IMPLEMENTATION OF WORD2VEC CBOW MODEL

Submitted by,
Arunkumar A

Abstract

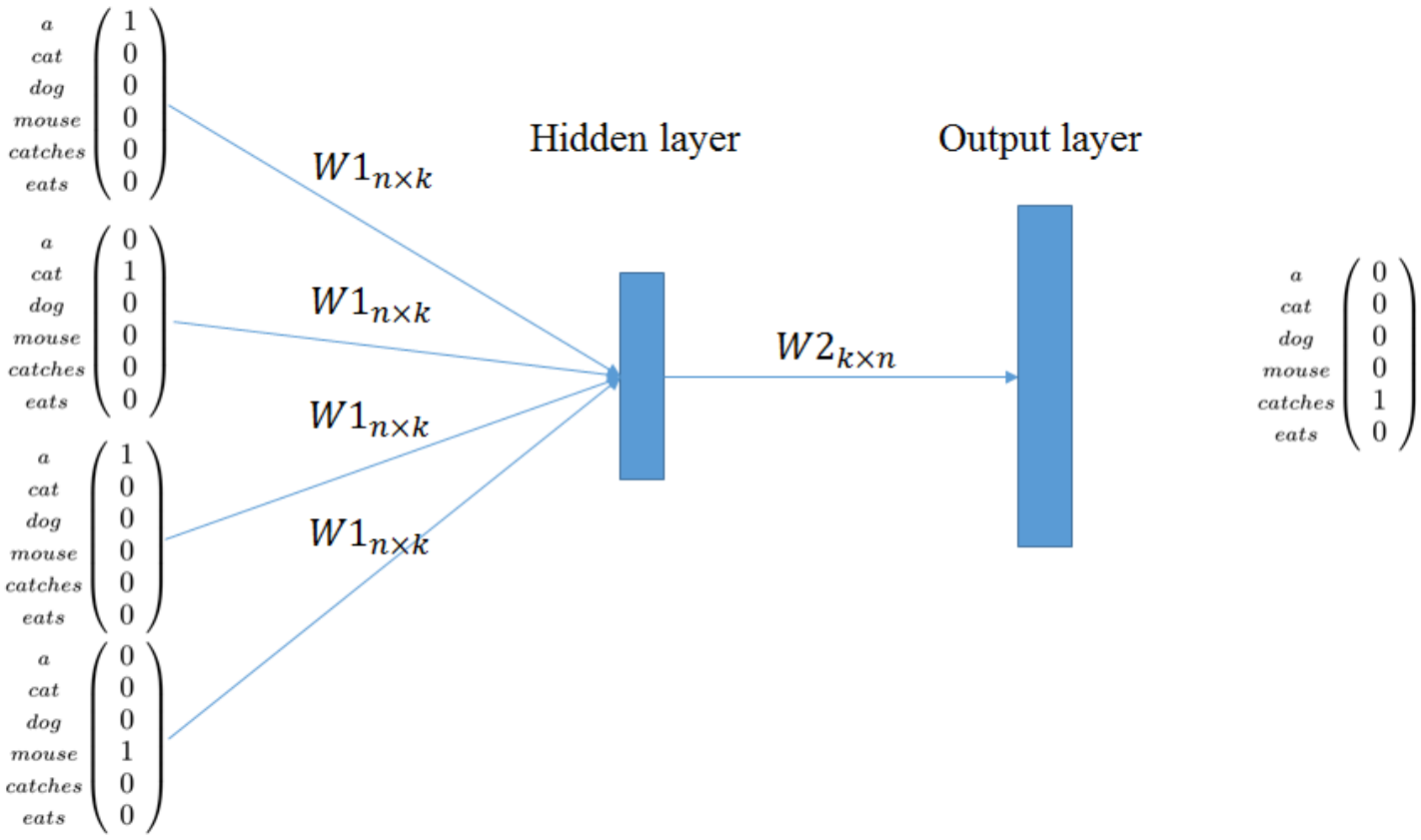
Word embedding is one of the most important techniques in natural language processing (NLP), where words are mapped to vectors of real numbers. Word embedding is capable of capturing the meaning of a word in a document, and relation with other words. It has been widely used for recommendation systems and text classification. Word embedding algorithms like word2vec and GloVe are key to the state-of-the-art results achieved by natural language processing problems like machine translation. In this project, we implement a genism Word2Vec CBOW model with an example of generating word embeddings for Hotel reviews.

Introduction

Word2Vec is touted as one of the biggest, most recent breakthrough in the field of Natural Language Processing (NLP). The concept is simple, elegant and (relatively) easy to grasp. A quick Google search returns multiple results on how to use them with standard libraries such as Gensim and TensorFlow.

With the rapid increase in internet resources, Natural Language Processing (NLP), including document clustering has become a widely researched topic for many decades. Many models have been built to handle natural language processing, such as N-gram and Hidden Markov Model. All of these are used to represent documents via extracting the semantic features. But these models suffer from high dimension problem.

Word2Vec is a new method which embed each word into a low dimensional continuous vector space in which words can be easily compared for similarity. It is capable of capturing context of a word in a document, finding semantic and syntactic similarity and relation with other words etc.



Background

The project is an implementation of Word2Vec CBOW model on Travel Industry dataset. The project helps explain the basic notion of NLP in a bottom-up approach.

Model	Relation of *w*, *c*	Representation of *c*
Skip-gram[4]	*c* predicts *w*	One of *c*
Continuous bag of words (CBOW)[4]	*c* predicts *w*	Average
Order	*c* predicts *w*	Concatenation
Log-bilinear language (LBL) model[5]	*c* predicts *w*	Compositionality
Neural network language model (NNLM)[1]	*c* predicts *w*	Compositionality
Collobert and Weston (C&W)[2]	Scores *w*, *c*	Compositionality

Preprocessing

Texts in the real world are highly subjective to deal with. It so difficult to categorize sentences just by reading the text. It is understood that language is highly inconsistent, even in the most formal settings. In this

section we will cover the standard NLP tool-set that is used to transform text from a raw sequence of characters to a form that is cleaner and easier to deal with computationally.

Tokenization

Tokenization is the task of splitting the text into more meaningful character sequence groups, usually words or sentences. It is almost always the first step in any NLP pipeline. Characters are hard to interpret by a computer on their own, but words are mostly self contained semantic units. It is a comfortable level of abstraction to work at, a lot of NLP operations act directly on words. A naïve approach to tokenization would be simply to split by spaces and remove any punctuation. It is a good baseline, but not ideal.

Parsing

After isolating and cleaning words, it is common to add yet another layer of processing to be able to work at higher abstraction levels. Parsing encompasses a set of annotation tasks to identify the role of each word with respect to its context in some text, usually a sentence. Parsing is usually the core of NLP tasks that requires deep understanding.

Embeddings

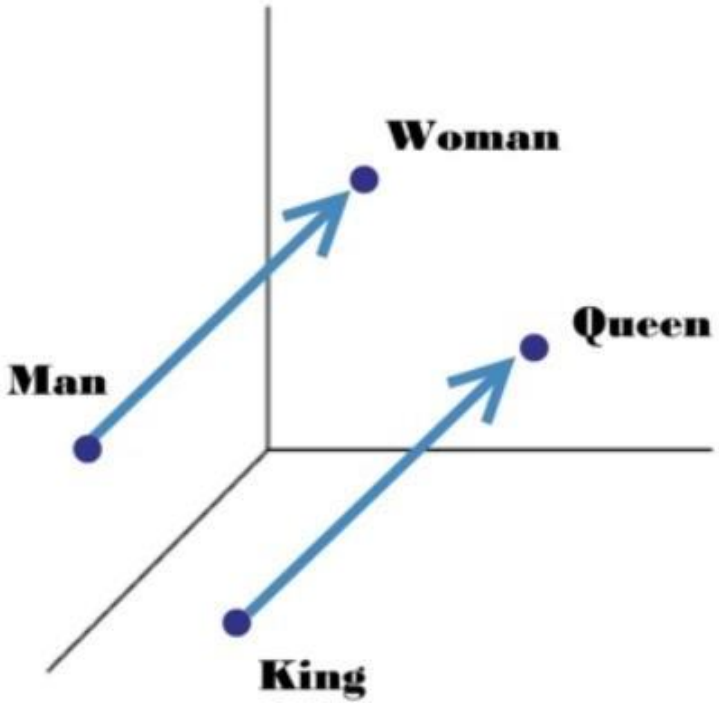
Embeddings are numerical representations of semantic units. The most common form of embeddings are word embeddings. Word embeddings assume that there is no ambiguity. So they end up capturing multiple senses in the same vector representation, which is not ideal. Sense embeddings are much harder to train where word embeddings can be efficiently learned just by scanning big corpora. There are sense embedding algorithms that utilize some form of clustering to distinguish senses, either during or after training.

Word Embeddings

Word embeddings are vector representation of the meaning of words. In practice, this usually means that word embeddings are placed in a high dimensional space where the embeddings of similar or related

words are close to each other and different word embeddings are placed far from each other. Word embeddings also acquire more complex geometric structures as a side effect of some algorithms. A typical example for this is real world analogies that can be discovered using simple vector arithmetic:

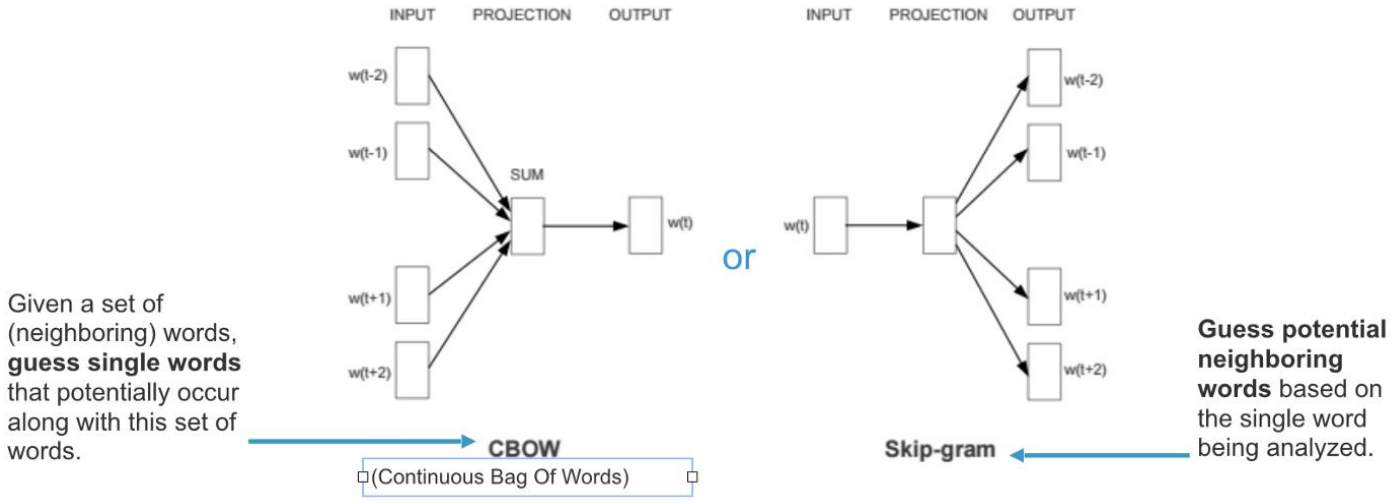
king - man + woman = queen



Word2Vec

Word2vec is one of the most popular technique to learn word embeddings using a two-layer neural network. Its input is a text corpus and its output is a set of vectors. Word embedding via word2vec can make natural language computer-readable, then further implementation of mathematical operations on words can be used to detect their similarities. A well-trained set of word vectors will place similar words close to each other in that space. For instance, the words women, men, and human might cluster in one corner, while yellow, red and blue cluster together in another.

There are two main training algorithms for word2vec, one is the continuous bag of words(CBOW), another is called skip-gram.



CBOW (Continuous Bag of Words)

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.

Dataset

The dataset used for this project is TripAdvisor Hotel Review Dataset. It contains around 20K reviews and ratings of hotels from

TripAdvisor site. We will be using the review text for training of our Word2Vec CBOW model.

Work Performed

1. **Data Pre-processing:**

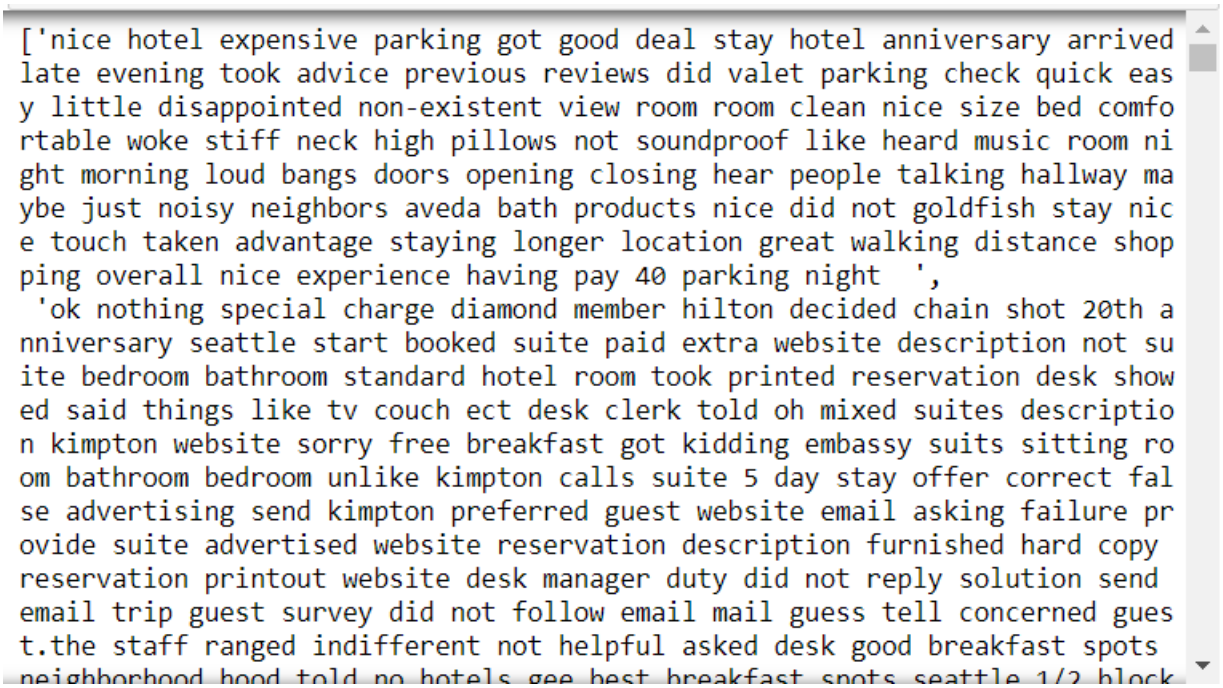
   The dataset consists of 20K reviews and ratings presented in a csv file. For our project, we only need the review text. Therefore we load the csv file into our Anaconda Environment using the *read_csv* code of pandas library provided by Python

	S.No.	Review	Rating
0	1	nice hotel expensive parking got good deal sta...	4
1	2	ok nothing special charge diamond member hilto...	2
2	3	nice rooms not 4* experience hotel monaco seat...	3
3	4	unique \tgreat stay \twonderful time hotel mon...	5
4	5	great stay great stay \twent seahawk game awes...	5

We can see that the full dataset has been loaded as a dataframe. We separate the reviews text from the entire dataset and convert into list. We use the python library NumPy for the task.

```
[['nice hotel expensive parking got good deal stay hotel anniversary \tarri
ved late evening took advice previous reviews did valet parking \tcheck qui
ck easy \tlittle disappointed non-existent view room room clean nice size
\tbed comfortable woke stiff neck high pillows \tnot soundproof like heard
music room night morning loud bangs doors opening closing hear people talki
ng hallway \tmaybe just noisy neighbors \taveda bath products nice \tdid no
t goldfish stay nice touch taken advantage staying longer \tlocation great
walking distance shopping \toverall nice experience having pay 40 parking n
ight \t '],
 ['ok nothing special charge diamond member hilton decided chain shot 20th
anniversary seattle \tstart booked suite paid extra website description not
\tsuite bedroom bathroom standard hotel room \ttook printed reservation des
k showed said things like tv couch ect desk clerk told oh mixed suites desc
ription kimpton website sorry free breakfast \tgot kidding \tembassy suits
sitting room bathroom bedroom unlike kimpton calls suite \t5 day stay offer
```

We can see above what the reviews text looks like. We have to remove the escape character. Also notice that the reviews text has been inside list of lists. We need to make sure that all the reviews are contained in a single list separated by a comma.



This is the final list that we will be using to build our model.

2. Tokenization

NLP libraries like NLTK offer a variety of language specific tokenizers that cover most of the corner cases. Each implementation has different priorities and one can choose the most appropriate one for each use-case. The RegexpTokenizer has been used in our project.

The screenshot above depicts tokenization into words.

3. Creating the model

Gensim's Word2Vec implementation lets us train our own word embedding model for a given corpus.

Training the model is fairly straightforward. We just instantiate Word2Vec and pass lists of words, where each list within the main list contains a set of tokens from a user sequence. Word2Vec uses all these tokens to internally create a vocabulary that is a set of unique words.

```
from gensim.models.word2vec import Word2Vec
```

```
UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
model = Word2Vec(words, size=2, min_count=1, seed=123456)
```

The model above has implemented CBOW model.

To train the model, we had to set some parameters. These have been discussed below: size: The size of the dense vector to represent each token or word (i.e. the context or neighboring words). If you have limited data, then size should be a much smaller value since you would only have so many unique neighbors for a given word. If you have lots of data, it's good to experiment with various sizes. A value of 200 has worked well for this study.

window: The maximum distance between the target word and its neighboring word. If your neighbor's position is greater than the maximum window width to the left or the right, then, some neighbors would not be considered as being related to the target word. In theory, a smaller window should give you terms that are more related. We used 20 as window in this study.

min_count: Minimium frequency count of words. The model would ignore words that do not satisfy the min_count. Extremely infrequent words are usually unimportant, so it's best to get rid of those. We used 3 as min_count.

workers: Number of threads to use behind the scenes. We have a virtual server consisting of 75 cores and 124 GB of RAM. We assigned number of CPU that is 74 to worker.
iter: Number of iterations (epochs) over the corpus. We used a minimum of 20 iterations. seed: Seed for random number generation.

4. Analyzing the results:

a. Returning vector of a given word:

```
print('word vector for the word "hotel"')
print(model.wv['hotel'])
```

```
word vector for the word "hotel"
[ 1.5175968 -3.9829397]
```

```
print('Cosine distance between "hotel" and "people" is {}'.format(model.wv.di
```

```
Cosine distance between "hotel" and "people" is 0.18806281204609432
```

```
import numpy as np
euc_dist = np.linalg.norm(model.wv['hotel'] - model.wv['people'])
print(euc_dist)
```

```
2.5040083
```

b. Comparing the cosine and Euclidean distances

c. Cosine similarity between two given words

```
Cosine similarity between "hotel" and "people" is 0.8119371879539057
```

d. Returning ‘n’ most similar words of a given word (here the word used is “hotel”)

```
w1 = ["hotel"]
model.wv.most_similar (positive=w1,topn=20)
```

```
[('byob', 1.0),
 ('30th', 1.0),
 ('olds', 1.0),
 ('plumeria', 1.0),
 ('understandably', 1.0),
 ('flex', 0.9999999403953552),
 ('fanuiel', 0.9999999403953552),
 ('stations', 0.9999999403953552),
 ('exhaustive', 0.9999999403953552),
 ('50k', 0.9999998807907104),
 ('walkable', 0.9999998807907104),
 ('greating', 0.9999998807907104),
 ('opportunistic', 0.9999998807907104),
 ('adopting', 0.9999998211860657),
 ('tipsfor', 0.9999998211860657),
 ('radisson', 0.9999997615814209),
 ('21st', 0.9999997615814209),
 ('vendi', 0.9999996423721313),
 ('convento', 0.999999463558197),
 ('catarina', 0.9999994039535522)]
```

5. Saving the model

#save model

Model.save(model.bin)

6. Loading the save

#load model

New_model = WordVec.load("model.bin")
Print(new_model)

Output

WordVec(vocab=52970,size=2,alpha=0,025)

Dataset used:

“Barkha Bansal. (2018). TripAdvisor HotelReview Dataset