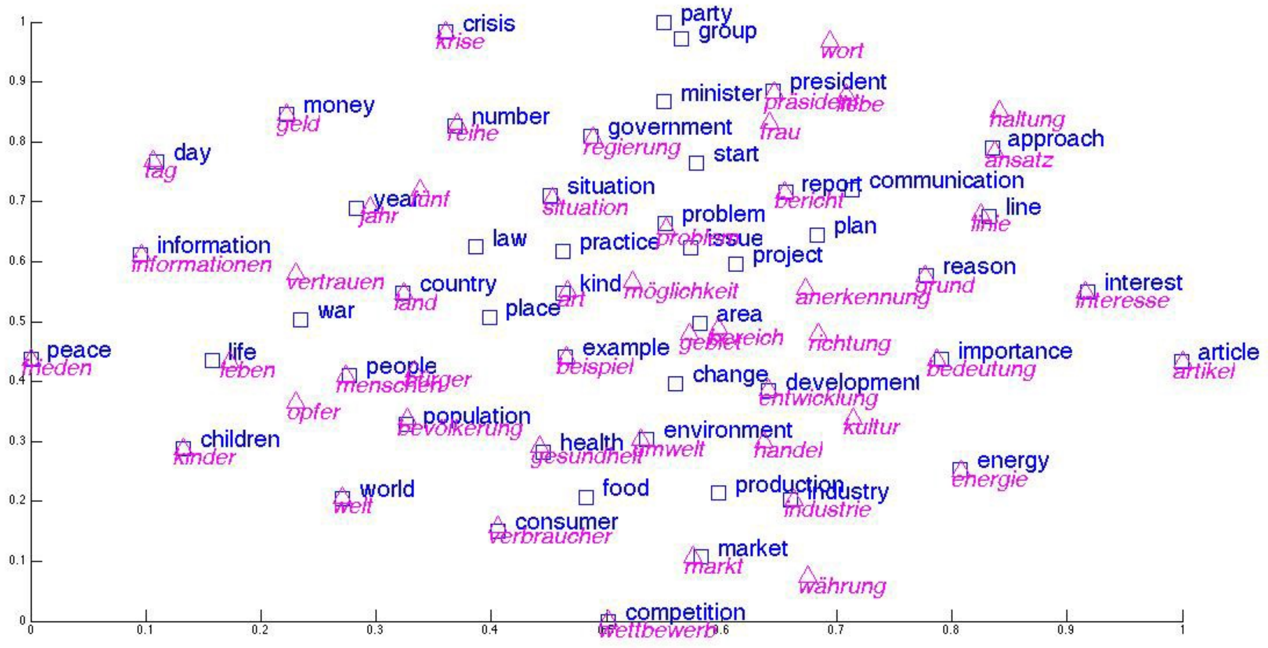
Glossary of Deep Learning: Word Embedding

Jaron Collis Follow
Apr 19, 2017 · 5 min read



A plot of word embeddings in English and German. The semantic equivalence of words has been inferred by their context, so similar meanings are co-located. This is because the relative semantics of words are consistent, whatever the language. [Source: Luong et al]

The Big Idea of Word Embedding is to turn text into numbers.

This transformation is necessary because many machine learning algorithms (including deep nets) require their input to be vectors of continuous values; they just won't work on strings of plain text.

So a natural language modelling technique like Word Embedding is used to map words or phrases from a vocabulary to a corresponding vector of real numbers. As well as being amenable to processing by learning algorithms, this vector representation has two important and advantageous properties:

- **Dimensionality Reduction** — it is a more efficient representation
- **Contextual Similarity** — it is a more expressive representation

If you're familiar with the Bag of Words approach, you'll know it results in huge, very sparse one-hot encoded vectors, where the dimensionality of the vectors representing each document is equal to the size of the supported vocabulary. Word Embedding aims to create a vector representation with a much lower dimensional space. These are called *Word Vectors*.

Word Vectors are used for semantic parsing, to extract meaning from text to enable natural language understanding. For a language model to be able to predict the meaning of text, it needs to be aware of the contextual similarity of words. For instance, that we tend to find fruit words (like apple or orange) in sentences where they're grown, picked, eaten and juiced, but wouldn't expect to find those same concepts in such close proximity to, say, the word aeroplane.

The vectors created by Word Embedding preserve these similarities, so words that regularly occur nearby in text will also be in close proximity in vector space. For examples of why this is useful, check out The amazing power of word vectors or this intro to Distributed Word Vectors on Kaggle.

So an answer to "What is word embedding?" is: it's a means of **building a low-dimensional vector representation from corpus of text, which preserves the contextual similarity of words.**

. . .

Introducing Word2Vec

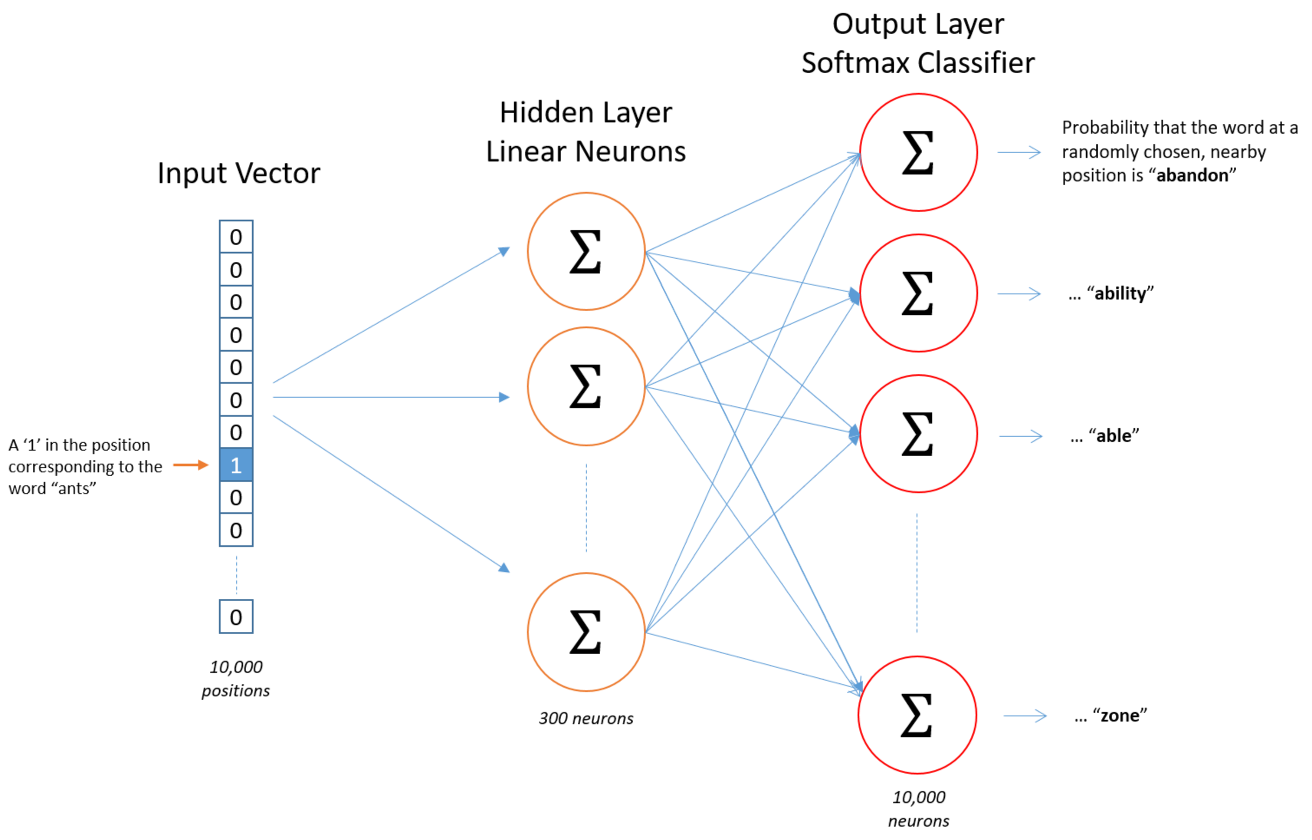
Now we know what it is, how does it work? If you've encountered dimensionality reduction before you'll know this is typically achieved using an unsupervised learning algorithm. And this is the approach used by one of the best known algorithms for producing word embeddings: word2vec.

There are actually two ways to implement word2vec, CBOW (Continuous Bag-Of-Words) and Skip-gram.

In CBOW we have a window around some target word and then consider the words around it (its context). We supply those words as input into our network and then use it to try to predict the target word.

Skip-gram does the opposite, you have a target word, and you try to predict the words that are in the window around that word, i.e. predict the context around a word.

The input words are passed in as one-hot encoded vectors. This will go into a hidden layer of linear units, then into a softmax layer to make a prediction. The idea here is to train the hidden layer weight matrix to find efficient representations for our words. This weight matrix is usually called the **embedding** matrix, and can be queried as a look-up table.



The word2vec architecture consists of a hidden layer and an output layer. [Source]

Xin Rong has created a very neat visual demo that shows how word embeddings are trained, and has given an excellent accompanying talk.

The embedding matrix has a size of the number of words by the number of neurons in the hidden layer (the embed size). So, if you have 10,000 words and 300 hidden units, the matrix will have size 10,000×300 (as we're using one-hot encoded vectors for our inputs). Once computed, getting the word vector is a speedy O(1) lookup of corresponding row of the results matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

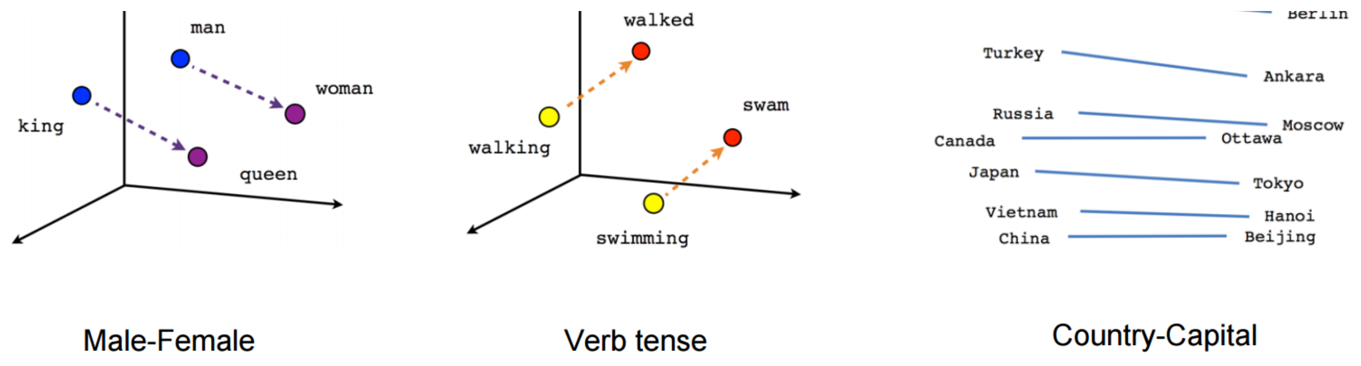
So, for the word that's the 4th entry in the vocabulary, its vector is (10,12,19).

So each word has an associated vector, hence the name: word2vec.

The embed size, which is the size of the hidden layer and thus the number of features that represent similarities between words, tends to be much smaller than the total number of unique words in the vocabulary, (hundreds rather than tens of thousands). The embed size used is a trade-off: more features mean extra computational complexity, and so longer run-times, but also allow more subtle representations, and potentially better models.

. . .

An interesting feature of word embeddings is that because they're numerical representations of contextual similarities between words, they can be manipulated arithmetically. The classic example is subtracting the 'notion' of "King" from "Man" and adding the notion of "Woman". The answer will depend on your training set, but you're likely to see one of the top results being the word "Queen". This great post explains why.



Word Embeddings are similarities based on context, which might be gender, tense, geography or something else entirely. The lines shown are just mathematical vectors, so see how you could move 'across' in embedding space from "Man" to "Queen" by subtracting "King" and adding "Woman".

Another word embedding algorithm worth knowing about is GloVe, which works slightly differently by accumulating counts of co-occurrences, (see How is GloVe different from word2vec?)

So where does deep learning come in?

Keep in mind that Word2vec is a two-layer shallow neural net, and so is not itself an example of deep learning. But techniques like Word2vec and GloVe can turn raw text into a numerical form that deep nets can understand, for instance, using Recurrent Neural Networks with Word Embeddings.

. . .

In summary then, the purpose of word embedding is to turn words into numbers, which algorithms like deep learning can then ingest and process, to formulate an understanding of natural language. Deeplearning4J has a lovely quote by the novelist EL Doctorow, who expresses this idea quite poetically:

> It's like numbers are language, like all the letters in the language are turned into numbers, and so it's something that everyone understands the same way. You lose the sounds of the letters and whether they click or pop or touch the palate, or go ooh or aah, and anything that can be misread or con you with its music or the pictures it puts in your mind, all of that is gone, along with the accent, and you have a new understanding entirely, a language of numbers, and everything becomes as clear to everyone as the

> writing on the wall. So as I say there comes a certain time for the reading of the numbers.

Word embeddings are just language, as numbers.

. . .

See also:

- A conceptual overview of word2vec from Chris McCormick
- word2vec paper and NIPS paper with improvements— Mikolov et al
- An implementation of word2vec — Thushan Ganegedara
- TensorFlow word2vec tutorial
- The amazing power of word vectors — Adrian Colyer
- An introduction to Distributed Word Vectors — Kaggle
- An online word embedding visualiser — Xin Rong
- How word similarities lead to analogies — Piotr Migdał
- What is word embedding in deep learning? — my original Quora post

. . .

SEE GLOSSARY INDEX

Machine Learning Artificial Intelligence NLP Deep Learning Data Science