# Natural Language Processing: Word2Vec

Nikhil Sardana

February 2018

## 1 Introduction

Natural Language Processing (NLP) is a field of computer science concerned with the generation, interpretation, parsing, and modification of written text. Although the field existed far before machine learning algorithms, we will focus primarily on deep learning applications to natural language processing.

Natural Language Processing algorithms are a significant part of our daily lives. Machine translation, like Google Translate, transform any language into another with mixed (although considerably improved compared to just a few years ago) results. Google Assistant, Siri, and Alexa parse and process voice commands and return information in a matter of seconds. Even the voice of Google Assistant, which relays this information back to you, is far more complex that simply concatenating small snippets of human recording.

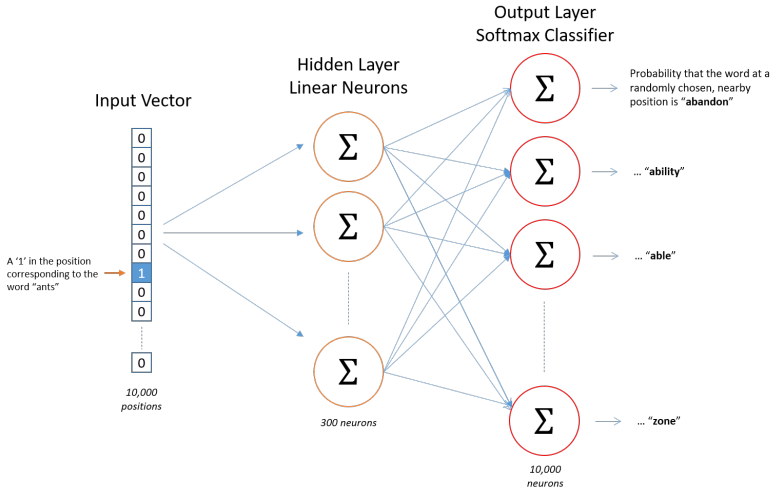
This lecture, along with subsequent NLP lectures, are not intended to be rigorous. The materials presented here and in future lectures cover numerous cutting-edge research topics, as well as many topics well-established in the field. Rather than delving deeply into one of these topics, I've chosen to present a rather high-level overview of many different breakthroughs in a series of lectures, as a sort of introduction into the field of NLP.

It should be noted that the main driving networks behind much of this work, as well as the field of Deep Learning for NLP as a whole for the past few years, are Recurrent Neural Networks and Convolutional Networks, both of which we have explained in previous lectures.

## 2 Word2Vec

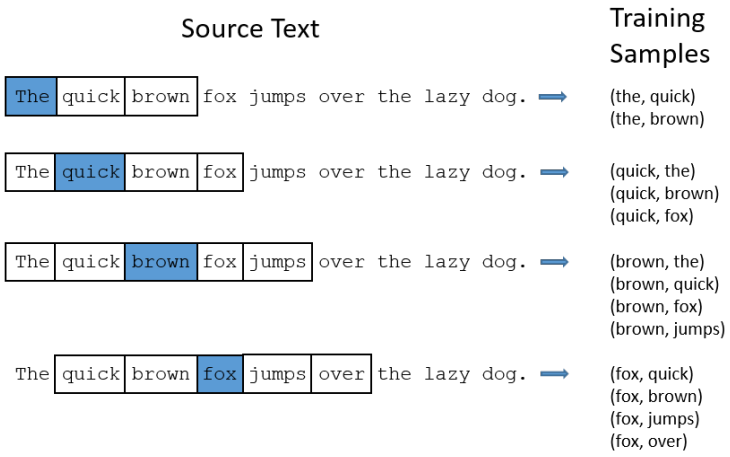
Given some input word, how can we determine probability that another given word will occur near it? Word2Vec is a simple model that learns this relationship between words. Essentially, Word2Vec is a standard neural network, with one hidden layer. A word is represented a one-hot encoded input vector. For example, if we have a $10,000$ word dictionary, the very first word of the dictionary would be represented like the $10000 \times 1$ column vector on the left.

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \cdots \end{bmatrix}$$



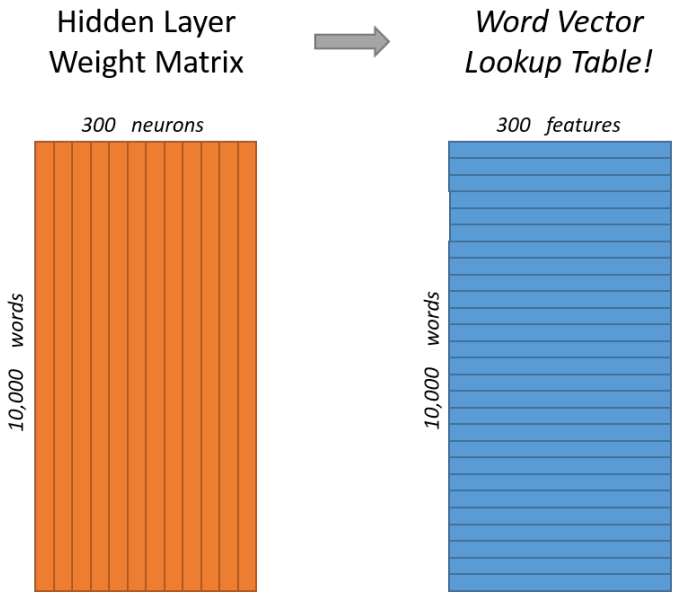
The neural network on the right shows the rest of the Word2Vec model: a single hidden layer and an output layer with the same number of neurons as the input. This is because the output is simply a Softmax: for some input vector word, the network will output a 10,000-row vector containing the probability every word in the dictionary will be near the input word.

How exactly do we feed this network training data? And what constitutes *near*? Well, we set nearness according to window size (see below for example window = 2).




The words in the white boxes, or window, are considered *near* the word in the blue box, and we use those as positive training data samples. We train it on a large selection of text (and I mean large. The original was trained on Google News.), and eventually we get a fairly decent Word2Vec network.

Now here's the thing. We don't actually care about the output layer. Word2Vec is useful not because it gives us the probability of some other word being close to another, but because the hidden layer gives us vector representations of words. In the particular network on Page 1, we have a $10,000$-row one-hot encoded input vector, and a 300-neuron hidden layer. This means that our first weight matrix is $300 \times 10000$.

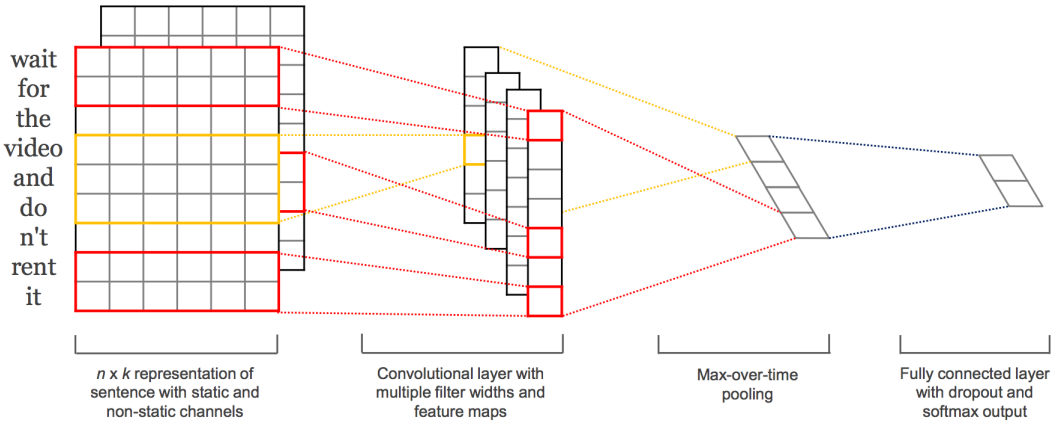



When we take some one-hot encoded input vector, and matrix multiply it by this $300 \times 10000$ weight matrix, we essentially get one column of this weight matrix—a $300 \times 1$ vector representing characteristics of the word.

$$\begin{bmatrix} 2.3 & 1.3 & 3.4 & -1.4 & \dots \\ 3.4 & -2.3 & 5.1 & 2.2 & \dots \\ 1.6 & -5.1 & 8.1 & 3.2 & \dots \\ 5.3 & 1.6 & 2.9 & 0.3 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \end{bmatrix} = \begin{bmatrix} 3.4 \\ 5.1 \\ 8.1 \\ 2.9 \\ \dots \end{bmatrix}$$

These are essentially what we are after—vector representations of words. Instead of representing a word in a dictionary as a string of arbitrary length, composed of difficult-to-manipulate English or Spanish or Portuguese or Arabic letters, each word is simply a vector—and if you've learned anything from these lectures, its that we love vectors and the matrix operations we manipulate them with.

Let's look at a more concrete example of the usefulness of vector representations. What if we wished to classify a sequence of words, but did not wish to use an LSTM or RNN. If we have word vectors, we can simply create an input "feature map" of sorts, by combining these word vectors into a matrix.

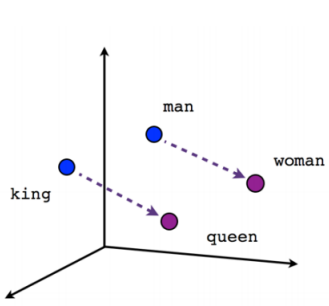



From there, we can simply use a standard convolutional network, treating the collection of word vectors as we would an image input.

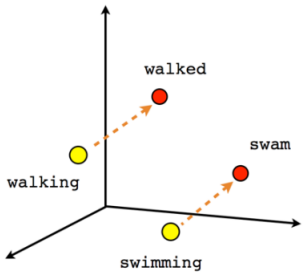
Another use-case of word vectors is in the comparison of words—not in terms of nearness, as previously described, but in terms of meaning. We can add and subtract words, after all, if they are just vectors. We can even visualize them, by reducing vectors into lower-dimensional space, we can see that the network has learned

$$\text{king} : \text{queen} :: \text{man} : \text{woman}$$

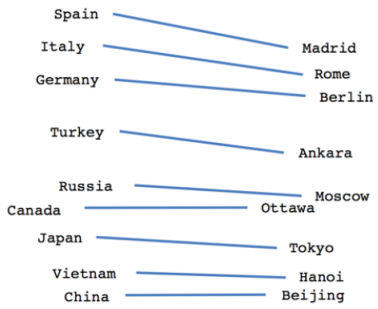
along with other various other results, a few of which are shown below.




Male-Female




Verb tense




Country-Capital