word2vec

Tool for computing continuous distributed representations of words.

Project Information

Starred by 857 users
Project feeds

Code license
Apache License 2.0

Labels
NeuralNetwork, MachineLearning, NaturalLanguageProcessing, WordVectors, Google

Members
tmiko...@gmail.com
6 contributors

Links

Groups
Discussion group for the word2vec project.

Introduction

This tool provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. These representations can be subsequently used in many natural language processing applications and for further research.

Quick start

- Download the code: svn checkout http://word2vec.googlecode.com/svn/trunk/
- Run 'make' to compile word2vec tool
- Run the demo scripts: *./demo-word.sh* and *./demo-phrases.sh*
- For questions about the toolkit, see http://groups.google.com/group/word2vec-toolkit

How does it work

The *word2vec* tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words. The resulting word vector file can be used as features in many natural language processing and machine learning applications.

A simple way to investigate the learned representations is to find the closest words for a user-specified word. The *distance* tool serves that purpose. For example, if you enter 'france', *distance* will display the most similar words and their distances to 'france', which should look like:

```
                 Word       Cosine distance
------------------------------------------
                spain              0.678515
              belgium              0.665923
          netherlands              0.652428
                italy              0.633130
          switzerland              0.622323
           luxembourg              0.610033
             portugal              0.577154
               russia              0.571507
              germany              0.563291

            catalonia              0.534176
```

There are two main learning algorithms in *word2vec* : continuous bag-of-words and continuous skip-gram. The switch *-cbow* allows the user to pick one of these learning algorithms. Both algorithms learn the representation of a word that is useful for prediction of other words in the sentence. These algorithms are described in detail in [1,2].

Interesting properties of the word vectors

It was recently shown that the word vectors capture many linguistic regularities, for example vector operations *vector('Paris') - vector('France') + vector('Italy')* results in a vector that is very close to *vector('Rome')*, and *vector('king') - vector('man') + vector('woman')* is close to *vector('queen')* [3, 1]. You can try out a simple demo by running *demo-analogy.sh*.

To observe strong regularities in the word vector space, it is needed to train the models on large data set, with sufficient vector dimensionality as shown in [1]. Using the *word2vec* tool, it is possible to train models on huge data sets (up to hundreds of billions of words).

From words to phrases and beyond

In certain applications, it is useful to have vector representation of larger pieces of text. For example, it is desirable to have only one vector for representing *'san francisco'*. This can be achieved by pre-processing the training data set to form the phrases using the *word2phrase* tool, as is shown in the example script *./demo-phrases.sh*. The example output with the closest tokens to *'san_francisco'* looks like:

```
                Word       Cosine distance
-------------------------------------------
          los_angeles              0.666175
          golden_gate              0.571522
              oakland              0.557521
           california              0.554623
            san_diego              0.534939
             pasadena              0.519115
              seattle              0.512098
                taiko              0.507570
              houston              0.499762
     chicago_illinois              0.491598
```

The linearity of the vector operations seems to weakly hold also for the addition of several vectors, so it is possible to add several word or phrase vectors to form representation of short sentences [2].

How to measure quality of the word vectors

Several factors influence the quality of the word vectors:

- amount and quality of the training data
- size of the vectors

- training algorithm

The quality of the vectors is crucial for any application. However, exploration of different hyper-parameter settings for complex tasks might be too time demanding. Thus, we designed simple test sets that can be used to quickly evaluate the word vector quality.

For the word relation test set described in [1], see *./demo-word-accuracy.sh*, for the phrase relation test set described in [2], see *./demo-phrase-accuracy.sh*. Note that the accuracy depends heavily on the amount of the training data; our best results for both test sets are above 70% accuracy with coverage close to 100%.

Word clustering

The word vectors can be also used for deriving word classes from huge data sets. This is achieved by performing K-means clustering on top of the word vectors. The script that demonstrates this is *./demo-classes.sh*. The output is a vocabulary file with words and their corresponding class IDs, such as:

```
carnivores 234
carnivorous 234
cetaceans 234
cormorant 234
coyotes 234
crocodile 234
crocodiles 234
crustaceans 234
cultivated 234
danios 234
.
.
.
acceptance 412
argue 412
argues 412
arguing 412
argument 412
arguments 412
belief 412
believe 412
challenge 412
claim 412
```

Performance

The training speed can be significantly improved by using parallel training on multiple-CPU machine (use the switch '-threads N'). The hyper-parameter choice is crucial for performance (both speed and accuracy), however varies for different applications. The main choices to make are:

- architecture: skip-gram (slower, better for infrequent words) vs CBOW (fast)
- the training algorithm: hierarchical softmax (better for infrequent words) vs negative sampling (better for frequent words, better with low dimensional vectors)
- sub-sampling of frequent words: can improve both accuracy and speed

for large data sets (useful values are in range 1e-3 to 1e-5)
- dimensionality of the word vectors: usually more is better, but not always
- context (window) size: for skip-gram usually around 10, for CBOW around 5

Where to obtain the training data

The quality of the word vectors increases significantly with amount of the training data. For research purposes, you can consider using data sets that are available on-line:

- First billion characters from wikipedia (use the pre-processing perl script from the bottom of Matt Mahoney's page)
- Latest Wikipedia dump Use the same script as above to obtain clean text. Should be more than 3 billion words.
- WMT11 site: text data for several languages (duplicate sentences should be removed before training the models)
- Dataset from "One Billion Word Language Modeling Benchmark" Almost 1B words, already pre-processed text.
- UMBC webbase corpus Around 3 billion words, more info here. Needs further processing (mainly tokenization).
- Text data from more languages can be obtained at statmt.org and in the Polyglot project.

Pre-trained word and phrase vectors

We are publishing pre-trained vectors trained on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. The phrases were obtained using a simple data-driven approach described in [2]. The archive is available here: GoogleNews-vectors-negative300.bin.gz.

An example output of *./distance GoogleNews-vectors-negative300.bin*:

```
Enter word or sentence (EXIT to break): Chinese river

                 Word       Cosine distance
--------------------------------------------
       Yangtze_River              0.667376

             Yangtze             0.644091
      Qiantang_River              0.632979
   Yangtze_tributary              0.623527
    Xiangjiang_River              0.615482
       Huangpu_River              0.604726
      Hanjiang_River              0.598110
       Yangtze_river              0.597621
         Hongze_Lake              0.594108
             Yangtse             0.593442
```

The above example will average vectors for words 'Chinese' and 'river' and will return the closest neighbors to the resulting vector. More examples that demonstrate results of vector addition are presented in [2]. Note that more precise and disambiguated entity vectors can be found in the following dataset that uses Freebase naming.

Pre-trained entity vectors with Freebase naming

We are also offering more than 1.4M pre-trained entity vectors with naming from Freebase. This is especially helpful for projects related to knowledge mining.

- Entity vectors trained on 100B words from various news articles: freebase-vectors-skipgram1000.bin.gz
- Entity vectors trained on 100B words from various news articles, using the deprecated /en/ naming (more easily readable); the vectors are sorted by frequency: freebase-vectors-skipgram1000-en.bin.gz

Here is an example output of *./distance freebase-vectors-skipgram1000-en.bin*:

```
Enter word or sentence (EXIT to break): /en/geoffrey_hinton

                        Word       Cosine distance
--------------------------------------------------
           /en/marvin_minsky              0.457204
             /en/paul_corkum              0.443342
 /en/william_richard_peltier              0.432396
           /en/brenda_milner              0.430886
    /en/john_charles_polanyi              0.419538
          /en/leslie_valiant              0.416399
         /en/hava_siegelmann              0.411895
            /en/hans_moravec              0.406726
         /en/david_rumelhart              0.405275
             /en/godel_prize              0.405176
```

Final words

Thank you for trying out this toolkit, and do not forget to let us know when you obtain some amazing results! We hope that the distributed representations will significantly improve the state of the art in NLP.

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

[3] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT, 2013.

Other useful links

Feel free to send us a link to your project or research paper related to word2vec that you think will be useful or interesting for the others.

Tomas Mikolov, Quoc V. Le and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. *We show how the word vectors can be applied to machine translation.* Code for improved version from Georgiana Dinu here.

Word2vec in Python by Radim Rehurek in gensim (plus tutorial and demo that uses the above model trained on Google News).

Word2vec in Java as part of the deeplearning4j project. Another Java version from Medallia here.

Word2vec implementation in Spark MLlib.

Comparison with traditional count-based vectors and cbow model trained on a different corpus by CIMEC UNITN.

Link to slides about word vectors from NIPS 2013 Deep Learning Workshop: NNforText.pdf

Disclaimer

This open source project is NOT a Google product, and is released for research purposes only.