Statistical NLP - Assignment 5
Word Embeddings

Daniel Rivera Ruiz
Department of Computer Science
New York University
New York, NY 10003
drr342@nyu.edu

1 Problem Definition

The current assignment deals with the problem of word embeddings. A word embedding refers to the representation of a word as a high-dimensional vector of real numbers. In Natural Language Processing, word embeddings have been used in recent years to boost the performance of tasks such as syntactic parsing or sentiment analysis.

Several models have been used to produce word embeddings, among which *word2vec* is one of the most widely used. Originally developed by Tomas Mikilov and a group of researchers at Google, *word2vec* is based in a two-layer neural network trained to reconstruct the linguistic contexts of words, such that words that share common contexts in the corpus appear close to one another in the vector space. This approach is able to capture different degrees of similarity between words, both semantic and syntactic.

In section 2, the results obtained by using *word2vec* and its Python implementation *Gensim* are presented. Section 3 introduces several pretrained word embeddings that yield state of the art results and a quantitative comparison among them. Finally, the conclusions of the assignment are presented in section 4.

All the models were tested using the *wordsim353* evaluation dataset provided in Parikh (2017) and scored using the Spearman's rank correlation coefficient.

2 Word2vec and Gensim

Gensim (Řehůřek and Sojka, 2010) is a Python implementation of the *word2vec* model briefly described in the previous section. Among other features, *Gensim* provides a method `Word2vec` that takes as input a corpus, and returns a *word2vec* model with an embedding for each word in the vocabulary. To retrieve these embeddings in a readable format the method `save_word2vec_format` is provided.

The training process of the model accepts several parameters as arguments to the `Word2vec` method. The impact of the following parameters in the model performance was explored in this part of the assignment:

1. `sentences`. This parameter takes an object of type `MySentences` as proposed in Řehůřek (2014). To explore the effects of the size of the training corpus, two scenarios were considered: the *base* case, with only one million sentences, and the *extended* case, which includes additional sentences from the *1 Billion Word Language Model Benchmark*[1] provided in Parikh (2017).
2. `size`. This parameter takes an integer that defines the dimension of the embedding vectors. Two different values were considered in this case: 100 and 300.

[1] Due to storage limitations in the CIMS Servers, only half the sentences provided were used during training.

3. `sg`. This parameter is associated to the algorithm used to train the model. With the default value ($sg = 0$) the Continuous Bag of Words (CBOW) algorithm is used, otherwise ($sg = 1$) Skip-Gram (SG) is used instead.

4. `hs`. This parameter is associated to the optimization technique used by the model algorithm. With the default value ($hs = 0$) negative sampling (NS) is used, otherwise ($hs = 1$) hierarchical softmax (HSM) is used instead.

Table 1 shows the results obtained for different values of these parameters. The python script that was used to train the models is provided along with this document under the name `word2vec.py`. The word embeddings generated were the input to the scorer provided by Parikh (2017), which extracted embeddings exclusively for the words present in the *wordsim353* dataset and calculated the Spearman's score.

At this point, the best model was getting a Spearman's score of 65.38%. Using the same parameters, a new model was generated where the training process ran over 10 epochs (instead of the default value of 5), yielding an improvement of about 2 points for a final score of 67.56%. The parameter for the number of iterations (epochs) was only modified for the best scoring model because it resulted in a considerably larger training time.

Table 1: Spearman's Scores for different *word2vec* models on the *wordsim353* dataset.

`sentences`	`size`	`sg`	`hs`	Score
Base	100	0	0	0.5385
Base	100	0	1	0.4912
Base	100	1	0	0.6050
Base	100	1	1	0.5909
Base	300	0	0	0.5417
Base	300	0	1	0.5114
Base	300	1	0	0.5856
Base	300	1	1	0.6158
Extended	100	0	0	0.5786
Extended	100	0	1	0.5299
Extended	100	1	0	0.6195
Extended	100	1	1	0.5983
Extended	300	0	0	0.5865
Extended	300	0	1	0.5341
Extended	**300**	**1**	**0**	**0.6538**
Extended	300	1	1	0.6328
Extended	**300**	**1**	**0**	**0.6756**[2]

3 Pretrained Models

Given the popularity that word embedding models have experienced in recent years and the large amount of time and computational power required to properly train them, several general-purpose models have been pretrained and released as open-source. These models are the result of exhaustive parameter tuning over large amounts of data to provide robust word embeddings for standard datasets. They are trained using data available online (Wikipedia, Twitter, etc.) and are based on the *word2vec* or *GloVe* algorithms. *Glove* (Global Vectors for Word Representation) is an alternative to *word2vec* that was developed in Stanford University and that is also widely used to generate word embeddings.

Table 2 shows some of the most common pretrained models retrieved from 3Top (2017) and Bojanowski et al. (2016), along with the Spearman's scores obtained by the word embeddings generated with them on the *wordsim353* dataset. Some models provided embeddings encoded in a *.bin* format, in which case the script `load_bin.py` was used to convert them to a *.txt* format that could be read by the scorer.

[2]Model trained for 10 iterations.

It can be observed that most pretrained models yield results that are similar (or even worse) to the ones obtained with the models from the previous section. However, there are three of them that achieve considerably higher scores. This result is very important because it shows that, as long as the test dataset does not include a lot of "rare" words i.e., words from very particular domains such as medicine or biology, using a robust general-purpose model to calculate the word embeddings can be better than training a model from scratch, both in terms of performance and time investment.

Table 2: Spearman's Scores for pretrained models on the *wordsim353* dataset.

Corpus Source	Corpus Size	Model	Vector Dimension	Score
Wiki + Gigaword	6B	GloVe	50	0.4722
Wiki + Gigaword	6B	GloVe	100	0.5152
Wiki + Gigaword	6B	GloVe	200	0.5581
Wiki + Gigaword	6B	GloVe	300	0.5815
Twitter	27B	GloVe	25	0.3360
Twitter	27B	GloVe	50	0.4437
Twitter	27B	GloVe	100	0.5075
Twitter	27B	GloVe	200	0.5204
Common Crawl	42B	GloVe	300	0.6302
Common Crawl	**840B**	**GloVe**	**300**	**0.7379**
Wikipedia	?	word2vec	1000	0.6096
FB Fast Text	**?**	**word2vec**	**300**	**0.7031**
Google news	**100B**	**word2vec**	**300**	**0.7000**

4 Conclusions

Throughout this assignment, the concept of word embeddings was explored by implementing and testing models like *word2vec* and *GloVe* with different parameters. After analyzing the results obtained in these experiments, the following conclusions can be drawn:

1. The size of the training corpus plays a very important role and impacts directly the robustness of the word embeddings generated by the model in question.
2. In general, better results can be obtained by using the Skip-Gram algorithm instead of Continuous Bag of Words.
3. Vectors with more dimensions tend to better capture the linguistic features of the words associated to them. However, it is important to consider that more dimensions will translate into longer training times and bigger output files.
4. Using word embeddings generated with pretrained models is a cheap alternative in terms of time (since no training is needed) that yields satisfactory results for standard datasets.

References

3Top. Simple web service providing a word embedding API, 2017. URL `https://github.com/3Top/word2vec-api`.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

Ankur Parikh. Statistical NLP: Assignment 5, 2017. URL `https://cs.nyu.edu/~aparikh/assignment5-2017.pdf`.

Radim Řehůřek. Word2vec Tutorial, 2014. URL `https://rare-technologies.com/word2vec-tutorial/`.

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.