Übung 5: Encoder-Decoder-Modelle - Report

Preprocessing und Hyperparameters

Before starting the training, I performed various preprocessing steps to the input file and changed some hyperparameters in the original codes.

About the first point, I practically followed what we did by Exercise 2 (so as a reference you see the steps on that assignment), i.e. normalize, tokenize, clean, true case, and train and apply a BPE model. But differently from that time, I changed the vocabulary size in the BPE training: instead of perform 30'000 operations during the BPE training, I preferred to lower the rate to 22'000. I opted for that value because I read (`http://forum.opennmt.net/t/bpe-vocab-size/897/12`) that the BPE's parameters size swing between 10'000 and 30'000, and lower or bigger value could cancel the BPE's effect. Since the Exercise 2's value was exactly at that limit, I preferred to try some sort of middle way, i.e. 22'000. I got some problems by the training and generation of the BPE model, and the subsequent translation, as you can see on the following picture:

```
York xth enced gress asion asion asion ple ssion aranept arance arangress ple cken pulation cken cken pulation cken cken pulation cken cken pulation cken
cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken ck
en pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken
pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken pulation cken cken
```

Figure 1: Part of the output of the BPE model.

In fact, the BLEU score on the devset (computed with `sacrebleu`) was very low, i.e. 8.0.

Regarding the hyperparameters: I increased the vocabulary size to 75'000 (see `constants.py`, lines 36-37), hoping that in this way I could lower the number of unknown words, and I increased as well the hidden and embedding size (respectively, to 1152 and to 640; see `constants.py`, lines 39-41), trying to improve my training and taking in mind that with the implementation of the dropout mechanism (see below) overfitting could be avoided. I also used a bigger number of epochs and batches: that was a huge mistake (particularly the first one), because it extremely extended the training time. According with Mathias, with the intent to spare money and time, I stopped my training by epoch 7 (iteration 5000 on 42'396).

Code(s) implementation

Precisely as a consequence of this money and time's issue, I wasn't able to test of my code implementation, in order to see which kind of problems I would get and how to solve it. So on my GitHub repository you will find my version of the implemented code(s) (mainly in `compgrahpy.py` and `train.py`), but I am not sure if I did it right and how I could potentially improve and correct it. I thought of three possible implementations: early stopping, dropout and removing unknown words.

Early stopping is a technique that can stop training at the point when performance on a validation dataset starts to degrade, in order to avoid under- or overfitting (`https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/`). Here how this version of early stopping works (or at least should work):

> we can implement early stopping as a callback function. Callbacks are functions that can be applied at certain stages of the training process, such as at the end of each epoch. Specifically, in our solution, we included `EarlyStopping(monitor='val_loss', patience=2)` to define that we wanted to monitor the test (validation) loss at each epoch and after the test loss has not improved after two epochs, training is interrupted. However, since we set `patience=2`, we won't get the best model, but the model two epochs after the best model. Therefore, optionally, we can include a second operation, `ModelCheckpoint` which saves the model to a file after every checkpoint (which can be useful in case a multi-day training session is interrupted for some reason. Helpful for us, if we set `save_best_only=True` then `ModelCheckpoint` will only save the best model (`https://chrisalbon.com/deep_learning/keras/neural_network_early_stopping/`).

Based on what is said on the cited website, I tried to implement early stopping in the `train.py` file (see lines 17, 54-57, and 63 of the code).

Another good way to improve the training and avoid overfitting is dropout,[1] which "approximate an exponential number of models to combine them and predict the output" (`http://laid.delanover.com/dropout-explained-and-implementation-in-tensorflow/`). The idea behind it is that "the network 'drops' (i.e. does not use) some of its nodes in the prediction. […]. Thanks to dropout, the network learns not to rely exclusively on particular nodes for its prediction" (`https://stackoverflow.com/questions/45917464/tensorflow-whats-the-difference-between-tf-nn-dropout-and-tf-contrib-rnn-dropo`): "each neuron is dropped at random with some fixed probability 1-*p*, and kept with probability *p* [i.e. the `keep_prob` argument in the `tf.nn.dropout` function]"

[1] Dropout is already implemented in Tensorflow itself. See: `https://www.tensorflow.org/api_docs/python/tf/nn/dropout`.

(`https://www.commonlounge.com/discussion/694fd08c36994186a48d122e511f29d5`). You can find the dropout implementation on the lines 43 and 46 of the `compgraph.py` file.

The last customization I tried was to remove the unknown words (to which is normally assigned a `<unk>` tag). But since I used a BPE model – which central scope is exactly to avoid unknown words by rendering these words as sequence of single characters – I believe that this last step could be avoided. By the way, you can see my modification in the `constants.py` (lines 12 and 16), `vocab.py` (39, 41, 62-66), and the `bin/daikon` (39-42) files, where I removed every possible reference to the unknown words.

Repository link

This is the link to my GitHub repository: `https://github.com/rborrello/daikon`.