1 Introduction

Word2Vec is a method that use a vector to represent a word, because calculating word in the network directly is difficult.

If you have two words that have very similar neighbors (meaning: the context in which its used is about the same), then these words are probably quite similar in meaning or are at least related. For example, the words shocked, appalled, and astonished are usually used in a similar context.

1.1 How to represent words?

To start off, we need to be able to represent words as input to our Machine Learning models. One mathematical way of representing words is as vectors. There are an estimated 13 million words in English language. But many of these are related. Spouse to partner, hotel to motel. So do we want separate vectors for all 13 million words?

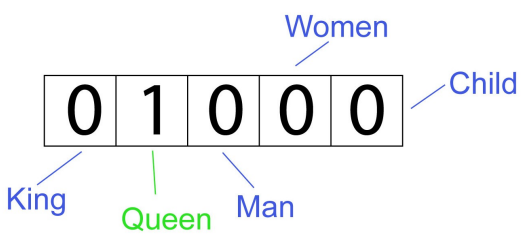
No. We must search for a N-dimensional vector space (where N <<13 million) that is sufficient to encode all semantics in our language. We need to have a sense of similarity and difference between words

1.2 How do we represent meaning of words?

If we use separate vectors for all 13 million words (or maybe more) in English vocabulary, we'll be facing several problems. Firstly, we'll have large vectors with a lot of 'zeroes' and one 'one' (in different position representing a different word). This is also known as *one-hot encoding*.

At one level, it's simply a vector of weights. In a simple 1-of-N (or 'one-hot') encoding every element in the vector is associated with a word in the vocabulary. The encoding of a given word is simply the vector in which the corresponding element is set to one, and all other elements are zero.

Suppose our vocabulary has only five words: King, Queen, Man, Woman, and Child. We could encode the word 'Queen' as:



Using such an encoding, there's no meaningful comparison we can make between word vectors other than equality testing.

In word2vec, a distributed representation of a word is used. Take a vector with several hundred dimensions (say 1000). Each word is representated by a distribution of weights across those elements. So instead of a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all of the elements in the vector, and each element in the vector contributes to the definition of many words.

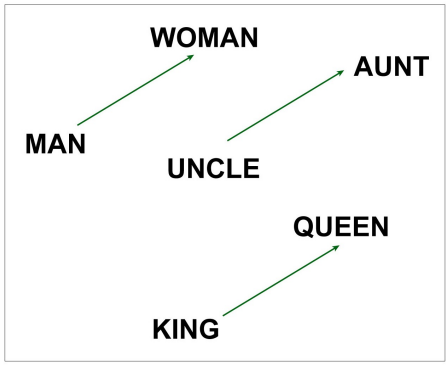
If I label the dimensions in a hypothetical word vector (there are no such pre-assigned labels in the algorithm), it might look a bit like this:

		king	Green	Women	Princess
Royalty		0.99	0.99	0.02	0.98
Masculinity		0.99	0.05	0.01	0.02
Femininity		0.05	0.93	0.999	0.94
Age		0.07	0.06	0.05	0.01

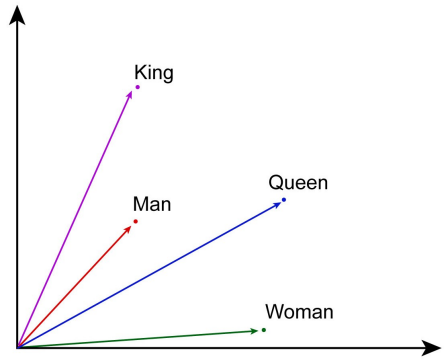
Such a vector comes to represent in some abstract way the 'meaning' of a word. By examining a large corpus (or a collection) it's possible to learn word vectors that are able to capture relationship between words in a expressive way.

The vectors are very good at answering analogy questions of the form a is to b as c is to ?. For example, man is to woman as uncle is to ? (aunt) using a simple vector offset method based on cosine distance.

For example, here are vector offsets for three word pairs illustrating the gender relation:



Vectors for King, Man, Queen, & Woman:

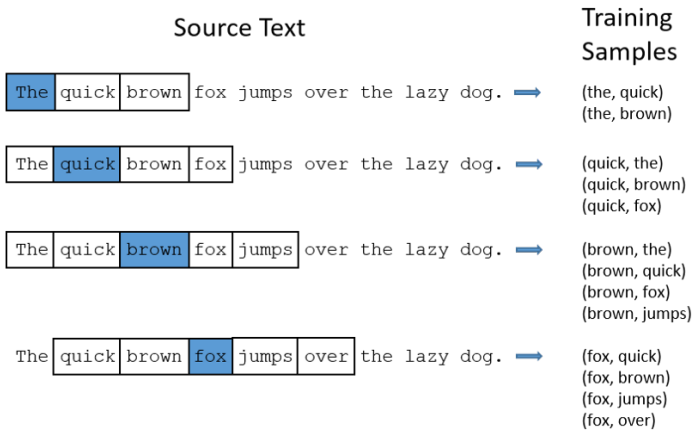


2 Word2Vec

How can we build simple, scalable, fast to train models which can run over billions of words that will produce exceedingly good word representations? Let's look into Word2Vec model to find answer to this.

Word2Vec is a group of models which helps derive relations between a word and its contextual words. The two important models in Word2Vec are: Skip-grams and Continuous Bag-of-Words model (CBOW)

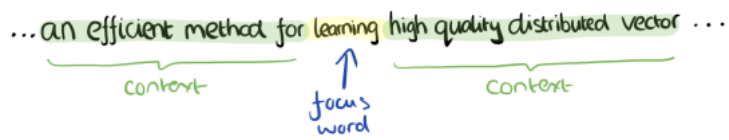
2.1 Skip-grams model



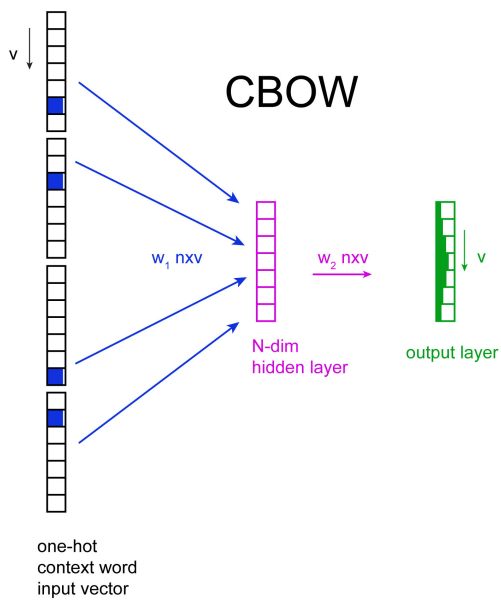
In Skip-gram model, we take a centre word and a window of context (neighbor) words and we try to predict context words out to some window size for each centre word. So, our model is going to define a probability distribution i.e. probability of a word appearing in context given a centre word and we are going to choose our vector representations to maximize the probability.

2.2 Continuous Bag-of-Words model

Consider a piece of prose such as "The Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precises syntatic and semantic word relationships." Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words and precede it, and the four words that follow it:

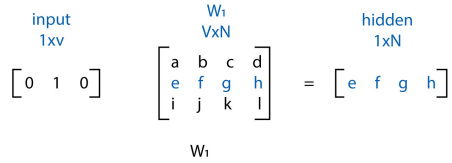


The context words form the input layer. Each word is encoded in one-hot form, so if the vocabulary size is V these will be V-dimensional vectors with just one of the elements set to one, and the rest all zeros. There is a single hidden layer and an output layer.



The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words, with regard to the weights. In our example, given the input ("an", "efficient", "method", "for", "high", "quality", "distributed", "vector") we want to maximize the probability of getting "learning" as the output.

Since our input vectors are one-hot, multiplying an input vector by the weight matrix W1 amounts to simply selecting a row from W1.



Given C input word vectors, the activation function for the hidden layer h amounts to simply summing the corresponding rows in W1, and dividing by C to take their average.

From the hidden layer to the output layer, the second weight matrix W2 can be used to compute a score for each word in the vocabulary, and softmax can be used to obtain the posterior distribution of words.

References

The blog https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/ An Intuitive Understanding of Word Embeddings
The blog https://code.google.com/archive/p/word2vec/ Introduction to Word2Vec
The blog https://www.deeplearningweekly.com/blog/demystifying-word2vec Word2Vec