, namely, WordSim353, SimLex999, and SimVerb3500. Detailed analysis of these data sets regarding word similarities and correlation Analysis is available in [36]. The data taken from [36] is summarized here to bring all the analytical information about word embeddings to a single place.

The average cosine similarities for each dataset are shown in Table 3. These findings suggest that FastText and GloVe are superior at detecting word similarities.

Because similarities acquired by one metric can be greater than those found by another, average similarity does not always convey useful information about word similarities. Despite having differing average values, both similarity distributions could be connected. As a result, for each pre-trained word embedding vector, the correlation coefficient between cosine similarities of word vectors and ground truth similarities is determined.

For each pair of ground-truth similarities and cosine similarities, the Spearman correlation coefficient (S), Pearson correlation coefficient (Pr), and Kendall's tau correlation coefficient (K) are computed as shown in Table 4.

From Table 5, it can be seen that word vectors obtained from RNN perform better on syntactic questions. NNLM has a significant improvement over RNN. CBOW, on the other hand, performs better in both syntactic and semantic cases. Skip-gram gives better performance on semantic questions and less accurate that CBOW for syntactic questions.

The results of the evaluated models are presented and compared with word vectors that are made public as shown in Table 6. It is clear that skip-gram has outperformed all other models. Table 7 shows the comparison of results for models trained for three epochs and one epoch where accuracies for both syntactic and semantic dataset is considered.

6 Applications and limitations of word2vec

6.1 Applications

Word2vec is one of the most popular types of word embeddings. word2vec takes a huge corpus of text as input and outputs a vector space with several hundred dimensions, with each unique word in the corpus allocated a matching vector in the space. These vectors are

Table 3 Average cosine similarities. Source [36]

WORD EMBEDDINGS	DATA SET		
	SimLex999	SimVerb3500	WordSim353
Ground Truth Similarity	4.56157	4.29155	5.85586
FastText embeddings(300 dimensions, pre-trained on Crawl dataset)	3.72571	3.40074	3.54365
FastText embeddings(300 dimensions, pre-trained on Wikipedia dataset)	4.81236	4.12076	4.68745
GloVe embeddings(200 dimensions, pre-trained on Twitter dataset)	4.62278	3.78956	3.91849
GloVe embeddings(50 dimensions, pre-trained on Twitter dataset)	6.08757	5.21661	5.37284
GloVe embeddings (300 dimensions, pre-trained on Wikipedia dataset)	3.91932	2.98322	3.3062
GloVe embeddings (50 dimensions, pre-trained on Wikipedia dataset)	5.86556	4.80882	5.18904
LexVec embeddings (300 dimensions, pre-trained on Crawl dataset)	3.65168	3.10501	2.73887
LexVec embeddings (300 dimensions, pre-trained on Wikipedia dataset)	3.65592	3.00337	2.72824
ConceptNet Number batch embeddings(300 dimensions)	3.88586	3.12	2.69218
word2vec embeddings (300 dimensions,pre-trained on Google News dataset)	3.7804	3.01903	2.7518

Table 4 Correlation coefficients between ground truth similarities and word vector cosine similarities for the WordSim353 dataset, SimLex999 dataset and SimVerb3500 dataset. Source [36]

Word Embeddings	WordSim353 dataset			SimLex999 dataset			SimVerb3500 dataset		
	S	Pr	K	S	Pr	K	S	Pr	K
FastText-Crawl-300	0.25	0.26	0.17	0.16	0.16	0.11	0.11	0.11	0.07
FastText-Wikipedia-300	0.19	0.19	0.13	0.09	0.07	0.06	0.03	0.02	0.02
GloVe-Twitter-200	0.52	0.53	0.36	0.13	0.14	0.08	0.06	0.07	0.04
GloVe-Twitter-50	0.46	0.46	0.32	0.10	0.10	0.06	0.03	0.04	0.02
GloVe-Wikipedia-300	0.61	0.60	0.45	0.37	0.39	0.30	0.23	0.23	0.16
GloVe-Wikipedia-50	0.50	0.51	0.36	0.26	0.29	0.18	0.15	0.16	0.10
LexVec-Crawl-300	0.72	0.68	0.53	0.44	0.45	0.31	0.30	0.31	0.21
LexVec-Wikipedia-300	0.66	0.63	0.48	0.38	0.39	0.27	0.28	0.28	0.19
ConceptNet-Numberbatch-300	0.81	0.75	0.63	0.63	0.65	0.46	0.57	0.59	0.41
word2vec-GoogleNews-300	0.69	0.65	0.51	0.44	0.45	0.31	0.36	0.38	0.25

Table 5 Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [38]

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [38]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Table 6 Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used. Source [12]

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collebert-Weston NNLM	50	660 M	9.3	12.3	11.0
Turian NNLM	50	37 M	1.4	2.6	2.1
Turian NNLM	200	37 M	1.4	2.2	1.8
Mnih NNLM	50	37 M	1.8	9.1	5.8
Mnih NNLM	100	37 M	3.3	13.2	8.8
Mikolov RNNLM	80	320 M	4.9	18.4	12.7
Mikolov RNNLM	640	320 M	8.6	36.5	24.6
Huang NNLM	50	990 M	13.3	11.6	12.3
MikolovNNLM	20	6B	12.9	26.4	20.3
MikolovNNLM	50	6B	27.9	55.8	43.2
MikolovNNLM	100	6B	34.2	64.5	50.8
CBOW	300	783 M	15.5	53.1	36.1
Skip-gram	300	783 M	50.0	55.9	53.3

Table 7 Comparison of models trained for three epochs on the same data and models. Trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set. Source [12]

Model	Vector Dimensionality	Training words	Accuracy [%]			Training Time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783 M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783 M	50.0	55.9	53.3	3
1 epoch CBOW	300	783 M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783 M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783 M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783 M	56.7	54.5	55.5	2.5

positioned in such a way that words sharing common context are in close proximity to one another. Some applications of word2vec are listed out here.

Survey Response Analysis: When machine learning algorithms are trained on different survey data using word2vec, it helps establish complex relationship among different words thereby helping in taking necessary action while formulating responses.

Recommendation Systems: A lot of streaming services are available now for users online. Implementing word2vec for recommending what content the user has to see next is very advantageous if these streaming services can use the algorithm. As word2vec establishes context oriented relationship between words in sentences, the words in the previous song can be helpful to recommend a song which has the same context. In this manner, movies, songs and many other entertainment services can be tailored according to user requirements which further can improve the rating of the streaming service.

Knowledge Discovery: Hidden knowledge in documents can be discovered by using word2vec. Researchers in [39] has trained word2vec model about three millions of documents and found that there is a relationship among chemical compounds.

6.2 Limitations

- Handling Out-Of-Vocabulary (OOV) words is not possible with word2vec. Random vectors are assigned to OOV words which are suboptimal.
- Semantic representation of word vector representation is only limited to local information of the language words.
- Learning new languages using word2vec is a tedious task as these parameters cannot be shared with other languages and training should be started from scratch.

7 Conclusion and future research

This paper gives an overview of existing word embedding techniques. A detailed review on word2vec is presented for better understanding. It is evident that, generation of high quality word vectors is very much essential for NLP applications. The detailed analysis shows that word2vec can be applied to different applications for better processing of text and word

vector generation. This paper acts as a base for understanding the advanced techniques of word embedding. The paper also gives information regarding merits and demerits of different word embedding techniques, applications and limitations of word2vec algorithm, along with comparative analysis with different datasets. More advanced word embedding algorithms like BERT (Bidirectional Encoder Representations from Transformers) has evolved in the recent years and has been a big revolution in the field of Natural Language Processing. BERT uses bidirectional training and attention mechanism which made it an advanced language processing technique. As part of future research analysis, the authors would like to give a detailed analysis on BERT model as currently most of the researchers favour BERT for text processing tasks.

Declarations

Conflict of interest The authors certify that there is no conflict of interest in the subject matter discussed in this manuscript.

References

1. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
2. Mcdonald, Scott. (2008). Testing the distributional hypothesis: The influence of context on judgements of semantic similarity.
3. Hillebrand L, Biesner D, Bauckhage C, Sifa R (2021) Interpretable Topic Extraction and Word Embedding Learning Using Non-Negative Tensor DEDICOM. Mach Learn Knowl Extraction 3(1):123–167. https://doi.org/10.3390/make3010007
4. Levy O, Goldberg Y (2014) Neural word embedding as implicit matrix factorization. In: Proceedings of the 27th international conference on neural information processing systems, vol 2, no. December, pp 2177–2185. https://doi.org/10.5555/2969033.2969070
5. Pennington, J.; Socher, R.; Manning, C. Glove: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 25–29 October 2014. 1532–1543.
6. Deerwester S, Dumias ST, Furmas GW, Lander TK, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407 (https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/(SICI)1097-4571(199009)41:63.0.CO;2-9)
7. Hofmann T (1999) Probabilistic latent semantic analysis. In:Thomas H (ed) Probabilistic latent semantic analysis. Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publisher Inc. 289–296
8. https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05
9. Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022
10. Lebret R, Collobert R (2014) Word embeddings through Hellinger PCA. In: 14th Conference of the european chapter of the association for computational linguistics 2014, EACL 2014, 482–490
11. Pennington J, Socher R, Manning C (2014) Glove: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1532–1543
12. Efficient Estimation of Word Representations in Vector Space Tomas Mikolov Google Inc., Mountain View, CA tmikolov@google.com Kai Chen Google Inc., Mountain View, CA kaichen@google.com Greg Corrado Google Inc., Mountain View, CA gcorrado@google.com Jeffrey Dean Google Inc., Mountain View, CA jeff@google.com arXiv:1301.3781v3 [cs.CL] 7 Sep 2013
13. arXiv:1411.2738v4 [cs.CL] 5 Jun 2016word2vec Parameter Learning Explained Xin Rong ronxin@umich.edu
14. Huang EH, Socher R, Manning CD, Ng AY (2012) Improve Word Representation via Global Context and Multiple Word prototypes. In: Proceedings of the 50th annual meeting of the association for computational linguistics no. July.JejuIslan: Korea: Association for Computational Linguistics, 873–882
15. Sen P, Ganguly D, Jones G (2019) Word-Node2Vec: improving word embedding with document-level non-local word cooccurrences. In: 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), pp 1041–1051. [Online]. Available: https://www.aclweb.org/anthology/N19–1109
16. Reisinger J, Mooney RJ (2010) Multi-prototype vector-space models of word meaning. In: NAACL HLT 2010 - human language technologies: the 2010 annual conference of the north american chapter of the association for computational linguistics, proceedings of the main conference, no June, 109–117
17. Wu Z, Giles CL (2015) Sense-aware semantic analysis: A multi-prototype word representation model using wikipedia. Proc Nat Conf Artif Intell 3:2188–2194
18. Neelakantan A, Shankar J, Passos A, McCallum A (2014) Efficient non-parametric estimation of multiple embeddings perword in vector space. In: EMNLP 2014 - 2014 Conference on empirical methods in natural language processing, proceedings of the conference, 1059–1069
19. Li J, Jurafsky D (2015) Do multi-sense embeddings improve natural language understanding? In: Conference proceedings - EMNLP 2015: conference on empirical methods in natural language processing no. September 1722–1732
20. Chen X, Liu Z, Sun M (2014) A unified model for word sense representation and disambiguation. In: EMNLP 2014 - 2014 Conference on empirical methods in natural language processing, proceedings of the conference, 1025–1035
21. Tian F, Dai H, Bian J, Gao B, Zhang R, Chen E, Liu TY (2014) A probabilistic model for learning multi-prototypeword embeddings. In: COLING 2014 - 25th International conference on computational linguistics, proceedings of COLING 2014: technical papers, 151–160
22. Peters M, Neumann M, Iyyer M, Gardner M, Clark C, Lee K,Zettlemoyer L (2018) Deep contextualized word representations.In: Proceedings of the 2018 conference of the north American chapter of the association for computational linguistics: humanlanguage, Vol. 1 (Long Papers) Technologies. New Orleans,Louisiana: Association for Computational Linguistics, pp 2227–2237. [Online]. Available: https://www.aclweb.org/anthology/N18-1202
23. Devlin J, Chang M-W, Lee K, Toutanova K (2019) BERT: pretraining of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: human language technologies. Minneapolis, Minnesota: Association for Computational Linguistics, pp 4171–4186. [Online]. Available: https://www.aclweb.org/anthology/N19-1423
24. Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov R, Le QV (2019) XLNet: generalized autoregressive pretraining for language understanding. In: Advances in neural information processing systems (NIPS 2019), pre-proceedings. Curran Associates, pp 5754–5764.[Online]. Available: http://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive -pretr aining-for-language-understanding.pdf
25. Maas AL, Daly RE, Pham PT, Huang D, Ng AY, Potts C (2011) Learning word vectors for sentiment analysis. In: Proceedings of the 49th annual meeting of the association for computational linguistics (ACL-2011), no. February, pp 142–150. [Online]. Available: https://www.aclweb.org/anthology/P11-1015
26. Mikolov T, Chen K, Corrado G, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: In Advances in neural information processing systems, 3111–3119
27. Zhang Z, Lan M (2016) Learning sentiment-inherent word embedding for word-level and sentence-level sentiment analysis. In: Proceedings of 2015 international conference on asian language processing, IALP 2015, 1: 94–97
28. Yu L-C, Wang J, Lai KR, Zhang X (2017) Refining word embeddings for sentiment analysis. In: Proceedings of the 2017 conference on empirical methods in natural language processing Copenhagen, pp 534–539. [Online]. Available: https://www.aclweb.org/anthology/D17–1056
29. Rezaeinia SM, Rahmani R, Ghodsi A, Veisi H (2019) Sentiment analysis based on improved pretrained word embeddings. Expert SystAppl 117:139–147. https://doi.org/10.1016/j.eswa.2018.08.044
30. Rahimi Z, Homayounpour MM (2021) TensSent: a tensor based sentimental word embedding method. ApplIntell. https://doi.org/10.1007/s10489-020-02163-8
31. Using Latent Semantic Indexing to Filter Spam Kevin R. Gee Dept. of Computer Science and Engineering The University of Texas at Arlington Arlington, TX 76019 geek@cse.uta.edu
32. K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in Proceedings of the 26th Annual International Conference on Machine Learning. ACM, 2009, 1113–1120.
33. G. Salton and M. McGill, editors.Introduction to Modern Information Retrieval.McGraw-Hill, 1983.
34. https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/
35. Golub G, Reinsch C (1970) Singular value decomposition and least squares solutions. Numerische-Mathematik 14(5):403–420
36. Comparative analysis of word embeddings for capturing word similarities , Martina Toshevska, Frosina Stojanovska and Jovan Kalajdjieski Faculty of Computer Science and Engineering, Ss. Cy.

37. Liu Z, Li M, Liu Y, Ponraj M (2011) Performance evaluation of Latent Dirichlet Allocation in text mining. Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD) 2011:2695–2698. https://doi.org/10.1109/FSKD.2011.6020066
38. T. Mikolov, W.T. Yih, G. Zweig. Linguistic Regularities in Continuous Space Word Representations. NAACL HLT 2013.
39. Tshitoyan V, Dagdelen J, Weston L et al (2019) Unsupervised word embeddings capture latent knowledge from materials science literature. Nature 571:95–98. https://doi.org/10.1038/s41586-019-1335-8