Published in Towards Data Science

Ria Kulshrestha Follow

Nov 24, 2019 · 7 min read · Listen

Save

# NLP 101: Word2Vec — Skip-gram and CBOW

A crash course in word embedding.


Photo by Sincerely Media on Unsplash

## What does word embedding mean?

Word embedding is just a fancy way of saying numerical representation of words. A good analogy would be how we use the RGB representation for colors.

## Why do we need word embedding?

As a human, intuitively speaking, it doesn't make much sense in wanting to represent words or any other object in the universe using numbers because numbers are used for quantification and why would one need to quantify words?

When in science, we say speed of my car is 45 km/hr we gain a sense of how fast/slow we are driving. If we say my friend is driving at 60 km/hr, we can compare which one of us is going faster. Furthermore, we can calculate where we will be at a certain point in time, when we will reach our destination given we know the distance of our journey etc etc.
Similarly, outside of science, we use numbers to quantify a quality, when we quote the price of an object we try to quantify its worth, the size of a garment we try to quantify the body proportions it will fit best.

All of these representations make sense because by using numbers we have made analysis and comparisons based on those qualities much much easier. What's worth more a shoe or a purse? Well, as different as those two objects are, one way to answer that is to compare their prices. Other than the quantification aspect, there isn't any thing else to be gained by this representation.

Now that we know numerical representation of objects aids in analysis by quantifying a certain quality, the question is what quality of words do we want to quantify?

The answer to that is, we want to quantify the ***semantics***. We want to represent words in such a manner that it captures its meaning in a way humans do. Not the exact meaning of the word but a contextual one. For example, when I say the word *see,* we know exactly what action — the context — I'm talking about, even though we might not be able to quote its meaning, the kind we would find in a dictionary, of the top of our head.

## What are good quality word embedding and how to generate them?

The simplest word embedding you can have is using one-hot vectors. If you have 10,000 words in your vocabulary, then you can represent each word as a 1x10,000 vector.

For a simple example, if we have 4 words — *mango, strawberry, city, Delhi* — in our vocabulary then we can represent them as following:

- Mango [1, 0, 0, 0]
- Strawberry [0, 1, 0, 0]
- City [0, 0, 1, 0]
- Delhi [0, 0, 0, 1]

There are a few problems with the above approach, firstly, our size of vectors depends on the size of our vocabulary(which can be huge). This is a wastage of space and increases algorithm complexity exponentially resulting in the *curse of dimensionality*.

Secondly, these embedding will be closely coupled to their applications, making





Lastly, the entire purpose of creating embedding is to capture the contextual meaning of the words, which this representation fails to do. There is no co-relation between words that have similar meaning or usage.

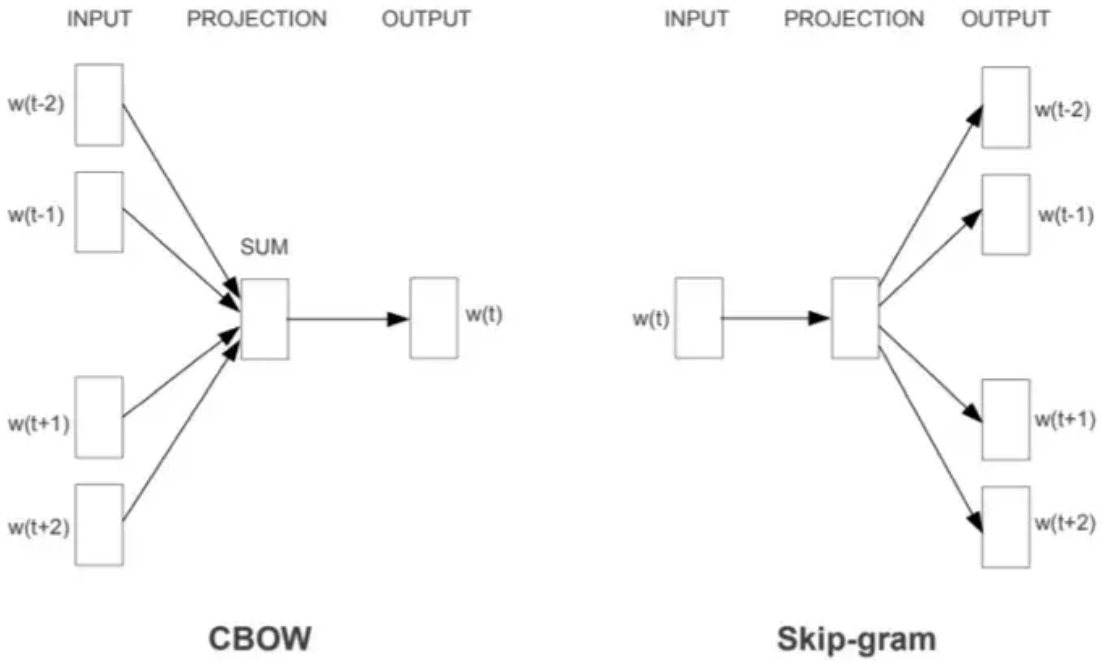
```
## Current situation
Similarity(Mango, Strawberry) == Similarity(Mango, City) == 0

## Ideal situation
Similarity(Mango, Strawberry) >> Similarity(Mango, City)

** Note: Similarity(a,b) = a.b/(||a||*||b||) Cosine similarity
```

#### Continuous Bag of Words Model (CBOW) and Skip-gram

Both are architectures to learn the underlying word representations for each word by using neural networks.




Source: Exploiting Similarities among Languages for Machine Translation paper.



In the **CBOW** model, the distributed representations of context (or surrounding words) are combined to **predict the word in the middle.** While in the **Skip-gram** model, the distributed representation of the input word is used to **predict the context.**

A prerequisite for any neural network or any supervised training technique is to have labeled training data. How do you a train a neural network to predict word embedding when you don't have any labeled data i.e words and their corresponding word embedding?

#### Skip-gram Model

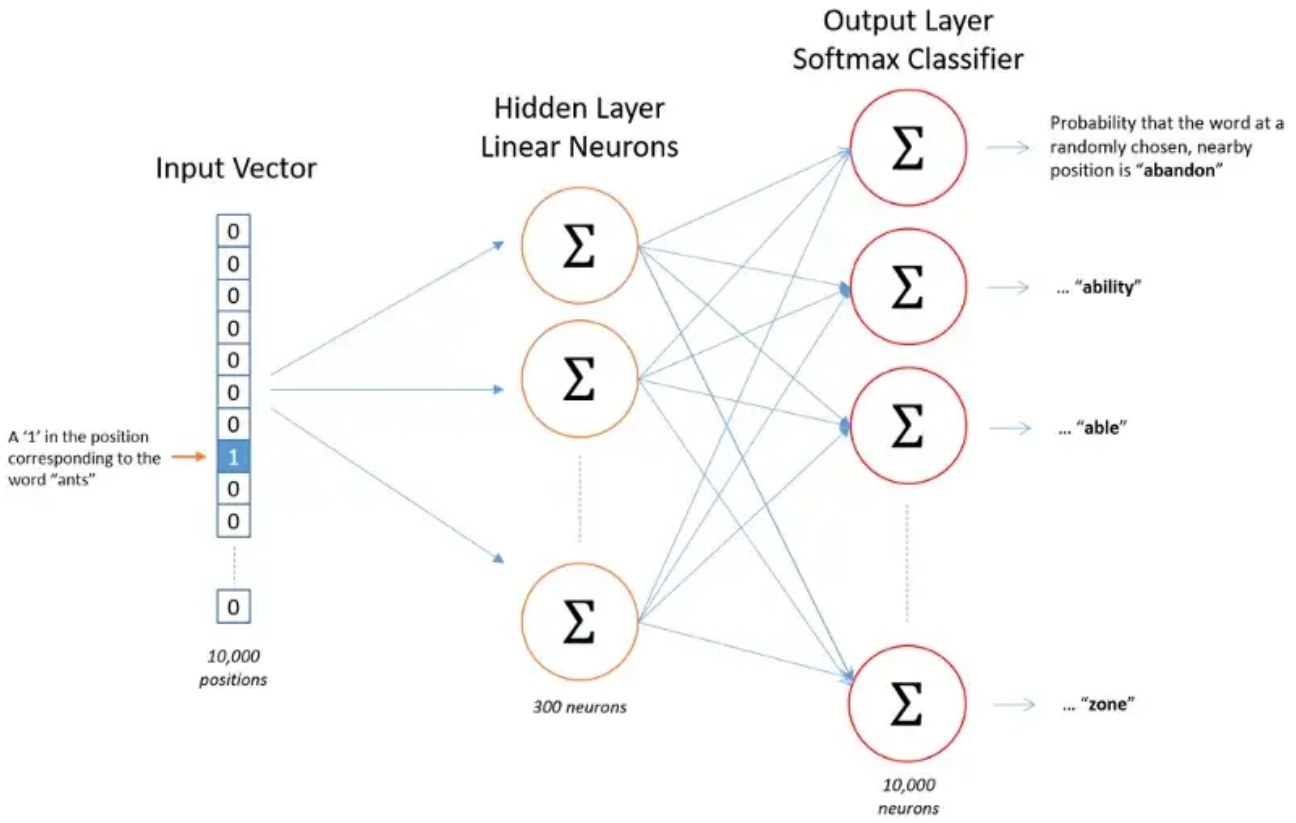
We'll do so by creating a "fake" task for the neural network to train. We won't be interested in the inputs and outputs of this network, rather the goal is actually just to learn the weights of the hidden layer that are actually the "word vectors" that we're trying to learn.

The fake task for Skip-gram model would be, given a word, we'll try to predict its neighboring words. We'll define a neighboring word by the window size — a hyper-parameter.

| Source Text | Training Samples generated from source text | | | |
|---|---|---|---|---|
| I will have orange juice and eggs for breakfast | (will, I) | (will, have) | (will, orange) | |
| I will have orange juice and eggs for breakfast | ( have, I) | (have, will) | (have, orange) | (have, juice) |
| I will have orange juice and eggs for breakfast | (orange, will) | (orange, have) | (orange, juice) | (orange, and) |
| I will have orange juice and eggs for breakfast | (juice, have) | (juice, orange) | (juice, and) | (juice, eggs) |
| I will have orange juice and eggs for breakfast | (and, orange) | (and, juice) | (and, eggs) | (and, for) |
| I will have orange juice and eggs for breakfast | (eggs, juice) | (eggs, and) | (eggs, for) | (eggs, breakfast) |
| I will have orange juice and eggs for breakfast | ( for, and) | ( for, eggs) | ( for, breakfast) | |

The word highlighted in yellow is the source word and the words highlighted in green are its neighboring words.

Given the sentence:
*"I will have orange **juice** and eggs for breakfast."*
and a window size of 2, if the target word is *juice,* its neighboring words will be *( have, orange, and, eggs).* Our input and target word pair would be (juice, have), (juice, orange), (juice, and), (juice, eggs).
Also note that within the sample window, proximity of the words to the source word plays no role. So *have, orange, and,* and *eggs* will be treated the same while training.




Architecture for skip-gram model. Source: McCormickml tutorial

The dimensions of the input vector will be 1xV — where V is the *number of words in the vocabulary* — i.e one-hot representation of the word. The single hidden layer will have dimension VxE, where E is the size of the word embedding and is a hyper-parameter. The output from the hidden layer would be of the dimension 1xE, which we will feed into an softmax layer. The dimensions of the output layer will be 1xV, where each value in the vector will be the probability score of the target word at that position.
According to our earlier example if we have a vector [0.2, 0.1, 0.3, 0.4], the probability of the word being *mango* is 0.2, *strawberry* is 0.1, *city* is 0.3 and *Delhi* is 0.4.

The back propagation for training samples corresponding to a source word is done in one back pass. So for *juice,* we will complete the forward pass for all 4 target words *( have, orange, and, eggs).* We will then calculate the errors vectors[1xV dimension] corresponding to each target word. We will now have 4 1xV error vectors and will perform an element-wise sum to get a 1xV vector. The weights of the hidden layer will be updated based on this cumulative 1xV error vector.

**CBOW**

The fake task in CBOW is somewhat similar to Skip-gram, in the sense that we still take a pair of words and teach the model that they co-occur but instead of adding the errors we add the input words for the same target word.

The dimension of our hidden layer and output layer will remain the same. Only the dimension of our input layer and the calculation of hidden layer activations will change, if we have 4 context words for a single target word, we will have 4 1xV input vectors. Each will be multiplied with the VxE hidden layer returning 1xE vectors. All 4 1xE vectors will be averaged element-wise to obtain the final activation which then will be fed into the softmax layer.

**Skip-gram**: works well with a small amount of the training data, represents well even rare words or phrases.
**CBOW**: several times faster to train than the skip-gram, slightly better accuracy for the frequent words.

In part II of this post: NLP 101: Negative Sampling and GloVe, we discuss:

- **Negative Sampling** — a technique to improve the learning without compromising the quality of embedding
- Another word embedding called **GloVe** that is a hybrid of count based and window based model.

## References


- Lecture notes CS224D: Deep Learning for NLP Part-I
- Lecture notes CS224D: Deep Learning for NLP Part-II
- McCormick, C. (2016, April 19). *Word2Vec Tutorial — The Skip-Gram Model.*


#### Other Articles by Me That I think You would Enjoy :D

- Yes, you should listen to Andrej Karpathy, and understand Back propagation
- Evaluation of an NLP model — latest benchmarks
- Understanding Attention In Deep Learning

- Transformers — the basic block for models such as Google's BERT and OpenAI's GPT.

> *I'm glad you made it till the end of this article.* 🎉
> *I hope your reading experience was as enriching as the one I had writing this.* 💖
>
> *Do check out my other articles here.*
>
> *If you want to reach out to me, my medium of choice would be Twitter.*

Machine Learning NLP Artificial Intelligence Word 2 Vec AI


Download on the
App Store


GET IT ON
Google Play