Deep Learning in NLP

Homework 2

Solution to be sent to `waszczuk@phil.hhu.de` and `cwurm@phil.hhu.de` by 16.11.2020 (included, i.e. can be sent till Monday in the evening). It is allowed to do the homework in groups (max. 3 persons per group). Please specify in the email (or in an accompanying README) the authors of the solution.

The archive file with one Python file per exercise (`ex1.py`, `ex2.py`, ...) can be found on the course's website. The missing pieces in the code are marked with `TODO`. Please do not modify the docstring tests, nor the sample datasets refered from the docstrings, they will be used for evaluation. The solution sent by email should have the same form: it should be a zip archive file with the same number of Python files, having the same names: `ex1.py`, `ex2.py`, ...[1]

Evaluation

The following command will be used to check the solutions:

```
python -m doctest ex1.py
```

Similarly for the other exercises. The solutions which pass all the tests will be considered as correct.[2]

Exercise 1

In this exercise we start to tackle the problem of predicting languages of names. The goal of Ex. 1 is to parse and extract the dataset from a raw string such as this:

```
German: Bach, Engel, Gottlieb, Zimmermann
English: Alderson, Churchill, Ecclestone
Czech: Blazejovsky, Hruskova, Veverka
Greek: Antonopoulos, Leontarakis
```

[1] In case you do not provide a solution for a particular exercise, no need to include it in the archive.

[2] However, we reserve the right to count solutions with significant issues in the code as merely „acceptable", even when all tests pass.

```
Japanese: Fujishima, Hayashi
Korean: Park, Seok
Spanish: Álvarez, Pérez
English: Keighley, Reynolds
```

The resulting dataset should take the form of a list of (name, language) pairs. More information, including examples of how the output should look like, can be found in `ex1.py`.

Exercise 2

The goal of Ex. 2 is to implement a function to encode a name/language dataset (see Ex. 1) in the tensor form.[3] As a result of encoding, each input *character* should be transformed to the corresponding index.[4] For instance:

```
Bach -> tensor([0, 1, 2, 3])
```

Each output class (language) should also be encoded as an index, for instance:

```
German -> tensor(0)
```

More information and examples can be found in `ex2.py`.

Note: It is not important if `German` is represented as 0 or 1, what is important is that each output class is represented as a unique class index (i.e., the index assigned to `German` differs from the index assigned to `English`), and that the set of indices forms the range from 0 to $C - 1$, where C is the number of classes (e.g., if there are 3 languages, the language indices should be 0, 1, and 2). Simiar considerations apply to encoding input names.

[3] No need to solve Ex. 1 to be able to solve Ex. 2.
[4] We will later use character-level embeddings as input to a neural language prediction model.