Demystifying Word2Vec

by Jan Bussieck on February 05, 2017

Research into word embeddings is one of the most interesting in the deep learning world at the moment, even though they were introduced as early as 2003 by Bengio, et al. Most prominently among these new techniques has been a group of related algorithm commonly referred to as *Word2Vec* which came out of google research.[^2]
In this post we are going to investigate the significance of *Word2Vec* for NLP research going forward and how it relates and compares to prior art in the field. In particular we are going to examine some desired properties of word embeddings and the shortcomings of other popular approaches centered around the concept of a *Bag of Words* (henceforth referred to simply as *Bow*) such as Latent Semantic Analysis. This shall motivate a detailed exposition of how and why *Word2Vec* works and whether the word embeddings derived from this method can remedy some of the shortcomings of *BoW* based approaches. Word2Vec and the concept of word embeddings originate in the domain of NLP, however as we shall see the idea of words in the context of a sentence or a surrounding word window can be generalized to any problem domain dealing with sequences or sets of related data points.

Definition of Terms

For the sake of precision we have to define a few terms formally - A **word** is the basic unit of discrete data, defined to be an item from a vocabulary indexed by $1, \ldots, V$. We represent words using unit-basis vectors that have a single component equal to one and all other components equal to zero (one-hot encoding). Thus, using superscripts to denote components, the *vth* word

$D = w_1, w_2, \ldots, w_M$. - A word embedding $W : words -> \mathbb{R}^n$ is a parameterized function mapping words in some language to high-dimensional vectors (perhaps 200 to 500 dimensions). For example, we might find:

$$W(\text{" } cat \text{ "}) = (0.2, -0.4, 0.7, \ldots)$$

$$W(\text{" } mat \text{ "}) = (0.0, 0.6, -0.1, \ldots)$$

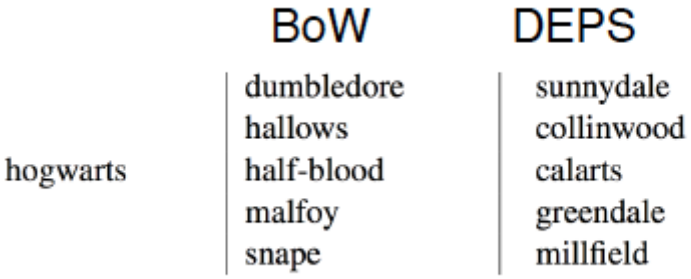
History and Landscape of Vector representations

Before the advent of word embeddings, words have been represented by models that made very strong simplifying assumptions, concretely, those representation rely on the notion of one hot encoded vectors, where a word is represented by a sparse vector with a dimension equal to the size of the vocabulary with a 1 at the index that stands for the word and zeros everywhere else. From this representation one can build up to a Bag of Words representation of text which represents a document by simply counting the number of times a word from the vocabulary occurs in it. From this document representation one can derive a word representation by using the co-occurrence of words in documents. Let us dig into one the most common of these methods; Latent Semantic Analysis (Deerwester et al., 1990).

Latent Semantic Analysis

Latent semantic analysis starts with a matrix, a so-called term document matrix, consisting of rows which stand for unique words and columns which correspond to a document, the entry is simply the frequency with which the word of the row appears in the document of the column. Instead of the raw term frequency one can also utilize a tf-idf transformation, that is a term-frequency-inverse-document-frequency which roughly corresponds to the frequency of a term within a document divided by its frequency in the entire corpus. Thereby expressing that terms which are common such as conventional stop words ("the", "a", "it" etc.) contribute less discriminative power to a document than terms which are highly document (or topic) specific. We can factorize this matrix with a method called singular value decomposition which yields three matrices, UDV^T where the columns of U and V are orthonormal and the matrix D is diagonal with positive real entries, the so-called singular values. These values are used to rank dimensions in U which allows one to identify the dimensions across which most of the variance in the space of tf-idf features is captured. That means that vectors of words that co-occur in certain documents, that is, words that exhibit similar variance across documents, end up close together in this new vector space called a topic space.

One shortcoming of *BoW* based methods is that co-occurrence in documents encodes a rather shallow kind of topical similarity, so that, for instance, "hogwarts" occurs close to other *Harry Potter* related terms as opposed to terms we would recognize as conceptually or functionally



From *Levy & Goldberg* (2014)

Word2Vec

Instead of relying on pre-computed co-occurrence counts, *Word2Vec* takes 'raw' text as input and learns a word by predicting its surrounding context (in the case of the cBoW model) or predict a word given its surrounding context (in the case of the skip-gram model) using gradient descent with randomly initialized vectors.
As an example let us look at the latter case applied to the following sentence.

"The fox jumped **over** the lazy dog"
Now, we want to learn the word vector for 'over' from its surrounding context, we call this vector v_{in}, *Word2Vec* uses different vectors for word embeddings depending on whether it is the word we are conditioning on or the word we are trying to predict. The probability we are trying to maximize is then:
$P(v_{out}|v_{in})$, where v_{out} is the output word and v_{in} the input.
The algorithm then moves over each word in the corpus and repeats the training step in an online fashion. The interesting property that word vectors obtained this way exhibit is that they encode not only syntactic but also semantic relationships between words. That means that not only are similar words close to each other in the vector space (as measured by some norm), but word analogies are reflected by the difference between word vectors. This property is referred to as 'additive compositionality' in the literature (Mikolov) and refers to the linear structure in the vector space that allows analogical reasoning. Word vectors can be seen as representing the distribution of the context in which a word appears and the sum of vectors roughly represents an AND concatenation, so if for instance 'Volga River' appears in the same context with words like 'river' and 'Russian', the sum of these two word vectors will be close to the vector of "Volga River". (this property will be examined in detail in later sections).

word model, the one we are going to examine now and which will serve as the reference for our comparative discussion of word embedding is the skip-gram model. The training goal of the skip-gram model is to arrive at vector representations of words that best predict a window of surrounding words. Formally the objective function is given by:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} log(p(w_{t+j}|w_t)) \quad (1)$$

Where c is the size of the context window. Here our aim is to maximize log probability of any context word given the current center word. For convenience with regard to taking the derivative during gradient descent, a popular probability measure has been the softmax function.

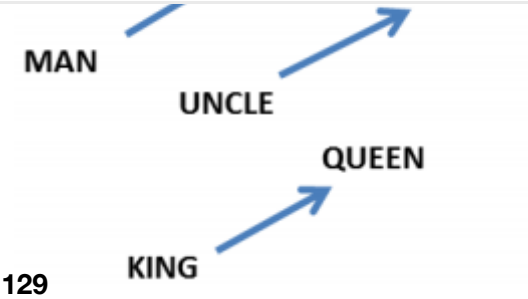
$$p(w_{t+1}|w_t) = \frac{exp(v_{w_o}\tilde{v}_{wI}^T)}{\sum_{k=1}^{V} exp(w_i^T \tilde{w}_k)} \quad (2)$$

where v_w and $\tilde{v_w}$ are the input and output vector representations of w and V is the number of words in the vocabulary. This expression is optimized using gradient descent. However, there is an obvious problem with this expression, every time we move over the context we have to recompute the normalization factor in the denominator for every word in W which can be on the order of $10^5 - 10^7$.

There are two major strategies that have been devised to attenuate the cost of computing the normalization term; negative sampling and hierarchical softmax. The idea in both cases is to compare the target word with a stochastic selection of contexts instead of all contexts. Since the denominator computes the similarity of all other contexts and the target word and thereby expresses that if two words are similar the expression in the nominator will be large relative to the denominator, it will be equally as large relative to a random selection of contexts.

How and Why does *Word2Vec* work?

Up until now we have not really tried to understand why *Word2Vec* works, in order to so let us look at how the aforementioned models are commonly presented. We can get a 'feel' for the word embedding space by illustrating them with t-SNE, a sophisticated method for dimensionality reduction:



65

t-SNE visualization of word embeddings: Left: Number Region, Right: Jobs Region. From Turian et al. (2010), original image (http://metaoptimize.s3.amazonaws.com/cw-embeddings-ACL2010/embeddings-mostcommon.EMBEDDING_SIZE=50.png)

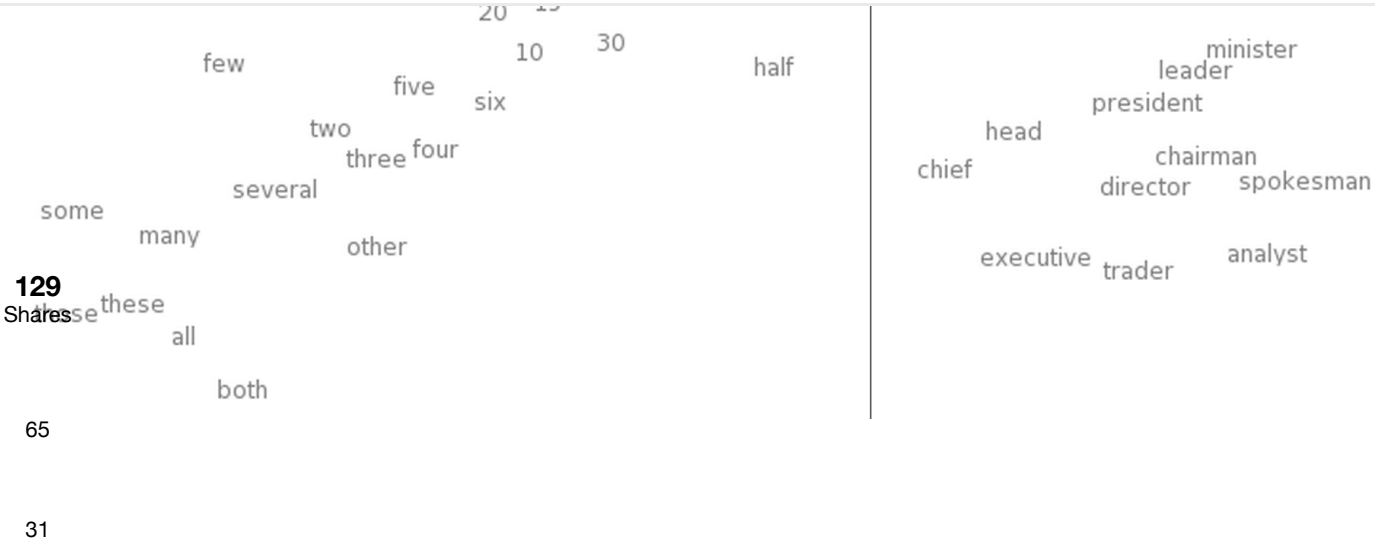
31

This makes intuitive sense to us because similar words are close to each other. After the above discussion it should come as no surprise that the neural network seem to produce vectors that reflect the similarity of the words they represent. If you switch a word for a synonym we would expect that it occurs in similar context, i.e. that it predicts similar words in its context window.

As we have seen word embeddings do even more, they have the remarkable property that analogies between words seem to be encoded in the difference between word vectors. For example there seems to be a constant male-female difference vector:

$$W(\text{" woman "}) - W(\text{" man "}) \cong W(\text{" aunt "}) - W(\text{"" uncle "})$$

$$W(\text{" woman "}) - W(\text{" man "}) \cong W(\text{" queen "}) - W(\text{"" king "})$$



From *Mikolov et al.* (2013a)

This may not seem entirely surprising considering how a word's context changes if it is replaced with its male or female counterpart. Most generally we write "he is the King", but "she is the Queen" along with this pronoun change we would expect other gender specific context discrepancies which are paralleled across words with different meanings. It turns out that there are a multitude of sophisticated relationships in several "dimensions of meaning" (Pennington et al. 2014). The difference vector of countries and their capitals, for instance are roughly the same as are the difference vectors of adjectives and their superlatives.

Considered from the perspective of Word2Vec while the *side effect* of encoding meaning seems to make intuitive sense the exact origins remain somewhat murky. In order to illuminate these, let us consider a derivation by Pennington, et al (2014) in their paper discussing the generation of word vectors from a matrix of word co-occurrence probabilities for a given context window. To understand the relevance of this derivation for our quest to understand the inner workings of Word2Vec, we briefly examine the paper's central idea.
The authors present a method centered around trying to make the dot product of the word vectors w_i and $\tilde{w_k}$ equal to the logarithm of their co-occurrence count X_{ij} (which denotes the number of times word j co-occurs with i).

$$w_i^T \tilde{w_k} + b_i + \tilde{b_k} = log(X_{ik}) \tag{3}$$

Where b_i and $\tilde{b_k}$ are merely bias terms simplifying downstream computation. In order to achieve this equality the authors employ a weighted least regression model, given by:

$$J = \sum_{ij=1}^{V} f(X_{ij})(w_i^T \tilde{w_k} + b_i + \tilde{b_k} - log(X_{ik}))^2 \tag{4}$$

There is no neural network or online update of word vectors and it is very easy to understand what this model is optimizing for. Let us now turn to the aforementioned derivation; we recall equation (2) from our discussion of the skip-gram model and follow the authors in their slight alteration of it to suit prior notation:

$$Q_{ij} = \frac{exp(w_i^T \tilde{w}_j)}{\sum_{k=1}^{V} exp(w_i^T \tilde{w}_k)} \tag{5}$$

The implied global objective function can be written as:

$$J = - \sum_{i \in corpus, j \in context(i)} log(Q_{ij}) \tag{6}$$

We are simply taking the log of the left-hand side and multiplying it by -1 in order to arrive at an objective function we can minimize. From here the authors follow a number of simplification and substitutions to finally arrive at:

$$J = \sum_{ij} X_i (w_i^T \tilde{w}_j - log X_{ij})^2 \tag{7}$$

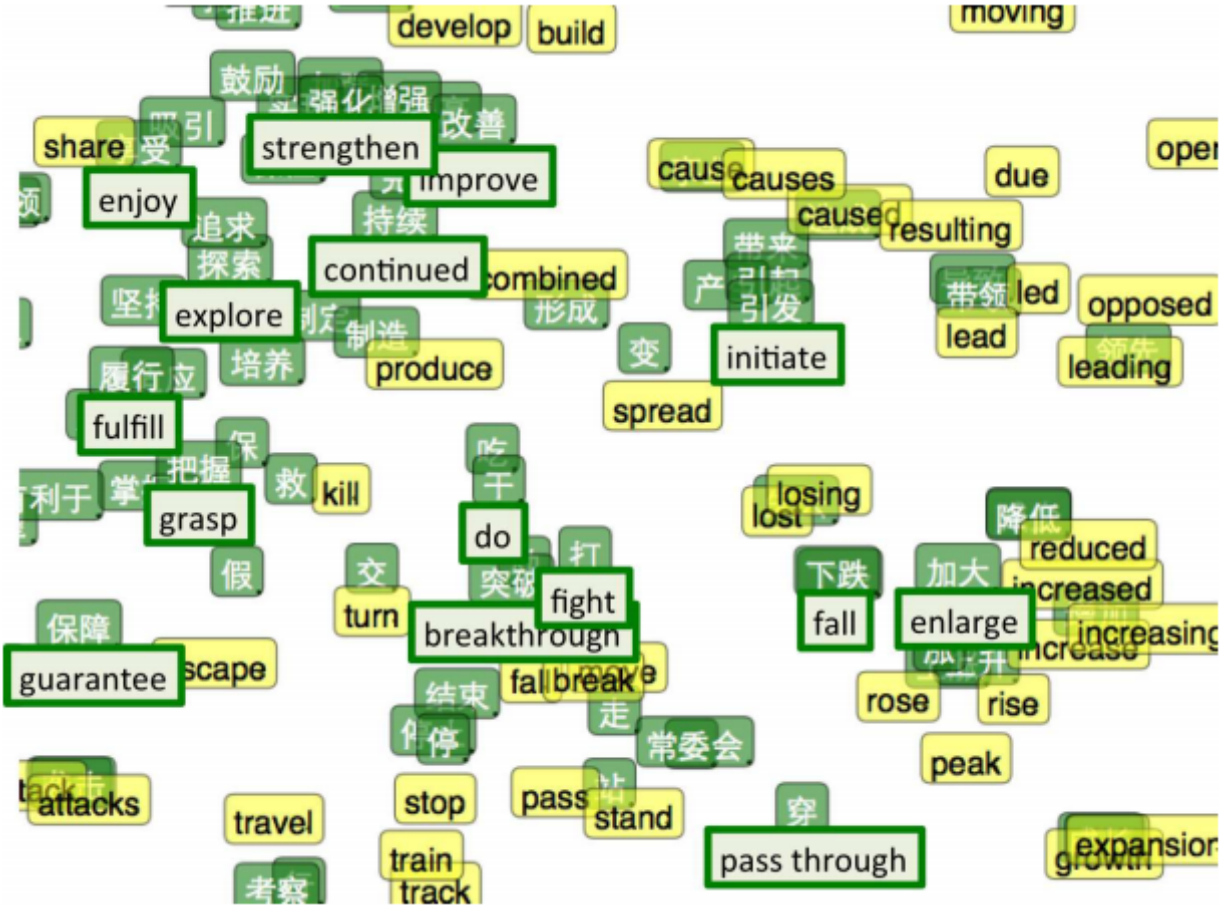
Which except for the weighting factor is identical to the model global optimization problem in equation (4).
So when viewed through the lens of corpus-wide, that is, global aggregation of words in their respective context windows Word2Vec reduces to the Glove model which explicitly aggregates global statistics of word co-occurrence. In this sense we have come full circle to the methods presented earlier that rely on matrix factorization (such as LSA). Where LSA uses SVD to find the best fitting subspace in terms of the co-variances of words across documents, the Glove model explicitly optimizes wordvectors to reflect the likelihood of their co-occurrence. The point, however, stands that Word2Vec is not an entirely novel approach to arrive at word vectors that capture semantic meaning in their substructure.

Applications

Let us now turn to some interesting new application of word embeddings derived from *Word2Vec*, both to other domains of machine learning and to concrete engineering problems.

To stay in the realm of NLP for a moment an interesting and natural application is in machine translation. Socher et al 2013 show that we can learn to embed words from two different languages in a single, shared space. In this case, we learn to embed English and Mandarin Chinese words in the same space.
In order to do so we simply train two word embedding models, W_{en} and W_{zh}, in a way similar to the

Not surprisingly observe that words with similar meaning end up close together, this is after all what the model optimized for. More remarkably, however, words whose translation was not known at the time of training the model ended up close together as well.



t-SNE visualization of bilingual word embedding. Green is Chinese, Yellow is English. (*Socher et al.*(2013a))

In light of our previous discussion this makes sense when we consider that word embedding seems to pull similar words close together, so hence by using a few translations as anchor point we create a mapping between English and Mandarin words. So we see that the essential property of word embeddings derived with *Word2Vec* helps us in other machine learning task by superimposing the structure of the word vector space on other domains (another conceivable domain might be image recognition).

are treated as words and other songs in a playlist as their surrounding context, depending on whether the playlists in question were genre specific the vocabulary encompassed a number of songs from that collection.

Now in order to recommend songs to a user one merely has to examine a neighborhood of the 'song embeddings' of songs the user already likes.

Similarly, we could recommend a user who to connect to in a social network setting by examining the graph of relationships, where the nodes represent words and a path through the graph represents a sentence, then nodes that occur in the context of similar other nodes, would be close together in the vector space. These examples show that the general applicability of *Word2Vec* based algorithms is very rich and that it behooves practitioners to examine their problem domain with respect to sequences of objects that occur in some meaningful context.

Literature

Bengio, Y. (2009). Learning deep architectures for AI. Foundations and Trends in Machine Learning.

Bengio, Y., Ducharme, R., Vincent, P. & Janvin, C. (2003). A neural probabilistic language model. JMLR, 3:1137–1155.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing By Latent Semantic Analysis. Journal of the American Society For Information Science , 41 , 391-407.

Levy, O. & Goldberg, Y. (2014). Dependency based word embeddings. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Baltimore, Maryland, USA, June. Association for Computational Linguistics.

Dennewitz, M., playlist-to-vec, (2015), GitHub repository, https://github.com/mattdennewitz/playlist-to-vec (https://github.com/mattdennewitz/playlist-to-vec)

Mikolov, T., Chen K., Corrado G., & Dean J. 2013a. Efficient Estimation of Word Representations in Vector Space. In ICLR WorkshopPapers.

Mikolov, T., Sutskever, S., Chen, K., Corrado, G., & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In NIPS, pages 3111–3119.

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. Empricial Methods in Natural Language Processing (EMNLP)

Made by Jan Bussieck (https://twitter.com/buss_jan) and Malte Baumann (https://twitter.com/codingdivision)

Deep Learning Weekly aims at being the premier news aggregator for all things deep learning. We keep tabs on major developments in industry be they new technologies, companies, product offerings or acquisitions so you don't have to. In addition our 'Learning' section features new content that makes difficult to understand areas in deep learning accessible to a wider audience and our 'Papers & Publications' section brings you the most exicting new research. If you have any thoughts or ideas how we might improve this newsletter we are interested in hearing them. Thank you in advance!