**Step 4: Mastering Text Preprocessing — 3**

[[Gourav Didwania](https://medium.com/@gourav.didwania?source=post_page-----9f1b560a0f05--------------------------------)](https://medium.com/@gourav.didwania?source=post_page-----9f1b560a0f05--------------------------------)

[[GoPenAI](https://blog.gopenai.com/?source=post_page-----9f1b560a0f05--------------------------------)](https://blog.gopenai.com/?source=post_page-----9f1b560a0f05--------------------------------)

[Gourav Didwania](https://medium.com/@gourav.didwania?source=post_page-----9f1b560a0f05--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F87956575bc01&operation=register&redirect=https%3A%2F%2Fblog.gopenai.com%2Fstep-4-mastering-text-preprocessing-3-9f1b560a0f05&user=Gourav+Didwania&userId=87956575bc01&source=post_page-87956575bc01----9f1b560a0f05---------------------post_header-----------)

Published in

[GoPenAI](https://blog.gopenai.com/?source=post_page-----9f1b560a0f05--------------------------------)

·

7 min read

·

Sep 5

60

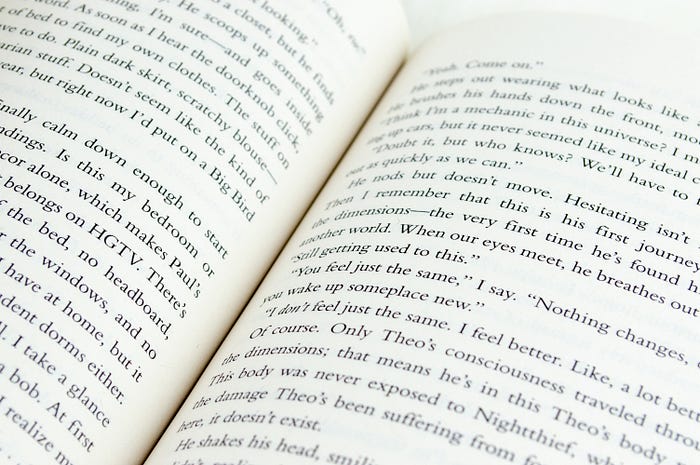


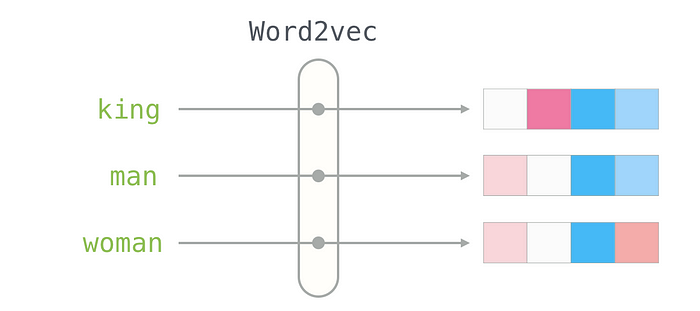
Photo by [Mel Poole](https://unsplash.com/@melpoole?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

In this phase, we explore more advanced techniques for transforming our textual data into numerical vectors, making it suitable for algorithms. These techniques are categorized mostly as Distributed Representations

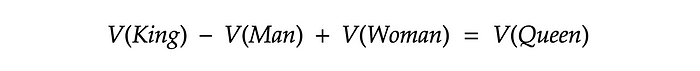
**Distributed Representations**

Distributed representations are the opposite of localist representations. They are dense vectors, which are typically much smaller than the vocabulary size and are designed in such a way that they capture similarity between related words. Word2vec and GloVe are distributed representations for large vocabulary sizes.

1.**Word2Vec:**Word2Vec is a popular word embedding technique that learns distributed representations of words in a continuous vector space. It not only preserves the relationship between words. but deals with the addition of new words to the vocabulary. It shows better results in lots of deep learning applications.



A very popular example of word2vec is this, where V(word) represents the Vector representation of the word.

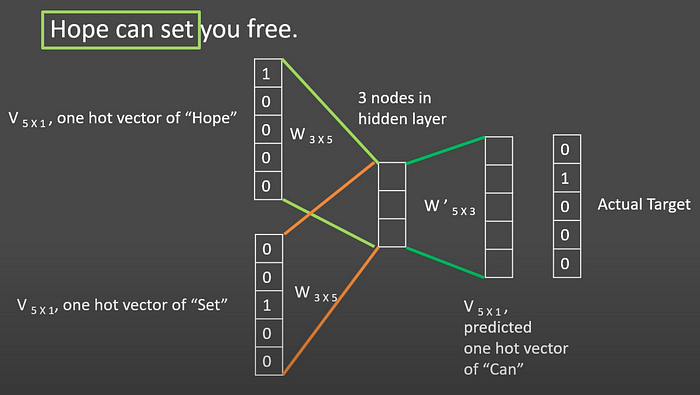


Word2Vec has two primary architectures. ***Continuous Bag of Words (CBoW)*** and ***Skip Gram***. These architectures are used to create word embeddings by training neural networks on large text corpora. Here’s an overview of both CBoW and Skip Gram:

**Continuous Bag of Words (CBoW):**In the CBoW architecture, the model’s objective is to predict a target word based on the context words (words that surround it within a specified *window*) in a sentence. It tries to learn the probability distribution of the target word given its context.

* The training data consists of pairs of *target* words and their *context* words.
* The context words are used as input to the neural network.
* The network’s hidden layer learns to predict the target word.
* The weights of the hidden layer neurons represent the word embeddings.
* **Advantages**: CBoW is faster to train than Skip Gram because it predicts a single target word from multiple context words. Also, It tends to work well for frequent words.

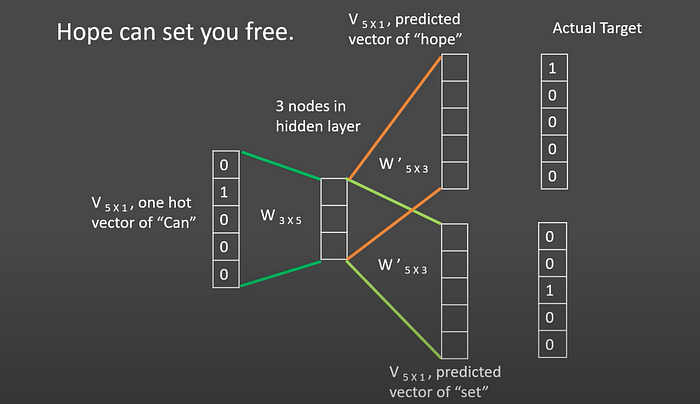
Let’s assume we have a sentence “*Hope can set you free*” with a window size of 3. We first have every pair of 3 adjacent words. The first pair will be “*Hope can set.*” Here, we’ll use One Hot Encoded vectors of Hope and pass it to a hidden layer of size equal to window size and try to predict the center word ‘*can.*’



**Skip Gram:**In the Skip Gram architecture, the model’s objective is the reverse of CBoW. It aims to predict context words (surrounding words) based on a target word.

* The training data consists of pairs of a target word and its context words.
* The target word is used as input to the neural network.
* The network learns to predict the context words.
* The weights of the hidden layer neurons represent the word embeddings.
* **Advantages:**Skip Gram is more effective for infrequent words and for capturing rare or nuanced semantic relationships. Also, It tends to produce better embeddings for a wide range of words.

In the above example. Again we first have every pair of 3 adjacent words. The first pair will be “*Hope can set.*” Here, we’ll use One Hot Encoded vector of “*Can*” and pass it to a hidden layer of size equal to window size and try to predict the other words in the set that is “*Hope*” and “*set.*”



In real-life scenarios, we generally train it over Wikipedia text using window size 5–10 and vector size is around 300.

**So, which one CBoW or Skip Gram is better?**

There’s no definitive “better” between CBoW and Skip Gram. Choose Skip Gram for small datasets and rare words, and CBoW for large datasets with a focus on frequent words. CBoW also trains faster. It depends on your specific data and goals

2.**Global Vectors (GloVe):** Word2Vec relies only on nearby words to create word embeddings, which can limit context. To address this limitation, GloVe comes into play. It is a word-level vector representation method that leverages the entire corpus to generate embeddings for words.

GloVe starts by constructing a word co-occurrence matrix. This matrix represents how often words co-occur in a given context within the corpus. The context can be defined based on the proximity of words within a certain window in a sentence or document.

***A word-word co-occurrence matrix****is essentially a 2D array that records the frequency of every potential word pair within a corpus. Think of it like a correlation matrix, where instead of correlation values, we have counts of how often****n****words appear together within a defined context window.*

Each row and column in the matrix correspond to a unique word in the vocabulary, and the entries in the matrix capture the frequency of co-occurrence between words. The core idea behind GloVe is that the ratio of the co-occurrence probabilities of two words should encode semantic information. Words that have a strong semantic relationship should have a high ratio, while unrelated words should have a low ratio.

GloVe defines an objective function that aims to learn word vectors and uses an optimization algorithm, typically **stochastic gradient descent (SGD)**, to minimize the loss function.

We have already trained models on Wikipedia text which you can find [here](https://nlp.stanford.edu/projects/glove/).

Here’s an example to illustrate this concept:



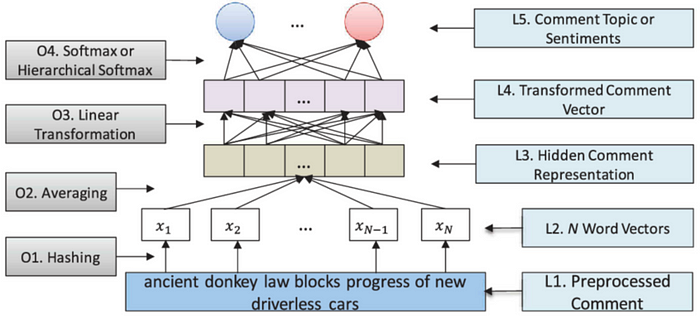
The notebook can be found on this [link](https://colab.research.google.com/drive/1OArAFoZxJ5KpHgiF5N2x9Mlw7lF8OQR3?usp=sharing).

3.**fastText:** FastText is an open-source, free, and lightweight library developed by **Facebook’s** AI Research lab for text representation and classification. FastText can be considered as an extension to the skip-gram word2vec model, with some subtle modifications. Instead of treating words as indivisible units, FastText breaks words down into smaller components by dividing them into character n-grams. Special characters are added at the beginning and end of each word. For example, a word like “example” is transformed into ‘**<example>**’ and then further split into character n-grams like **‘<e’, ‘ex’, ’am’, ‘mp’, ‘pl’, ‘le, ‘e>’** and so on.

Focusing on character-level n-grams enables the model to capture sub-word embeddings effectively. Consequently, *FastText can better understand variations of the same word*, such as **‘drives,’ ‘driving,’ ‘drove,’** etc. Additionally, it offers potential solutions for ***handling out-of-vocabulary*** ***(OOV)*** words by providing subword embeddings, even when the entire word may not be in the training data.

FastText uses a hierarchical softmax to speed up training and make it more efficient, especially for large vocabularies. Beyond word embeddings, fastText includes a **text classification** module. It can be used for various NLP tasks, including *sentiment analysis, topic classification,*and*spam detection.*

We have already trained models on Wikipedia text which you can find [here](https://fasttext.cc/docs/en/english-vectors.html).



Architecture of **fastText**

It takes text input, typically represented using pre-trained word embeddings, and employs average pooling to create a fixed-length vector representing the entire text. This vector is then passed through one or more hidden layers with non-linear activations, followed by an output layer using softmax to predict class probabilities. The model is trained to minimize cross-entropy loss through backpropagation, making it well-suited for multi-class classification tasks.

**Limitations of Word Embedding**

**Loss of Syntactic and Semantic Understanding**

Traditional word embeddings do not consider the order of words in a sentence, leading to a loss of syntactic and semantic understanding.

* For example: “**The cat chased the dog.**” and “**The dog chased the cat.**” would have similar representations even though they have opposite meanings.

**Contextual Ambiguity**

Word embeddings struggle with contextual ambiguity, where the meaning of a word varies depending on the context.

* For instance: “**He took a shot at the target.**” (referring to a rifle shot) and “**She took a shot of espresso.**” (referring to a drink). The word “shot” has different meanings in these contexts.

**Polysemy and Homonymy**

Polysemy (multiple meanings of the same word) and homonymy (different words with the same spelling) can confuse word embeddings:

* “**The bank is by the river.**” (referring to a financial institution) and “**I sat on the river bank.**” (referring to the river’s edge). The word “bank” has different meanings in these sentences.

**Inability to Capture Word Order**

Word embeddings fail to capture the impact of word order on meaning:

* “**I saw her with a telescope.**” (I used a telescope to see her) and “**I saw her with a microscope.**” (I saw her while using a microscope). The position of “with” changes the meaning, but word embeddings may not reflect this well.

**Fixed-Length Representations**

Word embeddings provide fixed-length representations for words, making it challenging to handle sentences or documents of varying lengths effectively. This can lead to information loss.

Sentence embeddings are similar to word embedding but instead of words, they encode the whole sentence into a vector representation. The obtained vector representation retains good properties by inheriting these features from underlying word embedding.

A simple way of obtaining sentence embedding is by averaging the word embeddings of all the words present in the sentence. But they are not accurate enough.

More advanced techniques like contextual embeddings (e.g., *BERT, ELMo, InferSent*) have been developed. These models consider the surrounding words and the order in which they appear, allowing for better capturing of sentence-level semantics. We’ll be covering them in the next steps.

Keep Learning till then!