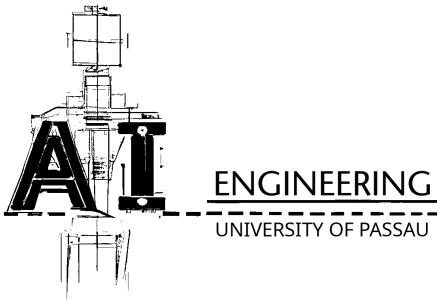
Deep Learning for Natural Language and Code

Exercise 4

Prof. Dr. Steffen Herbold
SoSe 2025
Due on 2025/06/05



General information for all exercises (read carefully!)

Within the "Deep Learning for Natural Language and Code Exercise," you will execute different tasks related to various NLP concepts. The main goal of these exercises is to teach you how to develop approaches. Once you have gained this knowledge, you will have the opportunity to use your own solution and compare it with existing solutions from popular libraries. This means that these exercises are not just about knowing how to use the libraries.

Problem description

One task that is executed in NLP is tokenization. It is time to work with learned embeddings. Until now, you have worked with a simple embedding, bag-of-words (BoW). But now, you will have the opportunity of implementing C-BoW and Skip-gram embeddings.

- `https://ai.stanford.edu/~amaas/data/sentiment/`

Data set description

For this exercise, we are going to work with the *Large Movie Review Dataset* [1]. This dataset was built for binary sentiment classification. It is composed of 25,000 highly polar movie reviews for training, and 25,000 for testing. Meaning that the middle values (i.e., scores of 5) are ignored. The dataset also contains unlabeled data. If you have memory restrictions please, follow the tips given in the *Restrictions and tips* section.

Programming Tasks (PT)

1. Implement the CBOW algorithm using Pytorch, and train it on the reviews from the train data.
2. Implement the Skip-gram algorithm using Pytorch, and train it on the reviews from the train data.
3. Analyze existing word embeddings:
   (a) Implement cosine similarity.
   (b) Execute the lines of code provided in the *Jupyter Notebook* related to `model1`, `model2`, and `model3`. Analyze why some of the cells fail to execute.

(c) Use cosine similarity to compute the similarities between `king_mins_man_plus_woman_m3/4 and queen_vector_m3/4`. Analyze the result of the similarities obtained.

(d) Search for a method that allows you to find the most similar words for a given word using the full Word2Vec model, loaded in the *Jupyter Notebook* (`model 4`). For example, for the word *phone*, the most similar ones are: *telephone*, *cell_phone*, and *cellphone*.

(e) Use the identified method in the previous step to find at least five analogies. **Hint:** for example, if you want to find the analogy presented in the lecture (Chapter 4, slide 27), *"The word king is to men in the same way as the word queen is to women"*, you will need to use the words king, man, and woman in order to identify the queen word.

(f) *Bonus:* Analyze your CBOW and Skip-gram embeddings in a similar way, e.g., try to find such analogies in these embeddings.

Restrictions and tips

- The tasks must be solved using only Python Standard Library (unless stated otherwise). You can use implemented data structures but for the text processing/tokenization you should use only the Python Standard Library.
- *Memory limitations.* Do not load to memory all the reviews, start executing the exercise with only 100 reviews, and increase the number of instances as wanted.
- Note that you could use Google Colab for executing the exercises (`https://colab.research.google.com`) or Kaggle Kernel (`https://www.kaggle.com/code`).
- *Memory limitations.* You can download a dataset directly to Google Colab. There are multiples tutorial in the the web for doing that. For example `https://niruhan.medium.com/downloading-a-dataset-and-displaying-an-image-in-google-colab-5370f20b236d`.
- You can re-use code between exercises.

Theoretical questions

1. Review the cosine similarity and answer the following questions:
   - What does it mean to have a cosine similarity of 0 between two vectors?
   - What does it mean to have a cosine similarity of 1 between two vectors?
   - What does it mean for two vectors to have cosine similarity equal to -1? Is it possible to have a cosine similarity of -1 when working with vector representation of frequencies?
2. How do the input and output of the learned CBOW and Skip-gram embeddings differ?
3. What will happen if padding is not used in CBOW?
4. What happens to the models if the size of the dataset is increased (e.g., vocabulary, number of instances to train on)?
5. Read the Gensim documentation.[1] What is the difference between a *KeyedVector* and a full model, and in which cases is each type of model useful?

[1] See `https://radimrehurek.com/gensim/models/keyedvectors.html` and `https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4`

6. After analyzing the similarities obtained in the programming task # 3 (c), provide a reason why the similarities vary that much between the `model3` and `model4`.
7. How does the method identified in programming task # 3 (d) find similar words?
8. If you use the identified method in the programming task # 3 (d) to find the most similar words to the vector `king_mins_man_plus_woman_m4`, why do you think that the most similar word is *King* and not *Queen*?.

References

[1] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.