Word to vector

Name: Wadie Bishoy
Id: 5074

Introduction

Word embeddings are basically a form of word representation that bridges the human understanding of language to that of a machine. Word embeddings are distributed representations of text in an n-dimensional space. These are essential for solving most NLP problems.

The importance of word embeddings in the field of deep learning becomes evident by looking at the number of researches in the field. One such research in the field of word embeddings conducted by Google led to the development of a group of related algorithms commonly referred to as Word2Vec.

Word2Vec one of the most used forms of word embedding is described by Wikipedia as:
“Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.”

Supervised Text Classification:

Classification is the task of choosing the correct class label for a given input. In basic classification tasks, each input is considered in isolation from all other inputs, and the set of labels is defined in advance. Some examples of classification tasks are:

- Deciding whether an email is spam or not.
- Deciding what the topic of a news article is, from a fixed list of topic areas such as "sports," "technology," and "politics."
- Deciding whether a given occurrence of the word bank is used to refer to a river bank, a financial institution, the act of tilting to the side, or the act of depositing something in a financial institution.

The first step in creating a classifier is deciding what features of the input are relevant, and how to encode those features.

Usually, we use **CountVectorizer** or **TfidfVectorizer** from **sklearn.feature_extraction.text** , because they are the most common & easy open libraries in sklearn , and we already use CountVectorizer as a baseline feature extractor to other classifiers .

However, **Word2vec** can be used as a feature extractor algorithm to enhance the classification problem.

Word2Vec in practice

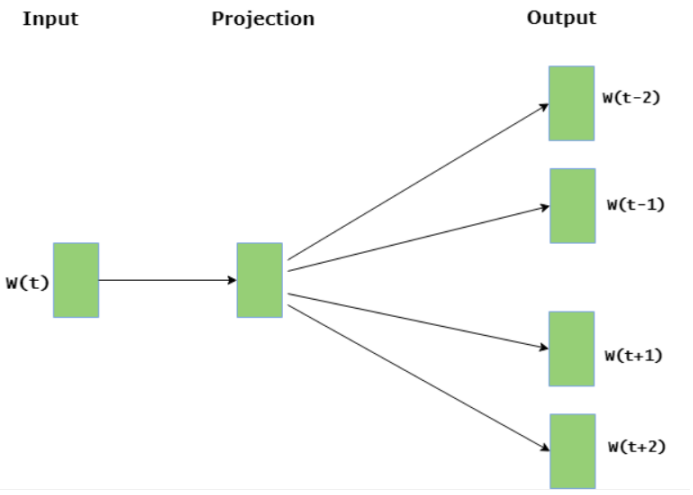
Word2Vec returns some astonishing results. Word2Vec's ability to maintain semantic relation is reflected by a classic example where if you have a vector for the word "King" and you remove the vector represented by the word "Man" from the "King" and add "Women" to it, you get a vector which is close to the "Queen" vector. This relation is commonly represented as:

```
King - Man + Women = Queen
```

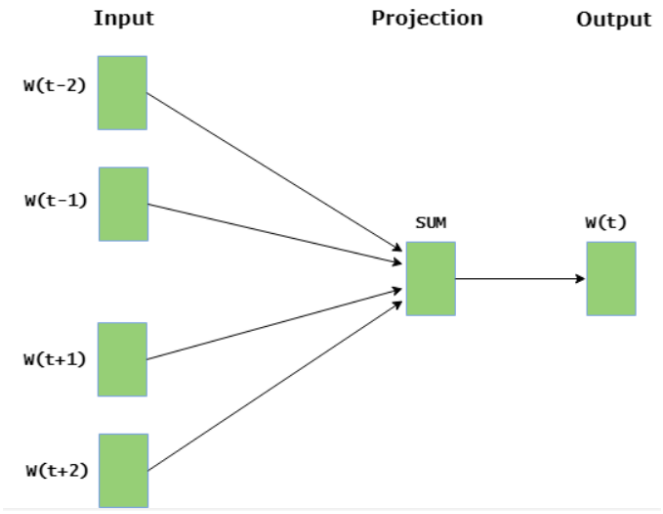
Word2Vec model comes in two flavors:

- Skip Gram Model
- Continuous Bag of Words Model (CBOW).

Skip gram : predicts the surrounding context words within specific window given current word. The input layer contains the current word and the output layer contains the context words. The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer. For instance, given a sentence "I love to dance in the rain", the skip gram model will predict "love" and "dance" given the word "to" as input.



CBOW model : predicts the current word given context words within specific window. The input layer contains the context words and the output layer contains the current word. The hidden layer contains the number of dimensions in which we want to represent current word present at the output layer. So, on the contrary, the CBOW model will predict "to", if the context words "love" and "dance" are fed as input to the model.



Advantages of Word2Vec

Word2Vec has several advantages over other word embeddings. Word2Vec retains the semantic meaning of different words in a document. The context information is not lost. Another great advantage of Word2Vec approach is that the size of the embedding vector is very small. Each dimension in the embedding vector contains information about one aspect of the word. We do not need huge sparse vectors, unlike the bag of words and TF-IDF approaches.

Implementing Word2vec

In Python, we can use the implementation of word2vec from the **gensim** package.

Although Word2Vec does not require graphics processing units (GPUs) like many deep learning algorithms, it is compute intensive. Both Google's version and the Python version rely on multi-threading (running multiple processes in parallel on your computer to save time.

Word2Vec expects single sentences, each one as a list of words. In other words, the input format is a list of lists.

Example of tokenized sentence:

```
['science', 'includes', 'such', 'diverse', 'fields', 'as', 'astronomy', 'biology', 'computer', 'sciences', 'geology', 'logic', 'physics', 'ch
emistry', 'and', 'mathematics', 'link']
```

With the list of nicely parsed sentences, we can train the model. However, There are a number of parameter choices that affect the run time and the quality of the final model that is produced, such as :

- **Architecture**: Architecture options are skip-gram (default) or continuous bag of words. We found that skip-gram was very slightly slower but produced better results.
- **Training algorithm**: Hierarchical softmax (default) or negative sampling. For us, we use the default one.
- **Downsampling of frequent words**: The Google documentation recommends values between .00001 and .001 (we keep it as the default value) .
- **Word vector dimensionality**: More features result in longer runtimes, and often, but not always, result in better models. Reasonable values can be in the tens to hundreds; we used 300.
- **Context / window size**: How many words of context should the training algorithm take into account? 10 seems to work well for hierarchical softmax.
- **Worker threads**: Number of parallel processes to run. This is computer-specific, but between 4 and 6 should work on most systems , we use 4.
- **Minimum word count**: This helps limit the size of the vocabulary to meaningful words. Any word that does not occur at least this many times across all documents is ignored. Higher values also help limit run time. We used 10 .

Exploring the Model Results

With the implementation of the model some interesting features occurred :

doesnt_match:

The "doesnt_match" function will try to deduce which word in a set is most dissimilar from the others

Example :

```
model.doesnt_match("man woman child kitchen".split())
```

```
'kitchen'
```

```
model.doesnt_match("france england germany berlin".split())
```

```
'berlin'
```

most_similar:

the "most_similar" function to get insight into the model's word clusters

Example:

```
model.most_similar("war")
```

```
[('notably', 0.9967514276504517),
 ('new', 0.9945055842399597),
 ('involvement', 0.9932841062545776),
 ('president', 0.9931340217590332),
 ('department', 0.9930415749549866),
 ('congress', 0.9928803443908691),
 ('wake', 0.992829442024231),
 ('late', 0.9923601150512695),
 ('election', 0.9921090602874756),
 ('early', 0.9914227724075317)]
```

index2word:

Index2word is a list that contains the names of the words in the model's vocabulary, (Converted to a set, for speed)

Example:

```
set(model.wv.index2word)
```

```
{'below',
 'sherman',
 'molecule',
 'atp',
 'hominoids',
 'conformational',
 'probability',
 'carrier'
```

Some of Results :

```
--> Using Word2vec
random forest (just for testing)..
              precision    recall  f1-score   support

           0       0.78      0.92      0.84       579
           1       0.75      0.47      0.57       286

    accuracy                           0.77       865
   macro avg       0.76      0.69      0.71       865
weighted avg       0.77      0.77      0.76       865
```

```
K-nearest neighbours..
              precision    recall  f1-score   support

           0       0.77      0.77      0.77       579
           1       0.54      0.54      0.54       286

    accuracy                           0.69       865
   macro avg       0.65      0.66      0.65       865
weighted avg       0.69      0.69      0.69       865
```

```
decision tree..
              precision    recall  f1-score   support

           0       0.77      0.75      0.76       579
           1       0.52      0.55      0.53       286

    accuracy                           0.68       865
   macro avg       0.64      0.65      0.65       865
weighted avg       0.69      0.68      0.68       865
```