Word2Vec in Natural Language Processing

Nuha Mohammed

April 2020

1 Introduction

Natural Language Processing (NLP) is a subset of machine learning which is directed toward reading and deriving meaning from human languages. NLP has a wide array of applications from spell checking to machine translation to semantic analysis.

2 Word Vector Representation

An important part of NLP is how we represent words as input to any model—we can't use words directly because the algorithm needs a way of determining similarity and difference between words.

2.1 One-hot Encoding

One-hot encoding is a common method used to turn categorical data into numerical data. In the context of NLP, the one-hot encoded vector is the same length as the word dictionary, with each word mapped to a unique index.

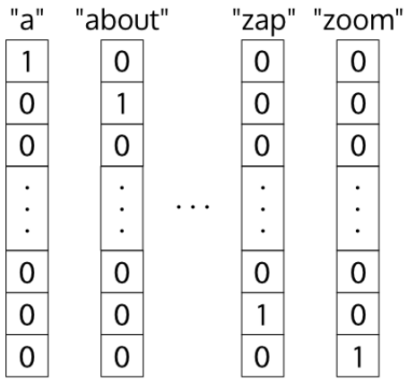

Figure 1: One-hot encoded vectors

The easiest way to keep track of each word-index pair is to pair the words to an index in alphabetical order. Then, in each word vector, a 1 is placed at the index of that word and a 0 is placed at all other indices.

2.2 Continuous Embedding Vectors

Although one-hot encoding is the most intuitive numerical way to represent words, it fails to capture the contextual meaning of a word as the distance between one-hot encoded vectors for any two words is always the same. For this reason, we use continuous vector embeddings which can capture different semantic meanings/features of a word in the context of the sentence or passage.

3 Word2Vec

Word2Vec models create continuous embedding vectors which represent words within a large text in an n-dimensional vector space.

In the following example, a Word2Vec model with 3 features is created, and for each feature/category, the word is scored for its likelihood of belonging to that category.

```
vec(Daisy)=[3.92, 1.34, -3.82]

vec(Rose)=[-0.93, 1.220, 1.82]

vec(Donald)=[4.09, -0.58, 2.01]
```

The embedding vectors show that "Daisy" and "Donald" are closest together on the first dimension, so the first dimension probably pertains to a feature that both "Daisy" and "Donald" share that "Rose" does not relate to and so on.

3.1 Data Setup

In order to set up the data, you need to identify context words for each center word. Based on the center word, each neighboring word within a selected window size is a context word and is paired with the center word to create the training samples.

The window size is a hyper parameter which can be used to tune the model. Although some models use the entirety of a sentence as the context, a larger window size does not necessarily result in a more accurate model. Larger window sizes often capture more information related to the topic/main idea (in context), while smaller window sizes capture more information about the center word itself. The window size is equal to 2 in the following diagram.

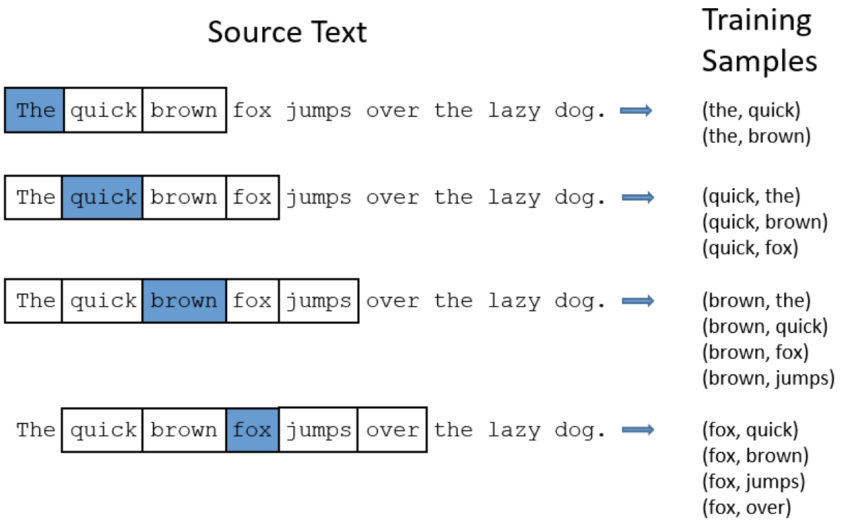

Figure 2: Depiction of how context-word pairs are created

3.2 Algorithms

Word2Vec includes two possible algorithms: Skip-Gram and Continuous Bag-of-Words (CBOW). CBOW works by trying to predict the center word from its neighboring context words, while Skip-Gram does the opposite and attempts to predict the context words, given the center word.

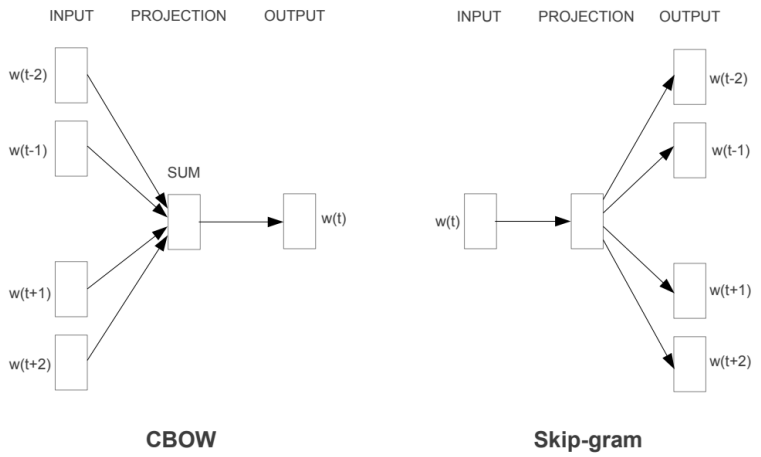

Figure 3: CBOW vs. Skip-Gram Model

Whether Skip-Gram or CBOW works better depends on the type and amount of data. However, I am only going to go in detail on the Skip-Gram model in this lecture as both are conceptually similar.

3.3 Training

In Skip-Gram models, the input is a one-hot encoded vector of the center word. Each neuron within the output layer gives the probability that the word at that position is a context word of that center word.

Since the output layer performs softmax, it generates values between 0 and 1 and all values in the layer add up to 1, giving us probabilities. We want our generated probability vector to match the actual probabilities which are represented with the one-hot encoded vector of the word.

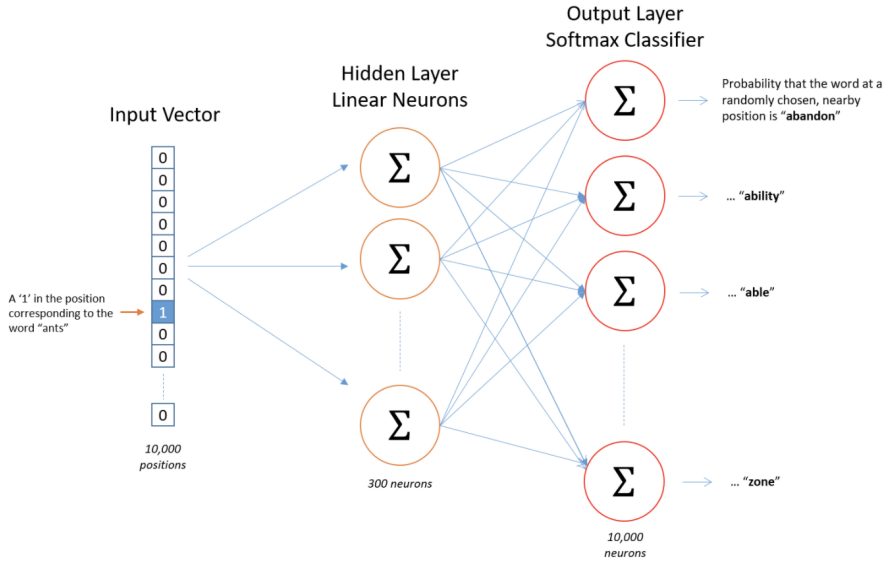

Figure 4: Neural network architecture in Skip-Gram

. The chart depicts the neural network structure for the context word "ant" among a dictionary of 10,000 words. The input vector is a one-hot encoded vector so only the position corresponding to the word "ant" contains a 1 and the other positions contain a 0, with a total of 10,000 positions.

Note that Word2Vec is not used for the task it is trained on. The model is not ultimately used to predict the output (which is already known) but to create the word embedding vector which in other words is the weight matrix W, as shown.

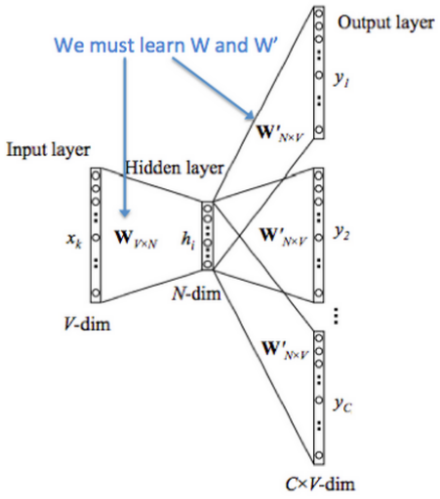

Figure 5: Weights of the hidden layer in the neural network

W and W' are the weight matrices between the input and hidden layer and the output and hidden layer, respectively. The model seeks to optimize the weight matrices by maximizing the probability of correctly predicting the context words.

The weight matrices are passed into the cost function as variables and optimized through gradient descent. Just as any other neural network, the model aims to minimize its cost function:

$$h = W^T x,$$
$$u_c = W'^T x = W'^T W^T x,$$
$$y_c = Softmax(u_c)$$
$$L = -\log P(w_{c,1}, w_{c,2}, ..., w_c | w_0) = -\log \prod_{c=1}^{C} P(w_{c,i} | w_0)$$

After training, each row in the input weight matrix (W) consists of a word-embedding vector for each word:

$$W = \begin{array}{c} \text{always} \\ \text{kite} \\ \text{should} \\ \text{...} \\ \text{there} \\ \text{who} \end{array} \begin{bmatrix} 0.5 & 0.3 & 7 \\ 8 & -0.9 & 2.2 \\ 0.3 & 5 & 2.1 \\ & \text{...} & \\ 0.6 & 7 & 3.2 \\ 2 & 0.5 & 4 \end{bmatrix}$$

The word embeddings serve as a foundation of a lot of NLP tasks such as topic modeling and analyzing meaning from passages.

With the word embedding themselves, you can discover underlying patterns in data. One interesting application is in Proteomics and Genomics— BioVectors encode n-grams in biological sequences such as DNA, RNA, and protein sequences. Using techniques such as Skip-Gram modeling, scientists have been able to group biological sequences based on underlying biochemical properties and interactions.

4 Code Sample

Checkout this implementation of Word2Vec with the Skip Gram model:
`https://github.com/DerekChia/word2vec_numpy/blob/master/wordtovec.py`

5 Further Exploration

For a large amount of data, the Word2Vec becomes inefficient as it will contain a huge number of weights. Gradient descent performed on that network will be extremely slow and it will be hard to avoid over-fitting. Therefore, the authors of Word2Vec also introduced a concept called Negative Sampling which takes care of this issue. I encourage you to look into their second paper to find out more about optimization.

6 References

The information and images I used in these lectures came from multiple sources. Credits go to their respective owners.

- Stanford CS224d Lecture: Deep Learning for NLP
- Word2Vec Paper: Efficient Estimation of Word Representations in Vector Space
- Chris McCormick's Word2Vec Tutorial
- Word2Vec (Skip-Gram model) Explained - Medium Article
- Eric Kim's Skip-Gram Modeling Tutorial
- ML Club RNN Lecture