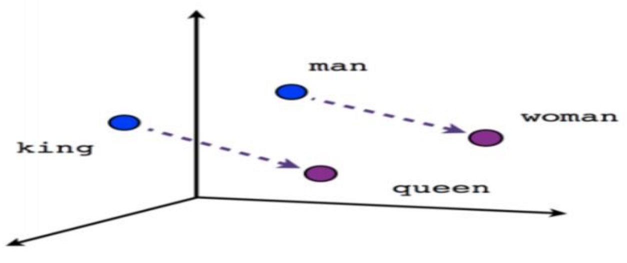
1. Introduction of Word2Vec
2. Introduction of CBOW Model
3. Introduction of Skip gram Model
4. Introduction of SoftMax function
5. Mathematical derivation of CBOW Model
6. Mathematical derivation of Skip gram Model
7. Case Study: Movie Review Sentiment analysis using word2vec

Word2Vec

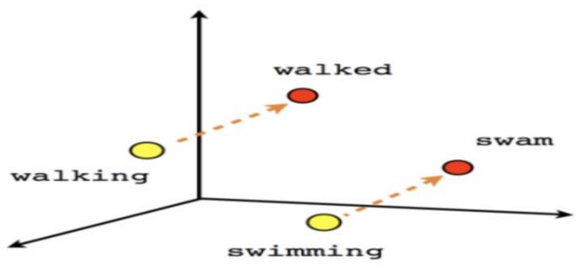
Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google.

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Word2vec represents words in vector space representation. Words are represented in the form of vectors and placement is done in such a way that similar meaning words appear together and dissimilar words are located far away. This is also termed as a semantic relationship. Neural networks do not understand text instead they understand only numbers. Word Embedding provides a way to convert text to a numeric vector.



Male-Female



Verb tense

Why do we need them?

Consider the following similar sentences:

1. *Have a good day*
2. *Have a great day.*

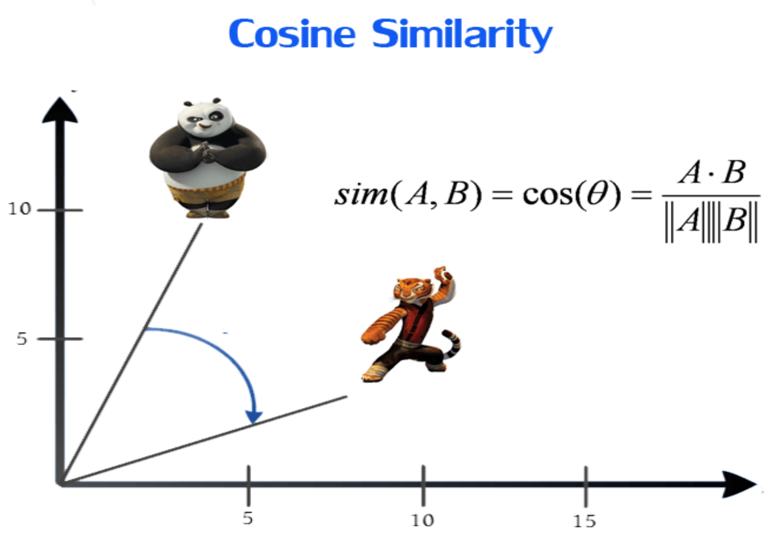
They hardly have different meaning. If we construct an exhaustive vocabulary (let’s call it V), it would have V = {Have, a, good, great, day}.

Now, let us create a one-hot encoded vector for each of these words in V. Length of our one-hot encoded vector would be equal to the size of V (=5). We would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary. That particular element would be one. The encodings below would explain this better.

Have = [1,0,0,0,0];a=[0,1,0,0,0] ; good=[0,0,1,0,0] ;great=[0,0,0,1,0] ; day=[0,0,0,0,1]

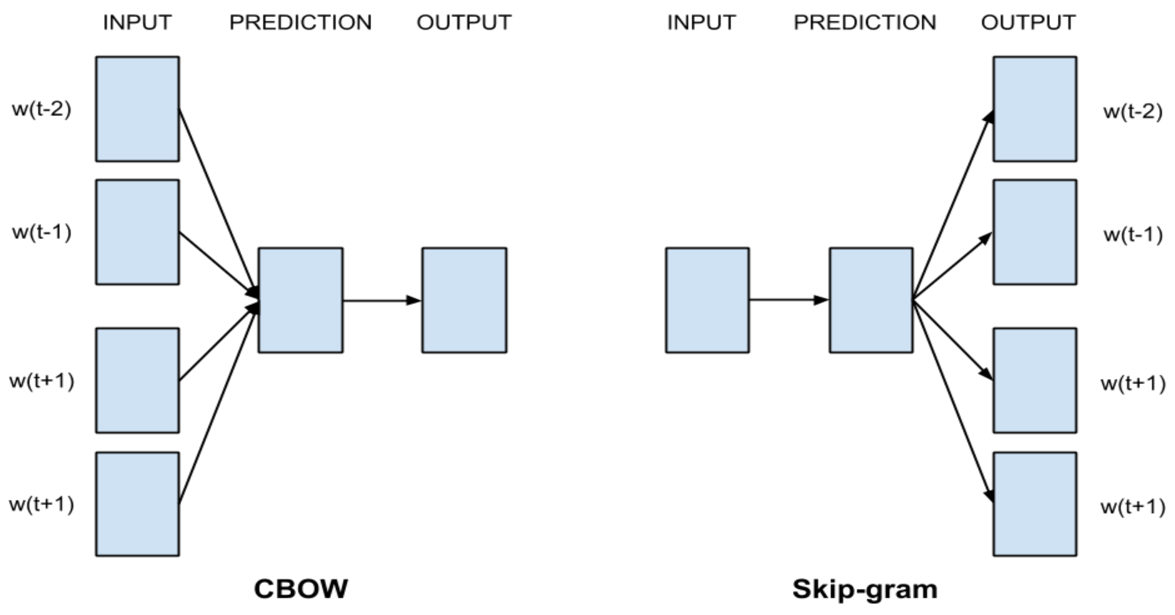
If we try to visualize these encodings, we can think of a 5-dimensional space, where each word occupies one of the dimensions and has nothing to do with the rest (no projection along the other dimensions). This means 'good' and 'great' are as different as 'day' and 'have', which is not true.

Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.

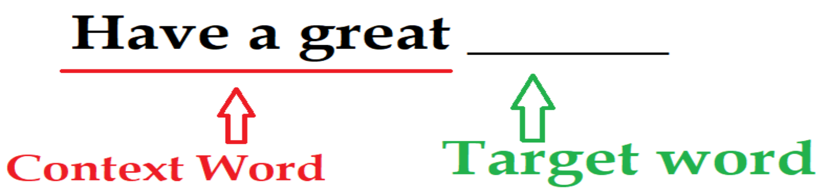


Word2Vec is a method to construct such an embedding. It can be obtained using two methods:

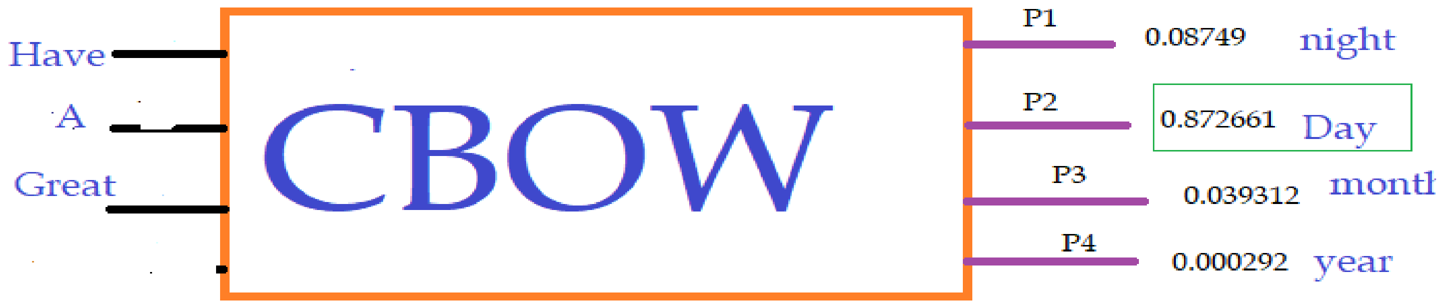
1. Common Bag of Words (CBOW)
2. Skip Gram

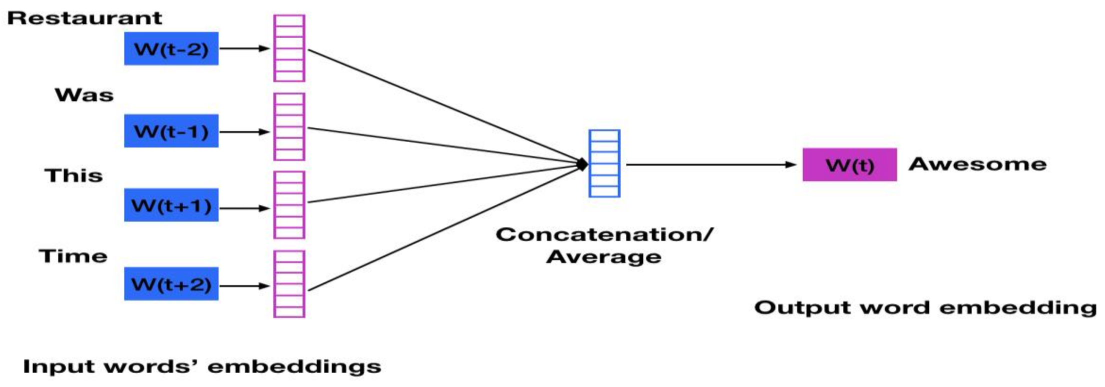


CBOW Model: This method takes the context of each word as the input and tries to predict the word corresponding to the context. Consider our example: *Have a great day*.

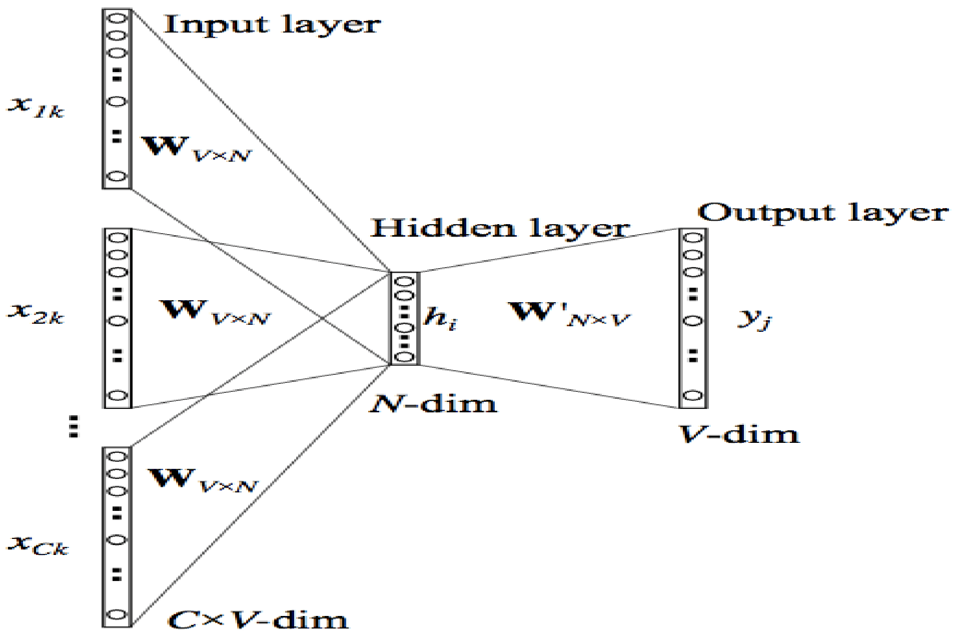


Let the input to the Neural Network be the word, great. Notice that here we are trying to predict a target word (day) using a single context input word great. More specifically, we use the one hot encoding of the input word and measure the output error compared to one hot encoding of the target word (day). In the process of predicting the target word, we learn the vector representation of the target word.

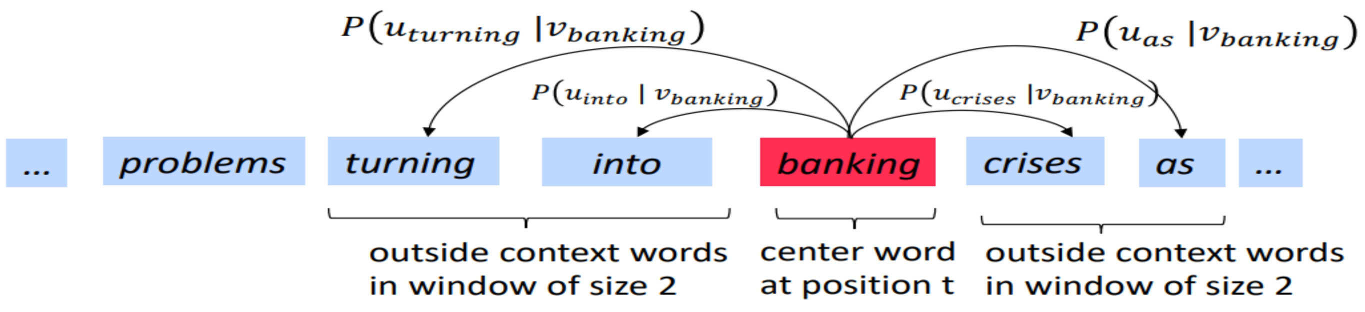




Let us look deeper into the actual architecture.

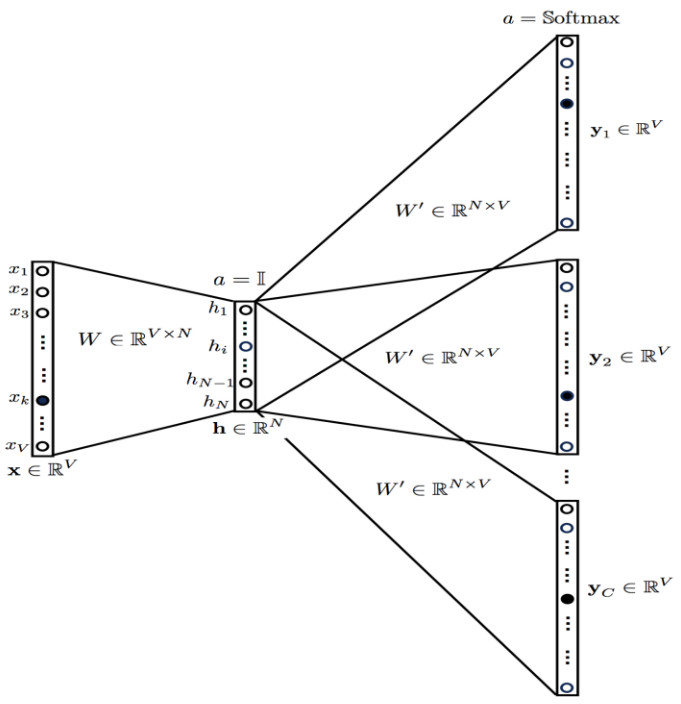


Skip-Gram: The input is the target word, while the outputs are the words surrounding the target words. It attempts to predict the immediate neighbours of a given word. It looks one word ahead and one behind, or two ahead, or some other variation. We use the word we want to model as our input X, and the surrounding words as target output Y.

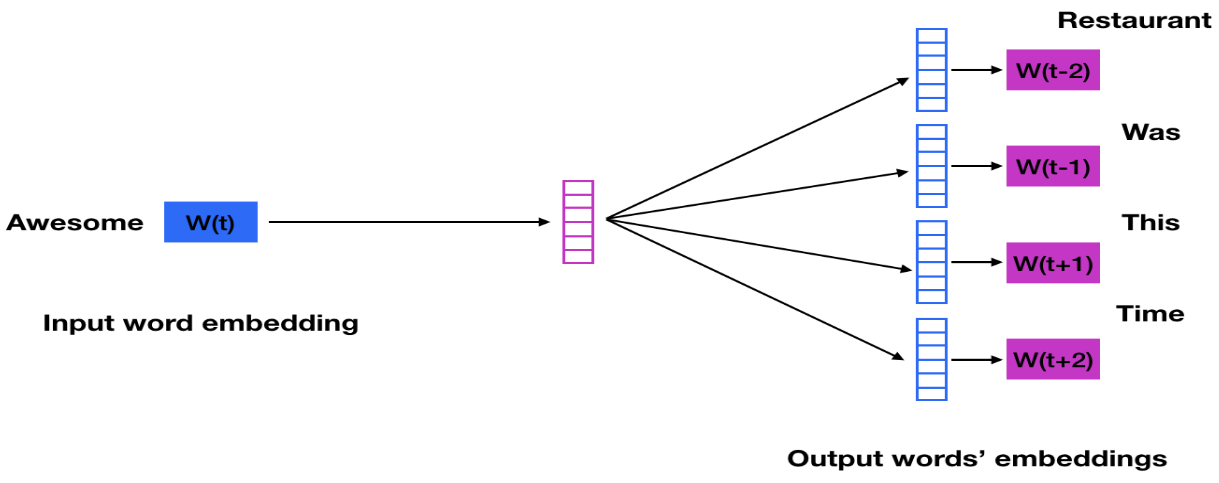


$P(u_{turning} | v_{banking})$
$P(u_{into} | v_{banking})$
$P(u_{crises} | v_{banking})$
$P(u_{as} | v_{banking})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

Let us look deeper into the actual architecture.

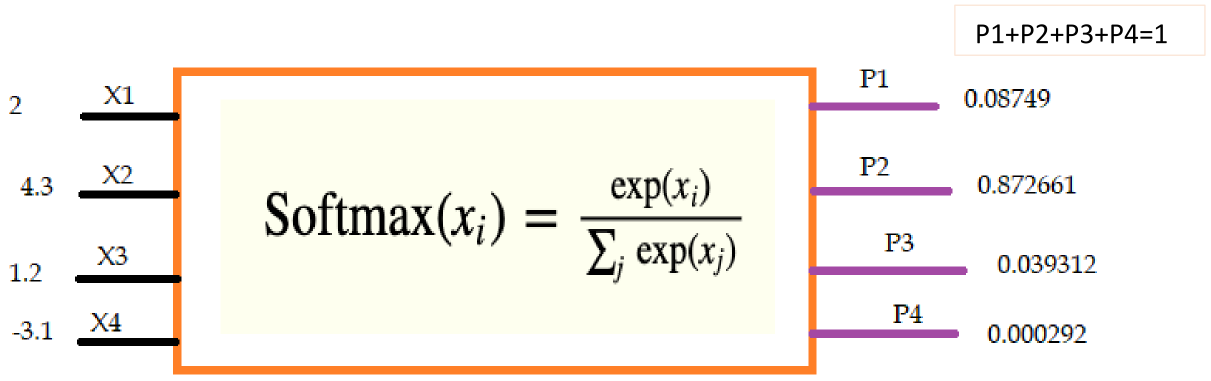


$a = \text{Softmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N\times V}$
$a = \mathbb{I}$
$W \in \mathbb{R}^{V\times N}$
x_1
x_2
x_3
x_k
x_V
$\mathbf{x} \in \mathbb{R}^V$
h_1
h_i
h_{N-1}
h_N
$\mathbf{h} \in \mathbb{R}^N$
$W' \in \mathbb{R}^{N\times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N\times V}$
$\mathbf{y}_C \in \mathbb{R}^V$



SoftMax function: -

Softmax function also known as soft argmax or normalized exponential function. This is a function that takes as input a vector of K real number and normalize into probability distribution consisting K probability proportional to the exponentials of the input number.



1. Prior to applying softmax, some vector components could be negative, or greater than one.
2. Negative input will be converted into non-negative values.
3. Each output will be in the interval (0,1).
4. As the denominator in each Softmax computation is the same,the value become proportional to each other, this normalization sure that together they sum to 1.

```
>>> import numpy as np
>>> a = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> np.exp(a) / np.sum(np.exp(a))
array([0.02364054, 0.06426166, 0.1746813, 0.474833, 0.02364054,
       0.06426166, 0.1746813])
```