Deep Learning in NLP

Homework 4

Solution to be sent to `waszczuk@phil.hhu.de` and `cwurm@phil.hhu.de` by 30.11.2020 (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] In case the original code contains additional modules required to complete the homework (e.g. `utils.py`, `data.py`, etc.), please include them in your submission.

Evaluation

Unless specified otherwise, 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

Implement a language prediction model which takes on input a name (a string, such as *Bach*, after encoding) and outputs the corresponding language (a string, such as *German*, after encoding).

Specification of the model:

[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.

- Input: sequence of character indices $(c_i)_1^n$, where n is the length of the input name; $\forall_{i=1..n}\ c_i \in \{0, 1, \ldots, m-1\}$, where m is the alphabet size
- Embedding: each character on input c_i should be embedded as a vector $x_i \in \mathbb{R}^d$, where d is the embedding size
- To obtain the representation of the entire input name, use the basic *continuous bag-of-words* (CBOW) representation:

$$v = \sum_{i=1}^{n} x_i \tag{1}$$

- Score the name CBOW representation using linear transformation:

$$s = \texttt{Linear}(v) \tag{2}$$

  Vector s should have as many values as there are different languages in a dataset.

See `ex1.py` for more details on the exercise.

Hint: Implement the CBOW component as a PyTorch `nn.Module`, which should be applied between the embedding and the scoring components. You can then extend the baseline model we have seen during the class to obtain the model specified above.

Bonus: Use the `EmbeddingBag` module provided by PyTorch in the implementation of the model. `EmbeddingBag` is equivalent[3] to a composition of `Embedding` and `CBOW` modules, but it's more efficient in practice.

(b) (optional)

Adapt the language prediction model to make it ignore input indices from outside the pre-determined alphabet. For instance, if there are 5 different characters in the training data (encoded as $0, 1, 2, 3, 4$), the model should give the same results for the following two input tensors:

- `torch.tensor([0, 2, 4])`
- `torch.tensor([-2, 0, 2, 4, 6])`

since neither -2 nor 6 corresponds to any actual character in the dataset.

Hint: To solve this exercise, you might want to revisit the section on tensors (recently updated regarding logical operators).

Note: If you wonder what might be the purpose of this exercise, it's related to the issue of the so-called *out-of-vocabulary* (OOV) words (well, OOV characters in this case...).

[3] When created with `mode="sum"`.

Exercise 2

Implement a function which returns the cross entropy loss between a *vector* of predicted scores and a target *scalar* index. Note that the target indices are scalar values in the encoded dataset, hence we need this variant of cross entropy loss for the sake of training the language prediction model.

Hint: There's a mismatch between the shapes expected by `nn.CrossEntropyLoss` and the shapes of the tensors in this exercise. Hence, you may need to *reshape* the tensors to satisfy the interface requirements of `nn.CrossEntropyLoss`.

Exercise 3

Train the model developed in Ex. 1 on the sample dataset, using the loss function from Ex. 2. The sample dataset as well as the pre-processing and encoding procedures are already implemented in the `data.py` module.

Hint: Have a look at the full SGD training example presented during the session.

Evaluation: Ex. 3 will be evaluated with the following command:

```
python ex3.py
```

which outputs the message whether the evaluation succeeds or fails. There's an evaluation section at the end of the `ex3.py` script which should not be modified.