



Published in Analytics Vidhya



Manoj Akella

Follow

Oct 21, 2019 · 7 min read · [Listen](#)



Save



Word2Vec(SkipGram) Explained!

Where Text meets AI

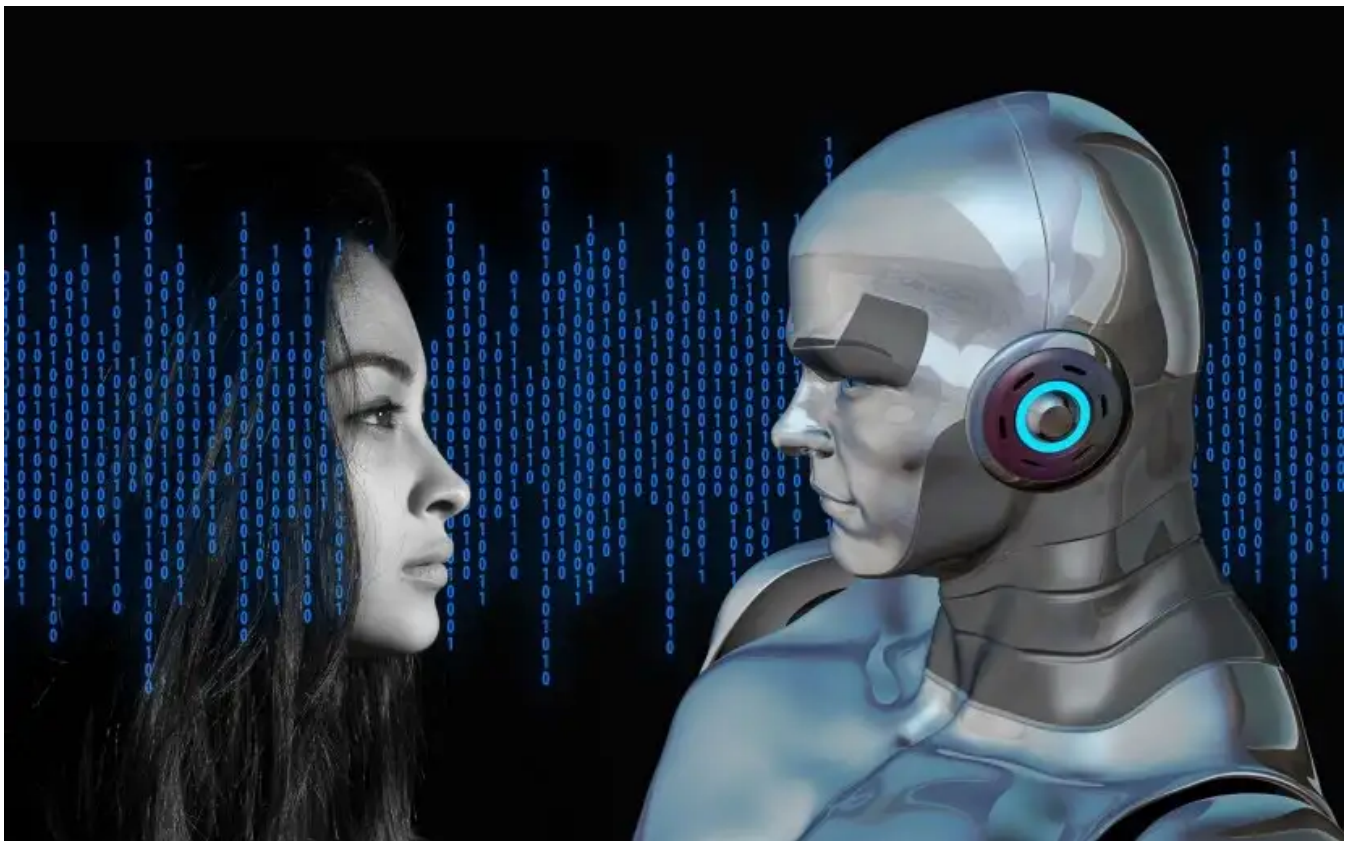


Image by [Gerd Altmann](#) from [Pixabay](#).

Introduction

In this article, we explore what exactly is the skip-gram model in Word2Vec, how the vectors are created and how they can be used. The focus is not on the introduction but on how to arrive at the word vectors.

Word2Vec is one way of creating a ‘vector representation’ of words. These are often referred to as word embeddings. The intention here is to bring words with similar meaning(or context) close together and words that are not related, far away from each other.

For example, let us assume each word in our vocabulary is represented using a 100 dimension vector. Then, with a proper embedding of the words, we should be able to see that vectors for ‘Apple’ and ‘Orange’ should be closer compared to vectors for ‘Apple’ and ‘Hydrogen’. When we say ‘closer’, we are referring to ‘cosine’ distance between these vector representations.

Now, let us see how we build these vectors. This tutorial will only cover the skip-gram model (Topics like CBOW or Glove are parked for a later article). We will see that Skip-gram turns out to be a very simple use case of neural networks (with just one hidden layer) and the way we arrive at the embeddings is too simple to believe.

Skip-gram Model

At a high level, given a word in context, the skip-gram model will try to predict a nearby word (as a target). Let us elaborate on this in detail.

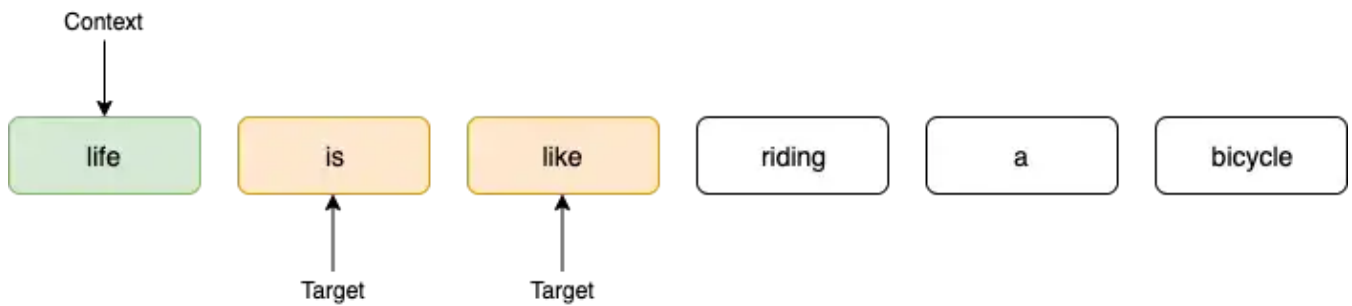
The Context and Target

To understand this, let us take the sentence “Life is like riding a bicycle”. Each word (or token) in this sentence can be considered as a ‘context’ and with this context, we can predict a nearby word. Obviously, we cannot define a word that is 5 paragraphs away as a ‘nearby’ word. So, a small enough window size has to be chosen to decide which words are occurring together. Let us take a window size of 2 words for our understanding.

Original Text



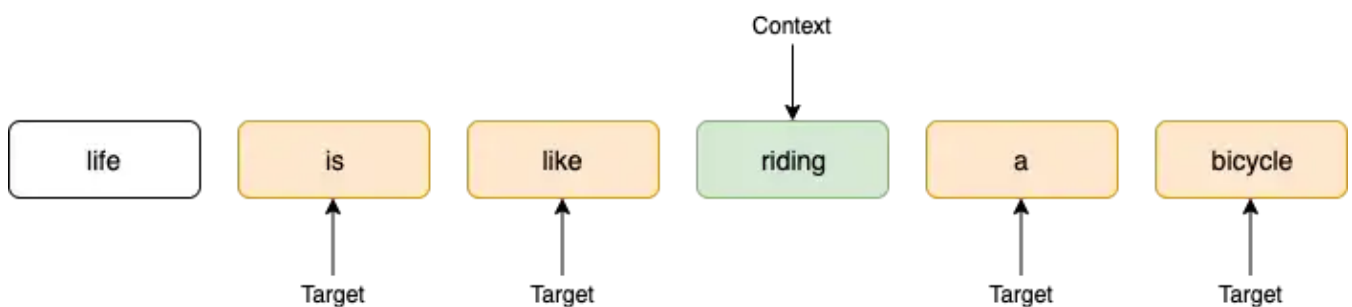
Let us use ‘life’ as a context word as an example. Because we took window size as 2, the words ‘is’ and ‘like’ are the targets.



Below are the context and its corresponding targets.

Context	Target
life	is
life	like

Just to make sure this idea is understood, the same step is repeated for another context word 'riding'.



Below are the context and its corresponding targets.

Context	Target
riding	is
riding	like
riding	a
riding	bicycle

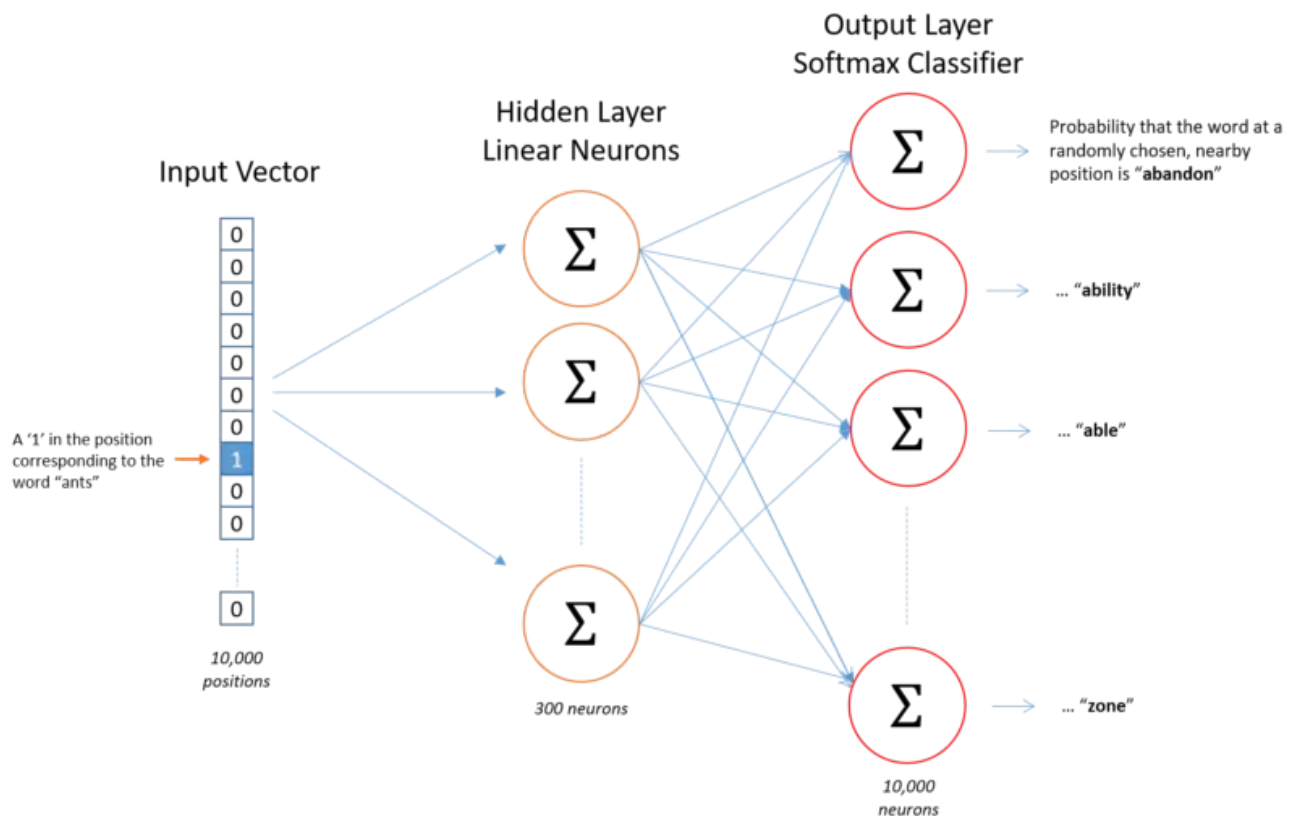
With this understanding, the context and target word combinations can be built for the entire corpus.

Here, a neural network is used to predict a target word using a context word. As the neural network learns and updates the weights of the hidden layer with a large

enough corpus, these weights will magically become our word vectors that represent the context word.

The Network

Now that the context and target are available, a single layer neural network is used to predict the target words using the context words. Let's dig a little deeper into what the neural network looks like.



source: <http://mccormickml.com/>

Input layer

As mentioned above, both the source and the target are the words picked from the corpus for the training of the neural network. The way each word is represented as input is using a one-hot encoded approach.

If there are 10,000 unique words in the corpus, the input is an array of length 10,000 where only one of them is '1' and rest are zero for any word.

Let us observe how the input vectors look like for the example we have taken above. Below are the one-hot representations of words we have taken (in real life, the words are sorted in alphabetical order before one-hot vectors are assigned).

life	is	like	riding	a	bicycle
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

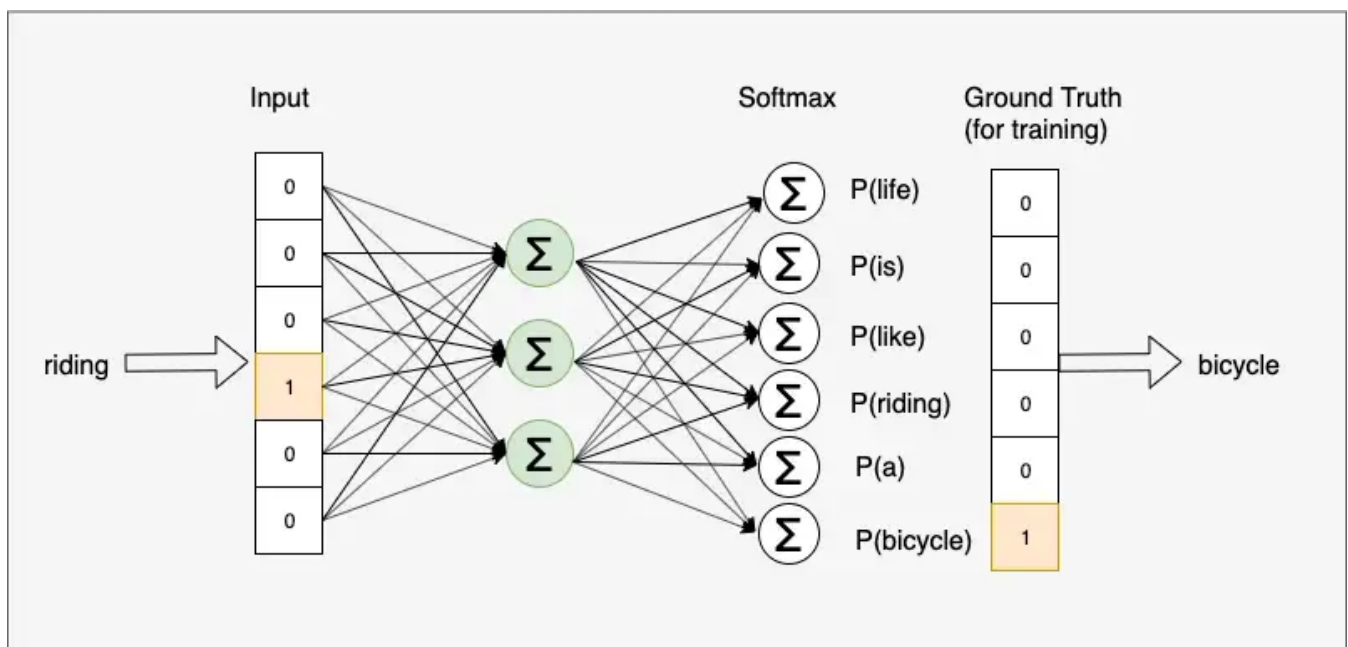
Output Layer

The output of the neural network is the probability of a particular word being ‘nearby’ for the given input word. If there are 10,000 words in the corpus, the output layer is an array of length 10,000 and each element is a value between 0 and 1 representing the probability of a target word occurring along with the context word.

Training

The neural network predicts a target word given an input word. The neural network takes the one-hot representation of input word as input and predicts the probability for each of the target words as output. For training the model and to minimize the loss, we need the actual values to compare with the predicted values and to calculate the loss. The one-hot representation of the target words will serve as the actual values of the output layer for this purpose. A Softmax layer is used at the output of the neural network.

Let us observe how the network will look like for the example sentence taken. Here, ‘riding’ is considered the center word while ‘bicycle’ is the target

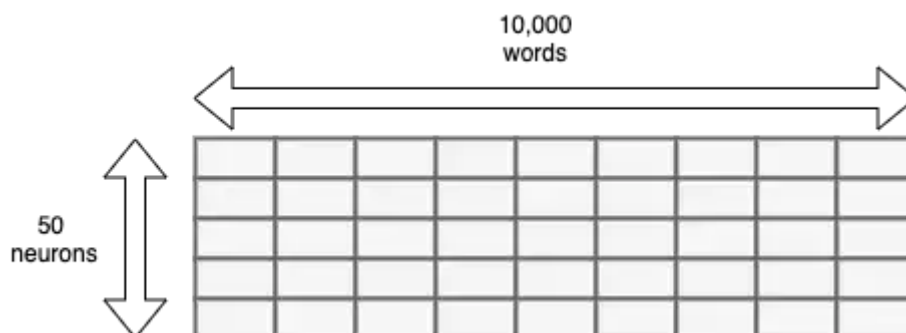


Hidden Layer

Here is where all the magic happens!!

The number of neurons in the hidden layer will define how (and to what dimension) we ‘embed’ the 10,000 length sparse vectors. Let us take 50 as the dimension of the hidden layer. This means we are creating 50 features for each word that passes through the network.

Let us now focus on the weights. in our example, the input layer is of size 10,000 and we just decided that the hidden layer is of size 50. Hence, the Weights matrix is of size (50 x 10000)



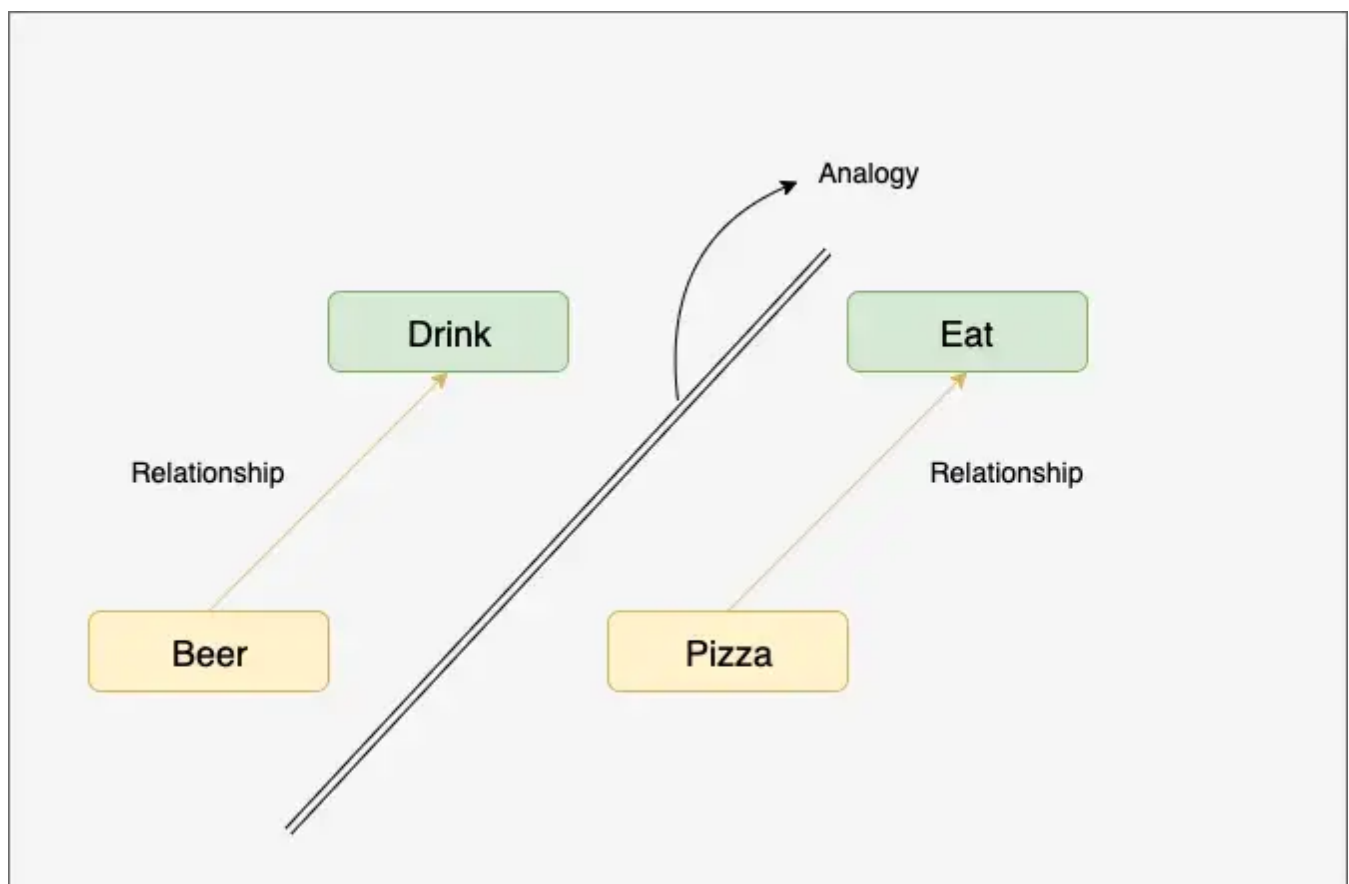
Firstly, let us look at weights from the hidden layer perspective. For the first neuron in the hidden layer, there are 10,000 input connections one from each element of the input array. But, because the input layer is a one-hot representation, only one of input will be coming in as ‘1’ and the rest of the inputs are zeroes. So, only one of the 10,000 weights will be passed on to the activation function for any word. this is true for all the 50 neurons in the hidden layer.

Now, let us look at the weights from the input layer perspective. For any word, only one element of the input array is '1' out of 10,000 because of one-hot representation. For that input element, there are 50 connections (weights) to each of the 50 neurons in the hidden layer. When the next word comes as input, another input element will have a value of 1 and it will have its own 50 connections (weights) to the hidden layer. So, in a way, each word in the corpus will have its own set of 50 weights which will be used when that word appears in the context.

These 50 weights are the word vectors that represent the words in the corpus.

Intuition

This is where things will get clear. Let us say two words have a similar context. If we take all the words occurring around the first word and all the words occurring around the second word, it is fair to say there will be some words in common. From the neural network perspective, in this case, two different context words are giving the same target words as output. Hence, weights for these context words will be updated similarly.



Spatial representation of words

As a larger number of target words turn out to be the same for two context words, their weight updates will be similar to each other. Hence, the final weights for the two context words will be close to each other. With a large corpus and good enough dimension of the hidden layer, these weights will turn out to be good spatial representations of the words.

With these weights, we can perform several tasks ranging from finding similar words, word analogies (e.g. king is to a man like a queen is to a woman) to more complex tasks like Named entity recognition, parts of speech tagging, etc

When Google published its paper on Word2Vec, it used a 300 dimension vector representation and the training is done on Google News articles.

Next

This article only covers the details of the initial implementation of Word2Vec using the Skip-gram model. But as it can be observed from the size of the weights matrix, getting the embedding layer is a computationally intense process. How Google handled this will be discussed in the next article in detail.

The next article also covers the python code to check similarities and analogies two or more words using Word2Vec.

Machine Learning

Natural language processing

Data Science

Deep Learning

Word Embeddings

Thanks to Chaitu Val Kan O



72





Sign up for Analytics Vidhya News Bytes

By Analytics Vidhya

Latest news from Analytics Vidhya on our Hackathons and some of our best articles! [Take a look.](#)

Your email

 Get this newsletter

Open in app 

Sign up

Sign In



[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

