Exercises

Exercise #1

The goal of the first exercise is to implement skip-gram with negative-sampling from scratch. You will be provided with a template (python), where you should fill in the missing functions [link].
Please use the following formatting conventions:

- call your main python file (the one that should be executed) skipGram.py
- use python 3
- do not rely on any libraries beyond numpy, scikit-learn, spacy. If you need otheres, please first check with me
- there should be
  - one file variable __authors__ which is a list of strings containing your names
  - one file variable __emailsl__ which is a list of strings containing your emails. Please revise this one carefuly
- keep the arguments format as given in the template (-- ...)

Submission:

- send your code as an attachment (.zip, or - prefereably - .tar.gz) to matthias.galle__at__naverlabs.com
- **deadline is 22/02 23:59 French time**. Penalities apply after that (1 point per day)
- the main python file should be in the root directory of that compressed archive. Do not include additional data or models (compressed file should be < 2 MB)
- do not include any comment in the email body. Anything you want to add put it into a file named readme.txt in the root directory.

Questions
Use the forum to ask questions. Individual requests will not be answered in general.

Evaluation

1. Performance - 10pt
   Done automatically, training your code on a fixed data-set (text file containing English sentences, one per line). Evaluation will be done measuring the correlation between the similarity that you compute between two words with respect to human evaluation. Test data will be a tab-separated csv file, with one header line containing the words under columns 'word1' and 'word2' respectively, and the (human-annotated) similarity score as a float under column 'similarity'.
   For the training, I will use increasingly longer prefixes of the Billion Word Corpus, and for testing a data-set similar (not necesarily that one) to SimLex-999. You can use those data-sets to make sure that your code works as expected.

Do not write anything to stdout except the simlarity (see the main function in the code skeletong). If you want to debug, please use a logger or stderr.

2. Code & Readme. 5pt + [-10,4]
   In the Readme file you should specify your **thought process**, what you tried, what didn't work and explain design choices. The code should **be reasonably commented and structured**. You should also **cite** any additional sources you used and papers you read. **DO NOT COPY & PASTE CODE FROM THE INTERNET** without trying first to implement it on your own. If the code resembles too much, and there is no **clear explanation of your thought-process in the Readme file a penalty of -10 applies.**
   If there is obvious evidence of a from-scratch derivation of the optimization and clear code, you can gain up to 4 bonus points

3. Report on additional experiments. 5pt + [0,4]
   This is up to your creativity.
   I expect a **report** (of roughly 2-3 pages), in pdf, where you detail the conclusions of experiments and variations you performend. As example you might want to test the impact of different hyperparameters, initializations, combination of embeddings or implement some of the advanced topics presented in class. These are just examples. Again, you can get up to 4 bonus point for exceptional work done her.

References

Goldbery & Levy. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. Tech Report.
Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.