The Evolution of Word Vector Representation
CS 410 Technology Review (Fall 2021)

Xiaoyu Zhu

Introduction

Word vector representation is one of the basic building blocks of modern nature language processing tasks. Word vectors are simply a matrix of numbers that represent the meaning of a word. It is needed as computers don't understand words or sentences like humans do. Words have to be transferred into numbers so that computers are able to understand. This technology review will discuss the intuition of different word vector representation methods, from the classical bag of word approach to modern advanced approaches, and their power of expression.

Bag of word representation

Uni-gram

A uni-gram bag-of-words model is similar to the idea of one-hot-encoding. The model counts how many times a word appears in a document, and turns the term frequency of a collection of words into fixed length vectors, disregarding grammar and even word order. The length of vectors is determined by the total number of unique words in a corpus. Let’s see an example of bag-of-words models. Assume there are two sentences in a document, each word is represented with a large vector of size |W|.

S_1 = “I like movie and music”

S_2 = “I like pizza “

After tokenization process, the word vector is shown below:

W = [I, and, like, movie, music, pizza]

Then, sentences can be represented as following:

	I	**and**	**like**	**movie**	**music**	**pizza**
S_1	1	0	1	1	1	0
S_2	1	0	1	0	0	1

N-gram

The Uni-gram model is an unordered document representation and it failed to capture english phrases such as “Cutting corners”, “Call it a day”. A more sophisticated approach N-gram model is introduced in order to create a vocabulary of grouped words. This changes the scope of the vocabulary and allows the bag-of-words to capture more meaningful information from the document. For example, when N equals to 2, it becomes the bi-gram language model, a vocabulary of two-word pairs is used to represent a sentence. In bi-gram representation, S_1 would become:

$$W_{S1} = \text{[I like, like movie, movie and, and music]}$$

Value of N could be further increased to track triplets of words, or four words collection and more. In most of the cases, however, bi-gram model is powerful enough, and proves very hard to beat.

The bag-of-words model is a very simple yet powerful approach that does well in NLP tasks such as document classification and sentiment analysis. Nevertheless, it suffers from some limitations. First, consider deploying the Bag of words approach to generate word vectors for large documents. The vectors will be of large dimension and will always contain far too many null values resulting in sparse vectors because there are many rare words that only appear in a few documents. This is a waste of space and increases algorithm complexity exponentially resulting in the curse of dimensionality. Second, because the order of words is ignored, the Bag of words approach failed to capture the contextual meaning of the words. There is no correlation between words that have similar meaning or usage. For example, “I like movie but hate video game” and “I like video game but hate movie” would have similar word vector representation, but the meaning of the two sentences are very different.

Word2Vec

A Word2Vec model is a neural network with a single hidden layer that is designed to group vectors of similar words together based on input vectors. Given a large enough dataset, Word2Vec can make strong estimates about a word’s meaning based on its surrounding context word. These estimates yield word associations with other words in the corpus. For example, words like “King” and “Queen” would be very similar to one another.

Co-occurrence matrix is a matrix which captures the number of times a term appears in the context of another term.Consider the previous example document containing sentences S_1 and S_2, the co-occurrence matrix would be:

	I	and	like	movie	music	pizza
I	0	1	2	1	1	1

and	1	0	1	1	1	0
like	2	1	0	1	1	1
movie	1	1	1	0	1	0
music	1	1	1	1	0	0
pizza	1	0	1	0	0	0

With this method, a word representation is not just a simple one-hot encoding anymore, we are able to get some ideas about the overall context. Word2Vec extends the idea of co-occurrence matrix. A window size of N is applied to limit the co-occurrence only to a word’s surrounding word. The model has two methods to generate word vectors, continuous Bag of Word model and Skip-gram Model, both models utilize a single hidden layer neural network to produce word embedding.

Continuous Bag Of Words Model (CBOW)

The CBOW model learns the word vector representation by predicting the center word based on its context. The model predicts the target word by taking its surrounding words as inputs. Consider the same sentence S_1, ‘I like movie and music’.The model converts this sentence into word pairs in the form of (Context word, Target word). The number of surrounding words or window size is a hyper-parameter set by the user. If the window for the context word is 2 then the word pairs would look like this: ([I, movie], like), ([movie, music], and). The complete training sample would be:

Independent Variable: [[I, movie], [like, and], [movie, music], [I, pizza]]
Response Variable: [like, movie, and, like]

With these word pairs, the context words are feeded into a neural network with a single hidden layer to predict the target word, illustrated in the figure below.

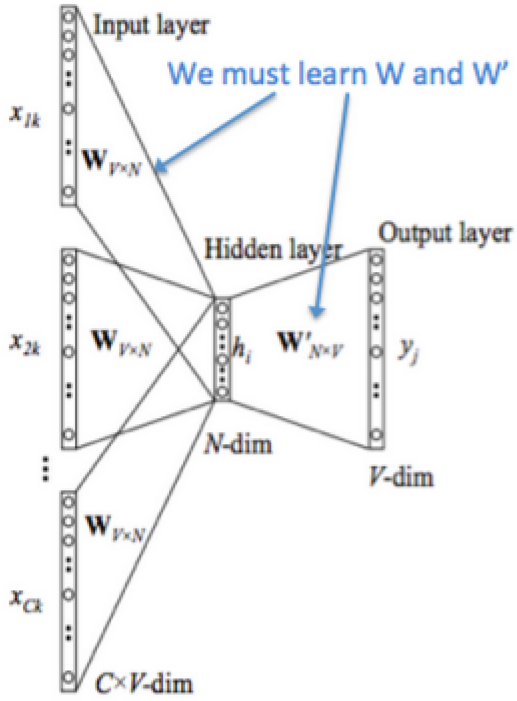


*Figure 1. CBOW illustration[1]

Since CBOW is used for word vector representation, getting the inputs and outputs of this network isn't the main purpose, rather the goal is to learn the weights of the hidden layer that are actually “word vectors” the CBOW model is trying to learn. This is also called embedding layer.

Skip-gram Model

The Skip-gram is somewhat similar to CBOW. The model still takes a pair of words and teaches the model their co-occurrence, but instead of predicting the center word based on its surrounding word, the skip-gram model takes the center word as input of the neural network, and predicts its surrounding word, illustrated in the figure below.

[1] Figure 1. reference: Chaubard, Francois. "Word Vectors I: Introduction, SVD and Word2Vec" CS224n: Natural Language Processing with Deep Learning, Winter 2019, Stanford University.

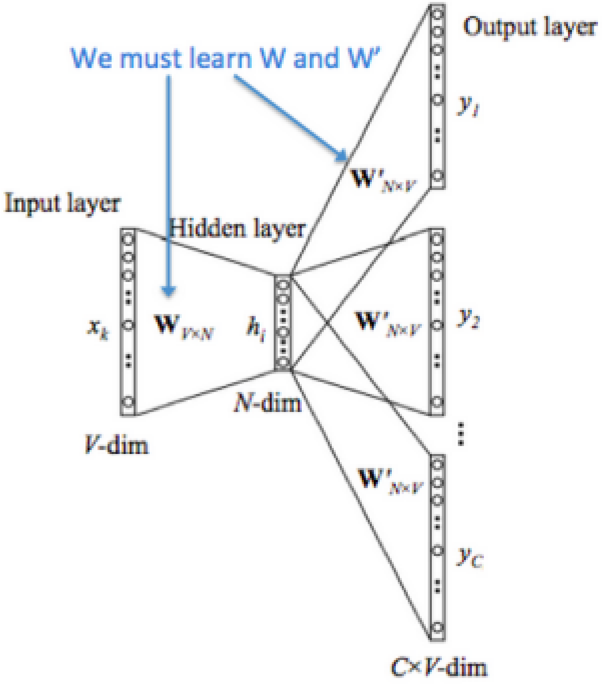


*Figure 2. Skip-gram illustration[2]

Again, same as CBOW, learned parameters of the hidden layer is the final "word embedding" output. The calculation follows the following steps:

1. Both input word and the output word are one-hot encoded into vectors x and y of size $|V|$, which represents the total number of vocabulary in a corpus.
2. Train the neural network and learn the parameters of the hidden layer.
3. Multiply the vector x by the word embedding matrix W of size V×N. This returns the embedding vector. (N is the number of hidden neurons in the hidden layer, and it is a hyper-parameter set by users.)
4. Multiply the embedding vector from the previous step by the context word matrix W' of size N×V. This produces a score vector that needs to be normalized into probability, which is the the final predicted vector $\hat{y}$.

If two different words $x_i$and x_j have very similar "contexts" (that is, what words are likely to appear around them), this means the output vector $\hat{y}$ for both words are close to each other. In this case, the learned embedding vectors (Multiplication of word vector x and word embedding matrix W) produced in step 3 must be similar for both words. This explains the reason why the skip-gram model does a good job on word analogy tasks.

[2] Figure 2. reference: Chaubard, Francois. "Word Vectors I: Introduction, SVD and Word2Vec" CS224n: Natural Language Processing with Deep Learning, Winter 2019, Stanford University.

In conclusion, predict based methods such as CBOW and Skip-gram learn word representations using local context information. These models demonstrate the capacity to capture complex linguistic patterns beyond word similarity through the use of neural networks to discover the hidden word pattern, but fail to make use of the global co-occurrence statistics. In other words, the semantics learnt is only affected by the surrounding words.

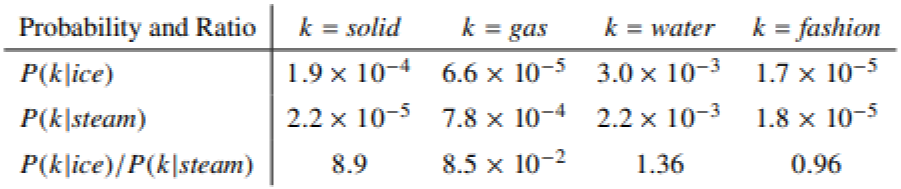
Global Vectors (Glove)

The Global Vector (GloVe) model proposed by Pennington et al. (2014) aims to combine the count-based matrix factorization and the context-based skip-gram model together. GloVe consists of a weighted least squares model that trains on global word co-occurrence counts to capture semantic similarity between words.

Let define:

$$p(w_k|w_i) = \frac{C(w_i, w_k)}{C(w_i)}$$

$p(w_k|w_i)$ is defined as the conditional probabilities of word k appearing, given word i is nearby. As Figure 3 illustrated below. $w_i = "ice"$and $w_j = "steam"$. When the $w_k = "solid"$, the conditional probability of $p(solid|ice)$ is higher compare to $p(solid|steam)$ as solid is more related to “ice”, therefore the ratio of $p(solid|ice)$ over $p(solid|steam)$ is large. The observation concludes that the absolute value of the ratio is higher when the w_k is related to either w_i or w_j, and the ratio is close to 1 if w_k is related to both words or not related at all.



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

*Figure 3. Co-occurrence probabilities table (adapted from Table 1 in [Pennington et al., 2014])

In summary, the table indicates that the ratio of conditional probabilities represents the word meanings. The model aims to learn the function “F” of w_i, $w_j$and w_k to encode the ratio of conditional probabilities.

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

Conclusion

Natural language processing (NLP) is a fast evolving branch in the field of artificial intelligence. Although some new word embedding methods such as ELMo and BERT were introduced and have proven stronger performance, the methods discussed in this technology review still remain to be the most fundamental and influential word embedding approach. It’s hard to tell what the best word embedding method is, it is determined by many factors such as the language users are trying to model, the topic of documents, the hyper-parameters selected and the consideration of computational speed. Overall, it is generally good to select a word embedding method based on what the task is. If the goal is to do sentiment analysis on a simple tweet, even a simple bi-gram bag of word model could have strong performance.