Deep Learning for Natural Language Processing

SS21 – Exercise sheet 0 – 10 points

Submission start via LernraumPLUS: 15.04.2021, 3 pm
Submission deadline via LernraumPLUS: 21.04.2021, 10 pm

Welcome to this lecture. The exercises are crucial to pass this course. Even more, the practice will give you experiences in programming natural language processing with the latest approaches. Hence, after the course, you will be probably able to face complex problems in natural language processing – a big plus in industry and research.

Probably, Python is the most common programming language for modern natural language processing. In this course, we will work with Python 3.8. Version 3.7 is suitable, too. Please notice that we cannot accept submissions in other programming languages or Python Version ≤ 3.5.

Hint: we recommend the IDE PyCharm[1] for developing Python applications.

Sheet 0 – Task 1 (1 points)

Please execute `TrainWord2Vec.py` from the provided code. To this end, you must probably ensure that your computer provides all imported libraries.

```
1    from gensim.models import Word2Vec
```

If not, you can use the `pip install` command.
Please submit a screenshot from a (Python) console with the logging output generated by `TrainWord2Vec.py`. You have not to ensure that the whole log is visible, but at least the vector representation of the word "Israel".

Note: the dimensionality of a word vector is set to 32 in the code: Play a little bit with the dimensionality by changing the size-parameter in

```
model = Word2Vec(sentences=sentences, corpus_file=None, size=32,
    window=5, min_count=2, workers=4, iter=3, compute_loss=True)
```

Sheet 0 – Task 2 (3 points)

Now, let's move to `Tasks.py`: Implement a method `count_similar_word_vector_positions` which takes two word embeddings (assert the same length) and a float number called *delta*.
Let $\vec{w}$ the first vector, $\vec{v}$ the second one. The method should count all vector fields where

$$abs(w_i - v_i) \leq delta \tag{1}$$

The method should return the counted number.

[1] `https://www.jetbrains.com/pycharm/download/`

Sheet 0 – Task 3 (3 points)

Again `Tasks.py`: Implement a method `get_most_similar` which takes a word (string) and returns the top 10 similar words (only strings – sorted by similarity of their word embedding towards the given word). If the given word doesn't exist as a word embedding, then the method should return an empty list.
The similarity of two vectors $\vec{w}$ and $\vec{v}$ should be computed by the cosine similarity:

$$cos(\vec{w}, \vec{v}) = \frac{\vec{w} \cdot \vec{v}}{||\vec{w}|| \cdot ||\vec{v}||} \tag{2}$$

Note, that $cos(\vec{w}, \vec{v}) \in [1, -1]$, where a number close to 1 indicates a strong similarity.
Answer the following question in an additional pdf-file: What are the reasons that a word doesn't belong to a word embedding?

Sheet 0 – Task 4 (3 points)

Again `Tasks.py`: Implement a method `get_D_of_A_is_related_to_B_like_C_to_D` which takes three words (strings), `A`, `B` and `C` and returns a list of fitting words the suits the following description: `A` is related to `B` like `C` to *fitting words*. An example: `king` is related to `man` like `queen` to *woman*. Hence, the relation is the gender: king is male, and thus, queen is female.
Word Embeddings can capture such relations by $\vec{A} - \vec{B} \approx \vec{C} - \vec{D}$. So, in our example, a well-trained word embedding collection would infer $\vec{king} - \vec{man} \approx \vec{queen} - \vec{women}$
Please execute your method with the example in the code `A="Israel", B="Jerusalem", C="U.S."` and look through the result. Choose a suitable length of the returned list.

Answer the following question in an additional pdf-file: which word would you expect in the example? Analyse the returned list – do you find the expected word? What else? Write a *short* explanation, why the result may is like it is and describe *one* possibility to improve the outcome.