word2vec

Mathematics and Applications of Machine Learning

Yannick Couzinié

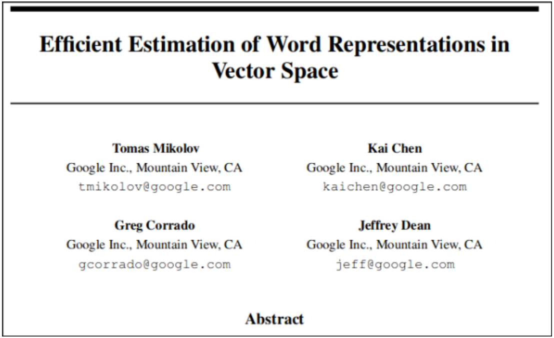
Ludwig-Maximilians-Universität München
Mathematics Deparment

31 May 2017

Structure

What is word2vec/word embedding?

The paper



Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

Figure 1: Original paper by Mikolov et al.

- Two 2013 papers accumulating over 7000 citations.

The task

Lorem ipsum dolor sit
amet, consetetur
sadipscing elitr, sed
diam nonumy eirmod
tempor invidunt ut
labore et dolore magna

Corpus C

The task

Lorem ipsum dolor sit
amet, consetetur
sadipscing elitr, sed
diam nonumy eirmod
tempor invidunt ut
labore et dolore magna

Corpus C

[Lorem]

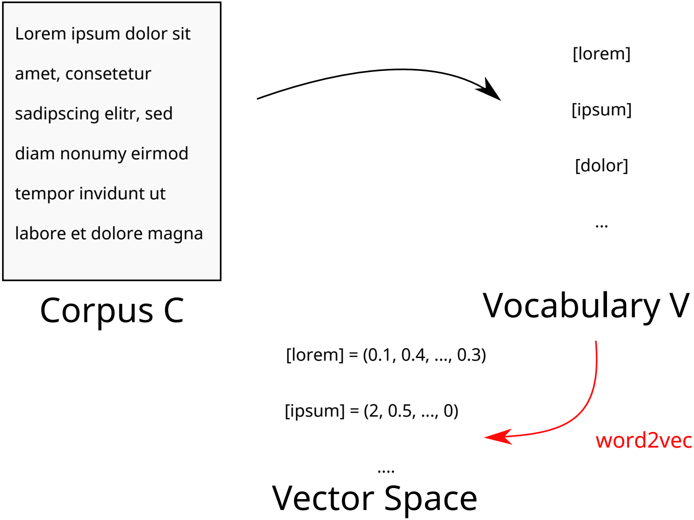
[ipsum]

[dolor]

...

Vocabulary V

The task


Lorem ipsum dolor sit
amet, consetetur
sadipscing elitr, sed
diam nonumy eirmod
tempor invidunt ut
labore et dolore magna
Corpus C
[lorem]
[ipsum]
[dolor]
...
Vocabulary V
word2vec
[lorem] = (0.1, 0.4, ..., 0.3)
[ipsum] = (2, 0.5, ..., 0)
....
Vector Space

Why do we want a word embedding algorithm?

Main uses

- Classification.
- Sentence or document analysis.d
- Similarity analysis.

$$\left.\begin{array}{l}\text{I admire my pet.}\\ \text{I adore my pet.}\\ \text{I love my pet.}\end{array}\right\} \Rightarrow \mathit{adore}(\text{I}, \text{pet})$$

```
word_vectors.similarity('love', 'adore')
>>0.681687380259
word_vectors.similarity('love', 'admire')
>>0.490552324418
word_vectors.similarity('adore', 'admire')
>>0.637308353311
```

Semantic networks

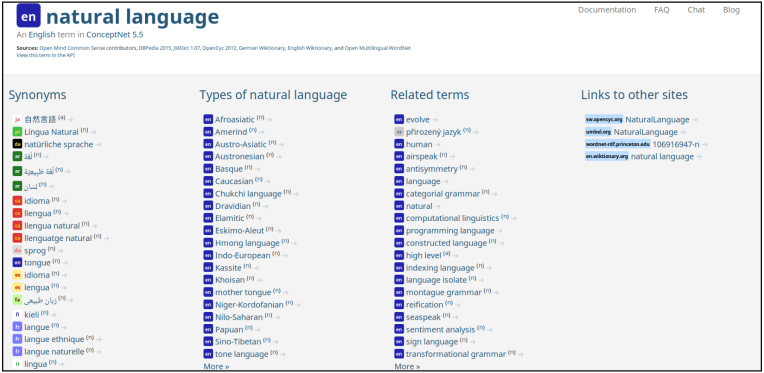


Figure 2: Screenshot of conceptnet webpage. Source: `conceptnet.io`

Problems:

- Newer and rarer words not well covered.

N-grams

Figure 3: Screenshot of Google's N-gram webpage. Source: `https://books.google.com/ngrams/`.

Problems:

- Relations one can infer statistically are limited.

> You shall know a word by the company it keeps.
>
> J.R. Firth, 1957

This is a sentence.

$\Rightarrow$ (this, is) (is, a) (a, sentence)

```
for word in sentence:
        take the current_word_vector
        predict the next word
        if prediction vector not next_word_vector:
        (i.e. if the prediction wrong)
                do gradient descent

repeat epoch times
```

Architecture

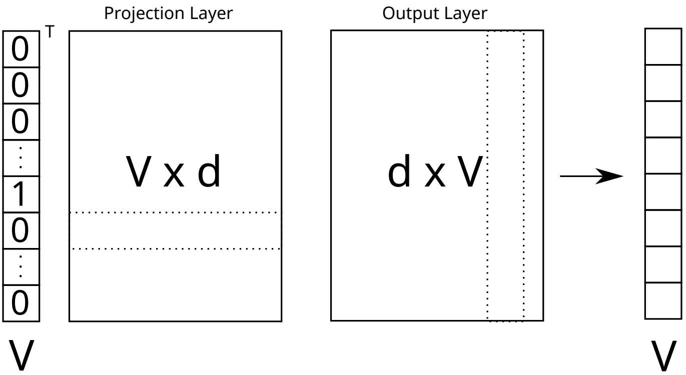


Figure 4: Architecture network used in the simple w2v-algorithm.

Effectively a 2-layer NN without activation function and cross entropy loss:

$$v_I(w_t)^T \cdot v_O(w_{t+1}) = v_{prediction} \rightarrow p(w_t) := softmax(v_{prediction})$$

Extract vocabulary and sentences using NLTK

```python
import nltk

# OMT: First read the files as a big string into self.text

nltk.download('punkt')
# Separate self.text per sentence into list of strings
self.sents = nltk.sent_tokenize(self.text)

# Extract the unique vocabulary
self.vocab = nltk.word_tokenize(self.text)
self.vocab = [x.lower() for x in self.vocab]
self.vocab = list(set(self.vocab))
```

Assign context to words

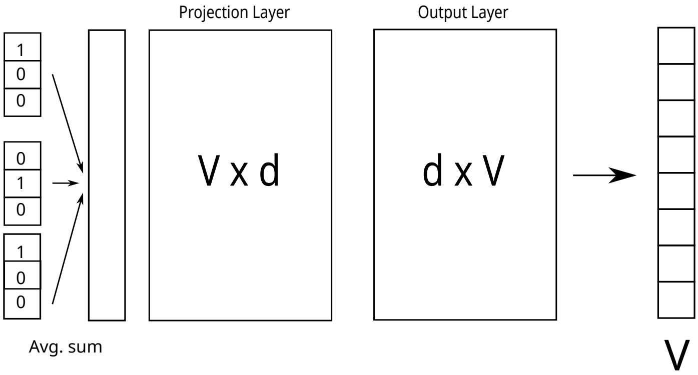
```
inps = []
outs = []
for sent in self.sents:
        sent = nltk.word_tokenize(sent)
        for i in range(len(sent)-1):
                # current input word
                word = sent[i]
                wordID = self.vocab.index(word.lower())
                inps.append(wordID) # sparse!
                # its corresponding context
                cntxt = sent[i+1]
                cntxtID = self.vocab.index(cntxt.lower())
                outs.append(cntxtID)
# The input data for the NN is inps whereby the i-th element
# has the i-th element of outs as corresponding label
# (in NN terms).
```

Increasing context window gives two possibilities:

- Predict target word from context (CBOW).
- Predict context from target word (Skip-gram).

Continuous Bag-of-words

Predict target word from context words as input.



$$\mathbf{v}_{Input} = \frac{1}{|C|} \sum_{c \in C} \mathbf{v}_c$$

Skip-gram

Predict context words from target word as input.

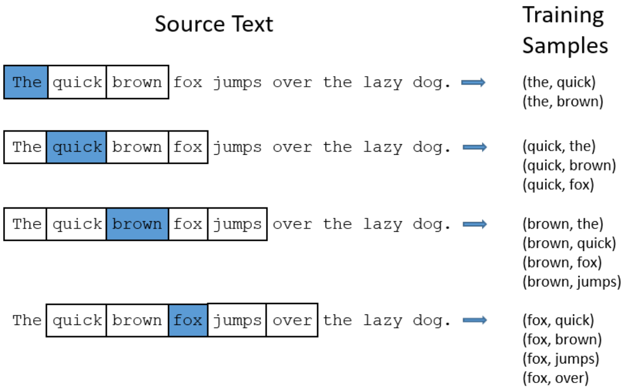


Figure 5: Image taken from `http://mccormickml.com/assets/word2vec/training_data.png` (27.5.2017).

Comparison

Computational costs:

- CBOW

$$O \propto (|C| \times d + d \times V)$$

- Skip-gram

$$O \propto |C| \times (d + d \times V)$$

Which one to use:

- CBOW is faster and better for frequent words.
- Skip-gram good with smaller corpus and rarer words.

N-grams

Extend vocabulary with ngrams

[New, York, Times] $\rightarrow$ [New, York, New York, Times, New York Times]

Introduce bigram score:

$$score(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)} ,$$

with discount coefficient δ (prevent infrequent n-grams).

- For n-grams run the bigram score multiple times.

Subsampling

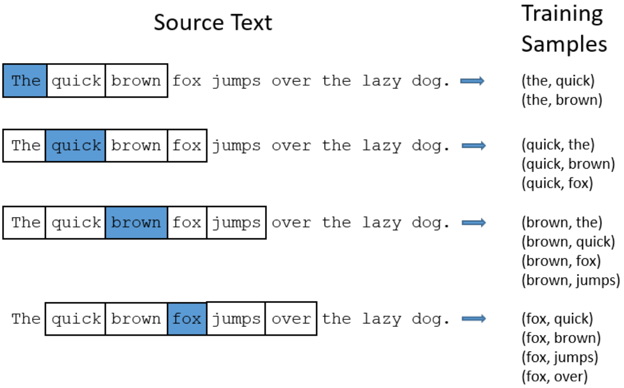


Figure 6: Image taken from `http://mccormickml.com/assets/word2vec/training_data.png` (27.5.2017).

$\Rightarrow$ *The* is meaningless.

Subsampling

Probability to remove/subsample:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \frac{0.001}{z(w_i)} \approx \frac{1}{\sqrt{z(w_i)}} ,$$

with $z(w_i)$ the relative frequency of the word.

- Only subsample words with frequency $\geq 0.26\%$.
- The frequent word does not appear in context windows.
- Deleting the window means up to $4|C|$ less training data.

Negative Sampling

- Gradient descent trains every weight based on one data tuple.
- Push one value to one and others to zero.
- Update only positive word and subset of negative words.
  - 5-20 words for smaller and 2-5 for large datasets.
  - For usual corpora: $\leq 0.1\%$ of weights!

Negative Sampling

In the code that means:

- Associate probability to each word given by

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum \left(f(w_j)^{3/4}\right)} .$$

- Create array of size $100M$.
- Enter each word $P(w_i) \times 100M$ times.
- Select random item from table.

$\Rightarrow$ Frequent words get corrected more frequently.

```
import gensim.models.KeyedVectors as kv

model = './GoogleNews-vectors-negative300.bin'
wordv = kv.load_word2vec_format( model, fbinary=True)

print('What is your base vector?')
positive1 = input()
print('What is the vector you want to substract?')
negative = input()
print('What is the vector you then want to add?')
positive 2 = input()

print('The most similar vector to this corresponds to: \n ')
print(wordv.most_similar(positive=[positive1, positive2],
          negative=[negative]))
```

Thank you for your attention.