HW-2

Name: Ren, Jing

NetID: jrb1992

Problem 1

In order to generate word2vec embeddings of my text corpus, I used the python genism library which appears to be faster than the original word2vec C implementation.

The text corpus was chosen from a collection of news articles. The preprocessing included tokenization and normalization (converting to lower cases and removing non-alphabetic chars). The results were saved as a text file, each line containing a single document (news article), being separated by a single white space.

When experimenting with word2vec model parameters, I kept most of parameters fixed except for size, window, and sg.

Note that these 3 parameters were defined as below:

- **Size**: indicates the dimensionality of the word vectors.
- **Window**: shows maximum distance between the current and predicted word within a sentence.
- **Sg**: stands for training algorithm used. For skip-gram, sg = 1; for CBOW, sg = 0.

I built 4 word2vec models with sets of parameters:

- **Model 1**: size = 50, window=5, sg=0
- **Model 2**: size = 50, window=50, sg=0
- **Model 3**: size = 200, window=50, sg=0
- **Model 4**: size = 50, window=50, sg=1

I evaluated the embeddings by hand-picking 10 words, and manually reviewed their 5 closest neighbors in terms of cosine similarity under given models.

- **10 words**: central, good, bad, fact, evolve, large, political, god, science, assumptions

By comparing different windows, larger windows seemed to capture more domain information such as other words that were used in similar discussions. Smaller windows seemed more focused on words that were functionally similar. By comparing different embedding sizes, not much of differences were displayed. By comparing CBOW and skip-gram, CBOW tended to identify more accurate words in terms of cosine similarity for this dataset. I attached an example of word “god” for comparison purpose, please refer to CSVs for details.

'god'	Model 1	Model 2	Model 3	Model 4
1st	('one', 0.9324800968170166)	('proves', 0.841456949710846)	('kierkegaard', 0.8995345234870911)	('choices', 0.7635437250137329)
2nd	('existence', 0.9207386374473572)	('faith', 0.8083686828613281)	('interest', 0.8507446050643921)	('volumes', 0.7420107126235962)
3rd	('statement', 0.9185029864311218)	('kierkegaard', 0.8059536218643188)	('proves', 0.8355854749679565)	('obeying', 0.7346591353416443)
4th	('true', 0.9144179224967957)	('believing', 0.7893356680870056)	('faith', 0.8058984279632568)	('omnipotent', 0.7299832105636597)
5th	('fact', 0.9127639532089233)	('supercede', 0.7881574630737305)	('drives', 0.7615327835083008)	('omnipotence', 0.7196305990219116)

- Data Source: http://qwone.com/~jason/20Newsgroups/
- Relevant Code: https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework2/HW2.py
- Preprocessed Output File: https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework2/Processed_Output.txt
- Model Output CSVs:
    - https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework2/model_cbow1.csv
    - https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework2/model_cbow2.csv
    - https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework2/model_cbow3.csv
    - https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework2/model_skipgram.csv

Problem 2

Comparison of Word2vec and BERT in two papers:

	Word2vec	**BERT**
Date of Publication	2013	2018
Number of Google Scholar Citations	22,964	10,265
Corpus Size Requirements	Large amount of the training data is crucial.	Document-level corpus. For the pre-training corpus we use the Books Corpus (800M words) and English Wikipedia (2,500M words).
Approach Summary	There are mainly two model architectures for distributed representations of words. Skip-gram model predicts the context given words, while Continuous-Bag-of-Words model predicts words given the context.	BERT is designed to pre-train deep bidirectional representations from unlabeled text with two steps: pre-training (Masked LM and Next Sentence Prediction) and fine-tuning. Masked LM: mask some percentage of the input tokens at random, and then predict those masked tokens. Next Sentence Prediction: jointly pretrains text-pair representations.
Ease of Installation	Original implementation in C. Could also be implemented in Python library genism.	Available in Python library keras-bert as well as PyTorch-Transformers.

- Required Readings:
  - https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
  - https://www.aclweb.org/anthology/N19-1423.pdf