Word2Vec Technology Review

Introduction

Natural language processing (NLP) is a way for computers to understand, analyze and derive meaning from human language in a smart and useful manner. The important function of language processing task is to create word representation that would capture their meaning, semantic relationships, and different types of contexts they are used. **One-hot vector** is a very straight forward way to represent word but they are very **inefficient** and they are **independent of any context**.

To overcome such limitation, representation of words is encoded such a way that similarities can be derived from their representation. **Distributional similarity-based word representation** is derived from the context in which the words appear.

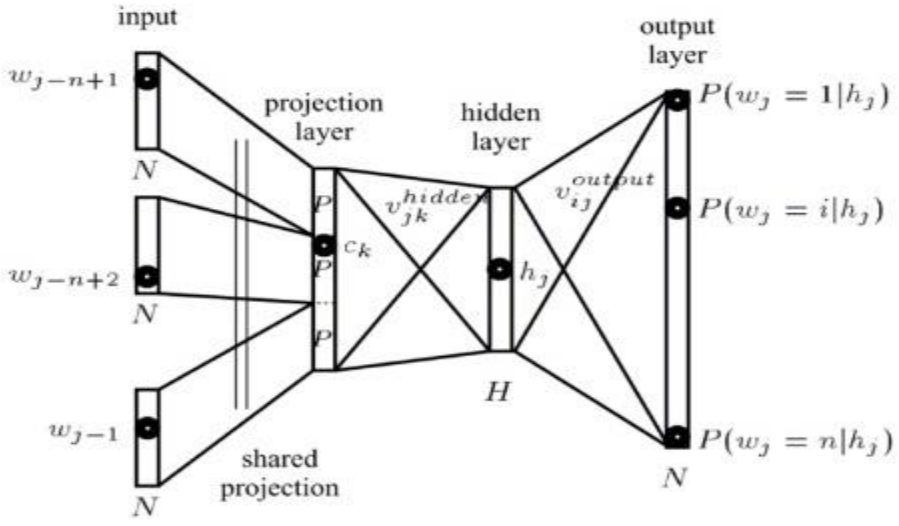
The **word2Vec** model builds simple and scalable model which can be run on large set of corpus of text to build **good distributional similarity-based word representation**. The Distributed Representations of Words and Phrases and their Compositionality paper presents several extensions of the original Skip-gram model to **faster training and better vector representation.**

Word Embedding

A word embedding is a learned representation for text where words that have the same meaning have a similar representation. Word embeddings are nothing but the individual word representation as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles **a Neural Network**.

Word embedding can be learnt with multiple techniques. **Embedding Layer and Word2Vec** are the most popular techniques for learning word embedding.

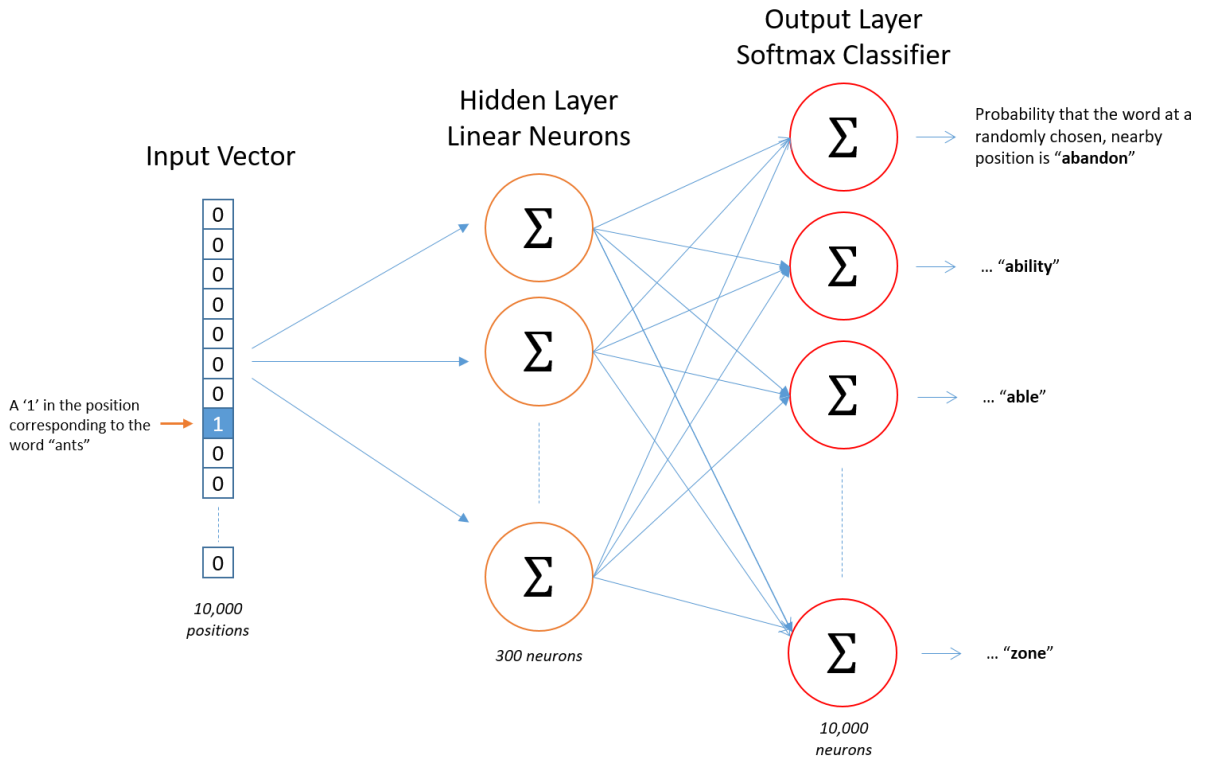
Neural Networks and Embedding Layer



Using feed-forward neural network, word embedding is derived by feeding words from a vocabulary, embed the word input into vectors in lower dimensional space using hidden layer, and fine-tuning weights through back propagation. The fine-tuned weights are the word embedding.

This approach of learning an embedded layer requires lot of training data and can be slow.

Skip-gram Model



The skip-gram neural network model uses single hidden layer. Using this simple neural network, weights of the hidden layer is learnt. These weights are nothing but the word vectors. The skip-gram neural network model is training by inputting specific word in the middle of a sentence (the input word) and nearby context word as the output. The network is going to tell us about the probability for every word in our vocabulary of being the "nearby word" that we chose.

In the above example, word like "ants" are represented as one-hot vector. This vector will have 10,000 components and 300 features (hidden layer represented with 10,000 rows and 300 columns). The output of the network contains the probability of the nearby word in the vocabulary.

The basic skip-gram model needs huge of amount of training data and training of such model could be slow. The authors of Word2Vec addressed this by following approaches:1. **Hierarchical Softmax** 2. **Negative Sampling** 3. **Subsampling of Frequent Words**

Hierarchical Softmax

Hierarchical Softmax essentially replaces the flat softmax layer with a hierarchical layer that has the words as leaves. This allows us to decompose calculating the probability of one word into a sequence of probability calculations, which saves us from having to calculate the expensive normalization over all words. Replacing a softmax layer with H-Softmax can yield speedups for word prediction tasks of at least 50x.

Negative Sampling

The size of our word vocabulary means that our skip-gram neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples! Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them.

Subsampling Frequent Words

Word2Vec implements a "subsampling" scheme to address an issue with common word such as "the". The common words will have many more samples that we need to learn a good vector.

When looking at word pairs, (For example, "fox", "the") doesn't tell us much about the meaning of "fox". "the" appears in the context of pretty much every word. The probability that we cut the word is related to the word's frequency.

Conclusions

The Distributed Representations of Words and Phrases and their Compositionality provides performance analysis of large data set (an internal Google data set with one billion words) and shows that how skip gram performance can be improved by Negative Sampling and subsampling frequent words.

The word vectors can be combined using just simple vector addition. Also the word2vec presents an approach that phrases can be represented with a single token. Paper states that such approach provides powerful way how to represent longer pieces of text, while having minimal computational complexity.

References

1. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013
2. http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling
3. https://aylien.com/blog/overview-word-embeddings-history-word2vec-cbow-glove
4. https://machinelearningmastery.com/what-are-word-embeddings/
5. Distributed Representations of Words and Phrases and their Compositionality [https://arxiv.org/abs/1310.4546]