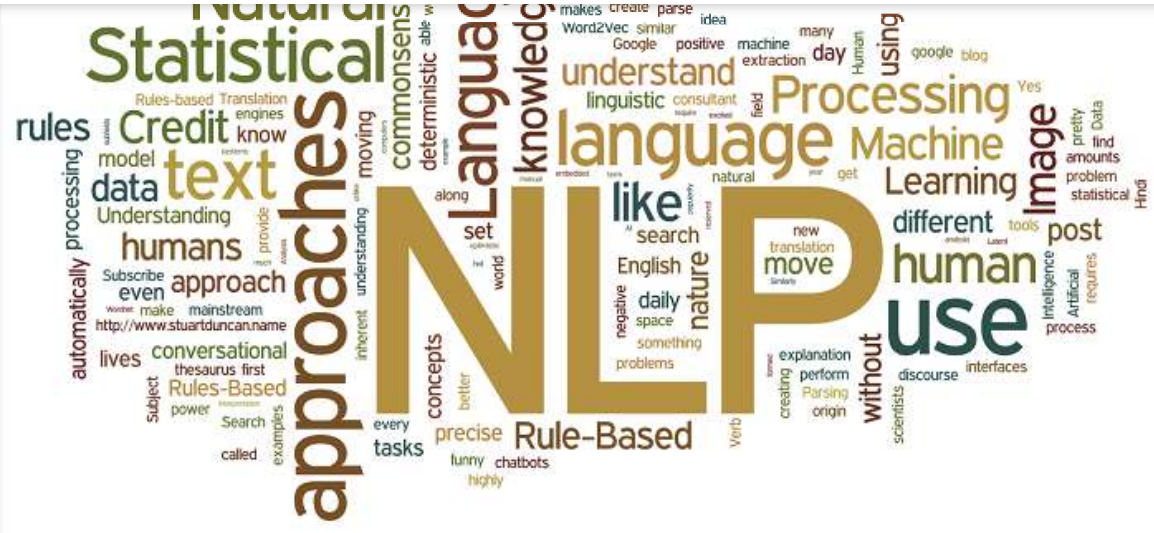
Chanika Ruchini

Summary

The provided text introduces the concept of word embeddings in natural

Introduction to Word Embeddings



Word embedding is one of the most powerful concepts of deep learning applied to Natural Language Processing. It is capable of capturing the context of a word in a document, semantic and syntactic similarity, relation with other words, etc. This article is gonna be ideal for those who are new to the concept of word embedding and can get a basic understanding of the need and usage of word embedding.

What is a word embedding?

> **Word embedding** is the collective name for a set of language modeling and feature learning techniques in language modeling where words or phrases from the vocabulary are mapped to vectors of real numbers.

representations of text in an n-dimensional space where words that have the same meaning have a similar representation. That means two similar words are placed very closely in vector space almost having similar vector representations. So, When constructing a word embedding space the goal is to capture some sort of relationship in that space, be it meaning, morphology, context, or some other kind of relationship.

Few main characteristics of word embedding are listed below:

- Every word has a unique word embedding (or “vector”), which is just a list of numbers for each word.
- The word embeddings are multidimensional; typically for a good model, embeddings are between 50 and 500 in length.
- For each word, the embedding captures the “meaning” of the word.
- Similar words end up with similar embedding values.

Why do we use word embeddings?

Humans have always excelled at understanding languages. It is easy for humans to understand the relationship between words but for computers, this task may not be simple. For example, we humans understand the words like king and queen, man and woman, tiger and tigress have a certain type of relation between them but how can a computer figure this out? This is where word embedding comes to play in natural language processing.

their raw form. They require numbers as inputs to perform any sort of job, be it classification, regression, etc.

There is no easy way to make a useful comparison between the two terms unless we already know what they say. The goal of word-embedding algorithms is therefore to embed words with meaning based on their similarity or relationship to other words.

One-Hot -Encoding can be identified as one of the simplest of representing words numerically. Here Create a vector that has as many dimensions as your corpora have unique words. Each unique word has a unique dimension and will be represented by a 1 in that dimension with 0s everywhere else. To get a better understanding, suppose my copus has two sentences:

* The man and women live happily.

* The king and Queen live happily.

You will note that these two sentences have the same contextual meaning and the vocabulary has 8 unique words as { “The”, “ man”, “king”, “and”, “women”, “queen”, “live”, “happily”}. When we apply one-hot encoding to the sentences,

```
man    - [0,1,0,0,0,0,0,0]
king   - [0,0,1,0,0,0,0,0]
and    - [0,0,0,1,0,0,0,0]
women  - [0,0,0,0,1,0,0,0]
queen  - [0,0,0,0,0,1,0,0]
live   - [0,0,0,0,0,0,1,0]
happily- [0,0,0,0,0,0,0,1]
```

Although this a simple implementation approach, there are some major disadvantages. As you might have noticed already, we're only setting one element using the word index of the entire vector. As the vocabulary size increases, we'd end up using an extensive length sparse vector for encoding a single word which would result in performance and storage penalties because of the curse of dimensionality. In addition to that, such representation is incapable of learning semantic relationships between words which is of essential importance when dealing with textual data.

Word embeddings eliminate all the above shortcomings and equip us with enriched powerful representations that are capable of capturing contextual and semantic similarities between words.

In this article, we will, however, focus on the main two word embedding methods which are more advanced in nature.

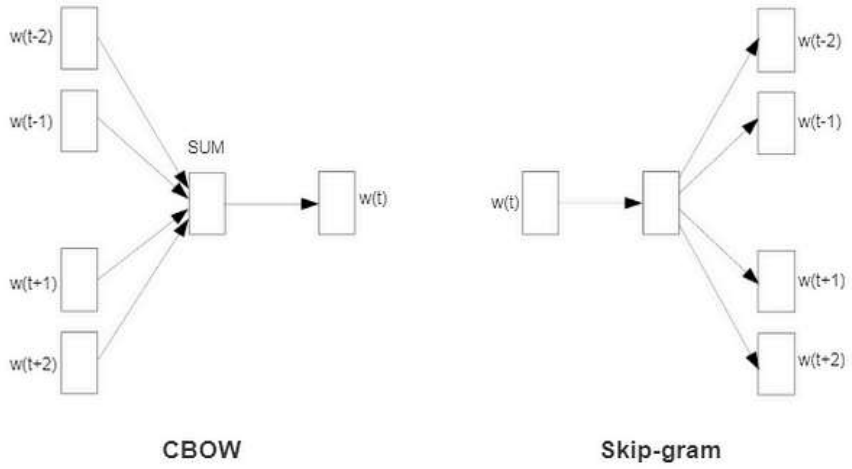
- Word2vec

Word2vec

Word2Vec is one of the most popular predication based algorithm, for generating word embeddings, which was originally proposed at Google by Mikolov et al. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. This takes corpus of texts as input and produces a vector for each word that is represented in the corpus.

Two different learning models were introduced that can be used as part of the word2vec approach to learning the word embedding as:

- Continuous Bag-of-Words, or CBOW model.
- Continuous Skip-Gram Model.



source: https://arxiv.org/pdf/1301.3781.pdf

Continuous Bag-of-Words, or CBOW model: This technique uses the shallow 2-layer neural network to predict the probability of a word given the context. A context can be a single word or a group of words.

Continuous Skip-Gram Model: The Skip-gram model is similar to the CBOW model, but instead of predicting the current word given the context, it tries to predict the context words from the current word.

Both models are focused on learning about words given their local usage context, where the context is defined by a window of neighboring words. This window is a configurable parameter of the model. *The main advantage of the word2vec approach is the ability to learn high-quality word embeddings*

embeddings from large text corpora.

Glove

The Global Vectors for Word Representation, or GloVe, the algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford in 2014. This is an unsupervised learning algorithm for word representation. It learns vectors or words on basis of their frequency of occurring together. GloVe's contribution was the addition of global statistics in the language modeling task to generate the embedding. There is no window feature for the local context. Instead, there is a word-context/word co-occurrence matrix that learns statistics across the entire corpora.

It first constructs a matrix X where the rows are words and the columns are contexts with the element value X_ij equal to the number of times a word i appear in a context of word j. By minimizing reconstruction loss, this matrix is factorized to a lower-dimensional representation, where each row represents the vector of a given word. Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that may result in a generally better word embedding.

Summary

and described the two most common word embedding approaches. Word Embeddings have played a huge role across the complete spectrum of NLP applications. So this provides you a basic understanding before diving into the deeper implementations.

I hope you found this article useful.

References

[1]https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

[2]. https://arxiv.org/pdf/1411.2738.pdf

[3] “Get Busy with Word Embeddings — An Introduction”, *Shane Lynn*

Word Embeddings NLP Word2vec Glovo

LangChain.

11 min read


Gaurav tailor

Google’s trained Word2Vec model in Python

In this post I will describe how to get Google’s pre-trained Word2Vec model up and running in Python to play with.

4 min read


Javier Castaño

Embedding Vectors in NLP

Word and Sentence Embeddings

19 min read


Jo Wang

Deep Learning Part 5 -How to prevent overfitting

Techniques used to prevent overfitting in deep learning models:

4 min read


Ajay Halthor

10 min read

Amit Yadav

Langchain vs Huggingface

If you think you need to spend $2,000 on a 120-day program to become a data scientist, then listen to me for a minute.

10 min read