Deep Learning in NLP

Practical Session P9

Introduction

We continue to work on the implementation of the LSTM-based POS tagger for UD. See the description of the task from the last week.

Preparation

You can continue working with your own codebase, but it is recommended that you download the current version from the website (or from github). It contains the solutions to the last homework exercises, the new code fragments we will work on today, and the English UD dataset.

Exercises 1-3

See the notes on github concerning the implemented homework solutions.

Exercise 4

The goal of this exercise is to implement a simplified verion of a `PosTagger` in the `main.py` file. It should:

- Use the word embedding module (`AtomicEmbedder`) from Ex. 2 to embed the individual input words.
- Use the linear layer to transform the resulting embeddings into score vectors, independently for each word in the input sentence.

Also, it may be useful to have a look at the PyTorch module usage example at this point.

4a

Finish the implementation of the `forward` and the `__init__` methods in the `PosTagger` class. The places to fill/update are marked with `TODO`s.

Question: Could the `WordEmbedder` be an arguent of the `forward` method rather than `__init__`? After all, we only need to really perform embedding in `forward`.

At the end of the `main.py` file, an instance of a `PosTagger` based on the word embedding module from Ex. 2 and the POS tagset from Ex. 3 is created. Calculate the score tensor for a sample sentence (e.g., *James eats pizza*) to verify that your implementation of `forward` and `__init__` works.

4b

Complete the implementation of the `tag` method of the `PosTagger` class, which POS tags the given sentence. It should be based on the `forward` method from Ex. 4a.

Remember that this method is not related to the training process. Hence, `torch.no_grad` should be used. Also, `torch.argmax` can be useful to solve this exercise.

Finally, test the implementation of `tag` by applying it to a sample sentence.

Exercise 5

Train the implemented model on the English UD train set and determine the resulting accuracy on the dev set.

5a

Complete the implementation of the `total_loss` function in `main.py` which calculates the cross-entropy loss over the given batch of sentences. The CrossEntropyLoss usage example should help with this exercise.

5b

Use the training function from `neural/training.py` in order to train the model. It is a variant of the `train` function that we implemented before, abstracted so as to work with generic PyTorch models. You can place the training-related code at the end of the `main.py` file.

Exercise 6

Add LSTM on top of the input word embeddings in order to obtain contextualized word representations, before they are scored with the linear layer. Have a look at the example of the PyTorch LSTM usage.

More information on this exercise will be provided later.