Deep Learning in NLP

Practical Session P11

Introduction

Today we finish the implementation of the LSTM-based POS tagger for UD (recall the description of the task and the previous exercises). Starting from the next week, we will be extending the tool with the dependency parsing abilities.

Preparation

Download the current version of the code from the website (or from github). It contains new code fragments we will work on today and an English UD dataset.

Exercise 6d (reminder)

Experiment with different LSTM hyperparameters:

- bidirectional LSTM (by default, PyTorch LSTM is unidirectional)
- number of layers (by default, one LSTM layer is used)
- `hidden_size` of the LSTM

Is any of these crucial to get high POS tagging accuracy?[1]

Exercise 7

The goal of this exercise is to add the pre-trained `fastText` embeddings to our model. More precisely, we shall use `fastText` instead of training our custom embeddings.

Places to fill are marked with `TODO EX7`.

[1] Hint: consider two sentences *he has a hat* and *he has bought a hat*. Is it within the power of the tagger to tag both sentences – in particular, the word *has* – correctly?

7a

Download the English embeddings from the course webpage and unzip them. Have a look inside to understand the format. Also, you can run the following in IPython:

```
run word_embedding.py
ft = FastText("wiki-news-300d-1M-subword-selected.vec")
ft.data.keys()          # list of words with embedding vectors
ft.data["confusion"]    # embedding for "confusion"
```

where `wiki-news-300d-1M-subword-selected.vec` is the downloaded embedding file.

7b

The `word_embeddings.py` file constains a stub of the `FastText` word embedding class. Complete it by implementing the `forward` method.

Do *not* make the `FastText` embedding vectors parameters of the model – we want to keep them fixed throughout the training process (what is the advantage of that?).

7c

Use the `FastText` word embedder as a component of the POS tagger in `main.py`.

Train the tagger and see if there is a benefit of using pre-trained embeddings.

Note: you may want to reduce the LSTM `hidden_size` (to e.g. 50) to speed up training.

Exercise 8

Dropout is a technique which allows to avoid overfitting and to better generalize the model. The goal of this exercise is to apply this technique in our POS tagger.

Places to fill are marked with `TODO EX8`.

8a

Read the section about dropout on github, which shows the usage of the PyTorch Dropout class.

8b

Apply dropout (with the dropout probability $p = 0.25$) to the output of the first layer of the LSTM. To this end, you can simply set the LSTM's `num_layers` hyperparameter to 2 and its `dropout` hyperameter to 0.25. Also consider training for 50 epochs.

IMPORTANT: remember to use dropout only during evaluation/tagging!

Re-train the model and check the influence of dropout on the resulting accuracy on the dev set.

8c (optional)

There are other places in the code/model where applying dropout could make sense:

- The `fastText` vectors
- The LSTM output vectors

Perform additional experiments to see if you can still improve the performance of the model. You may also want to change other hyperparameter values.