Word2vec - NLP Lecture - 5

Word2vec | CBOW and Skip-gram

INSTRUCTOR: Nitish Sir (CampusX)

Author: Harshavardhan BR

Word Embeddings

In natural language processing, word embedding is a term used for the representation of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning.

Types of word embeddings:

- Frequency based: based on the frequency of the words
  - BOW
  - Tf-Idf
  - Glove (Matrix factorization)
- Prediction based: based on the prediction of some algorithm
  - Word2vec

Word2vec

Word2vec is a technique used in natural language processing to create vector representations of words. These vectors can be used to measure the similarity between words, which can be used for tasks such as text classification, machine translation, and question answering.

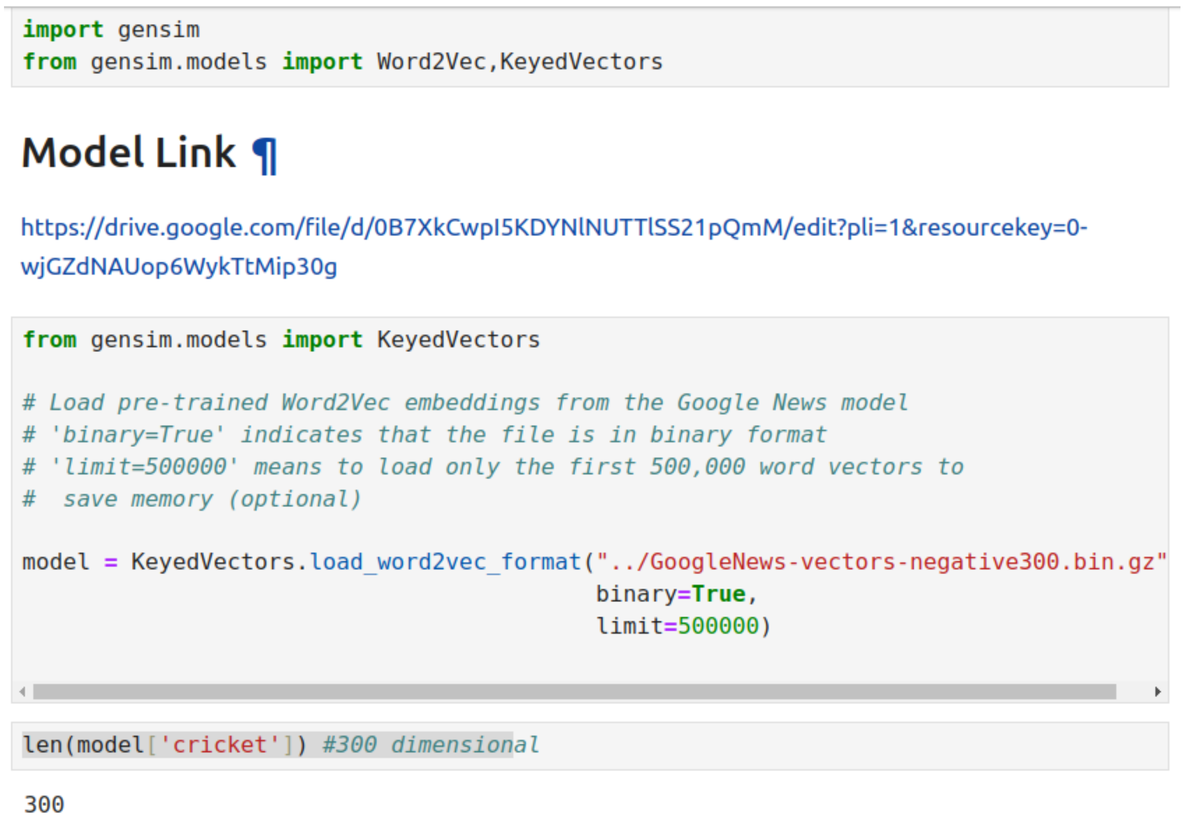
For example, the word2vec model might learn that the words "cat" and "dog" are similar because they are both animals. This information could then be used to classify a sentence as being about animals, or to translate the sentence into another language.

Benefits of Word2vec over other techniques such as Tf-Idf, BOW etc

- Word2Vec captures semantic meaning, while bag of words and n-grams only consider word occurrence.
- Word2Vec handles unknown words better than bag of words and n-grams.
- Word2Vec provides computationally efficient dense vector representations.
- Word2Vec captures word relationships for tasks like similarity and analogy.
- Word2Vec can be pre trained and transferred to specific tasks.
- Word2Vec improves performance in various NLP tasks
- Word2Vec outputs a dense vector

Implementation using a pre-trained model

We will use the pre-trained weights of word2vec that was trained on Google News corpus containing 2 billion words. This model consists of 300-dimensional vectors for 3 million words and phrases.



```
import gensim
from gensim.models import Word2Vec,KeyedVectors
```

Model Link ¶

https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?pli=1&resourcekey=0-wjGZdNAUop6WykTtMip30g

```
from gensim.models import KeyedVectors

# Load pre-trained Word2Vec embeddings from the Google News model
# 'binary=True' indicates that the file is in binary format
# 'limit=500000' means to load only the first 500,000 word vectors to
#  save memory (optional)

model = KeyedVectors.load_word2vec_format("../GoogleNews-vectors-negative300.bin.gz"
                                          binary=True,
                                          limit=500000)
```

```
len(model['cricket']) #300 dimensional
```

```
300
```

Retrieve the word vector for the word "cricket" from the model and then return the the vector, which is of 300 dimensions.

```
model['cricket']
```

```
array([-3.67187500e-01, -1.21582031e-01,  2.85156250e-01,  8.15429688e-02,
        3.19824219e-02, -3.19824219e-02,  1.34765625e-01, -2.73437500e-01,
        9.46044922e-03, -1.07421875e-01,  2.48046875e-01, -6.05468750e-01,
        5.02929688e-02,  2.98828125e-01,  9.57031250e-02,  1.39648438e-01,
       -5.41992188e-02,  2.91015625e-01,  2.85156250e-01,  1.51367188e-01,
       -2.89062500e-01, -3.46679688e-02,  1.81884766e-02, -3.92578125e-01,
        2.46093750e-01,  2.51953125e-01, -9.86328125e-02,  3.22265625e-01,
        4.49218750e-01, -1.36718750e-01, -2.34375000e-01,  4.12597656e-02,
       -2.15820312e-01,  1.69921875e-01,  2.56347656e-02,  1.50146484e-02,
       -3.75976562e-02,  6.95800781e-03,  4.00390625e-01,  2.09960938e-01,
        1.17675781e-01, -4.19921875e-02,  2.34375000e-01,  2.03125000e-01,
       -1.86523438e-01, -2.46093750e-01,  3.12500000e-01, -2.59765625e-01,
       -1.06933594e-01,  1.04003906e-01, -1.79687500e-01,  5.71289062e-02,
       -7.41577148e-03, -5.59082031e-02,  7.61718750e-02, -4.14062500e-01,
       -3.65234375e-01, -3.35937500e-01, -1.54296875e-01, -2.39257812e-01,
       -3.73046875e-01,  2.27355957e-03, -3.51562500e-01,  8.64257812e-02,
        1.26953125e-01,  2.21679688e-01, -9.86328125e-02,  1.08886719e-01,
        3.65234375e-01, -5.66406250e-02,  5.66406250e-02, -1.09375000e-01,
```

```
model.most_similar('man')
```

```
[('woman', 0.7664012908935547),
 ('boy', 0.6824871301651001),
 ('teenager', 0.6586930155754089),
 ('teenage_girl', 0.6147903203964233),
 ('girl', 0.5921714305877686),
 ('robber', 0.5585119128227234),
 ('Robbery_suspect', 0.5584409832954407),
 ('teen_ager', 0.5549196600914001),
 ('men', 0.5489763021469116),
```

```
model.similarity('man','woman')
```

```
0.76640123
```

```
model.similarity('man','Python')
```

```
0.035492986
```

```
model.doesnt_match(['PHP','java','monkey'])
```

```
'monkey'
```

```
vec = model['king'] - model['man'] + model['woman']
model.most_similar([vec])
```

```
[('king', 0.8449392318725586),
 ('queen', 0.7300517559051514),
 ('monarch', 0.645466148853302),
 ('princess', 0.6156251430511475),
 ('crown_prince', 0.5818676352500916),
 ('prince', 0.5777117609977722),
 ('kings', 0.5613663792610168),
 ('sultan', 0.5376775860786438),
 ('queens', 0.5289887189865112),
 ('ruler', 0.5247419476509094)]
```

```
vec = model['INR'] - model ['India'] + model['England']
model.most_similar([vec])
```

```
[('INR', 0.6442341208457947),
 ('GBP', 0.5040826797485352),
 ('England', 0.44649264216423035),
 ('£', 0.43340998888015747),
 ('Â_£', 0.4307197630405426),
 ('£_#.##m', 0.42561301589012146),
 ('Pounds_Sterling', 0.42512619495391846),
 ('GBP##', 0.42464491724967957),
 ('stg', 0.42324796319007874),
 ('£_#.###m', 0.4201711118221283)]
```

Word2Vec Intuition

The underlying assumption of Word2Vec is that two words sharing similar contexts also share a similar meaning and consequently a similar vector representation from the model. Word2Vec aims to represent words as dense vectors in a continuous space, capturing semantic relationships based on their contextual usage in text. It uses models like CBOW and Skip-gram to learn word embeddings, allowing comparisons, similarities, and meaningful operations on words, which benefits various natural language processing tasks.

Types of Word2Vec

1. **CBOW (Continuous bag of words)**

   Continuous Bag of Words (CBOW) is a technique in Word2Vec that tries to predict a target word by looking at the words around it in a sentence. It takes a bunch of nearby words and tries to guess the missing word in the middle. CBOW is like a "word guessing game" where it learns to predict a word based on its context. This way, it creates meaningful word representations that can help us understand how words are related to each other in a text.

2. **Skip-gram**

   Skip-gram is another technique in Word2Vec that works in the opposite way of CBOW. Instead of guessing the missing word from the context, Skip-gram takes a single word as input and tries to predict the words that are likely to appear around it in a sentence. It's like playing a "word association" game, where it learns to guess the related words based on a given word. Skip-gram helps create meaningful word representations that capture the relationships between words in a text, which is useful for various natural language processing tasks.

Note: If you are working with small amount of data then CBOW provides better results