Machine Translation: Practical Work 3

Sharid Loáiciga

In this assignment, you will train your first Neural Machine Translation (NMT) models. You will learn how to apply various preprocessing steps and how to use the OpenNMT-py toolkit[1]. You will work on a dataset of image captions in English, German, French, and Czech. This dataset, called multi30k[2], is quite small (30,000 parallel sentences), but it will allow you to obtain results relatively quickly.

1 Set up

You will be working in the CogSys GPU server. To log in, you need the `ssh` command:

```
ssh USER@jarvis.ling.uni-potsdam.de
password:  PASSWORD
ssh pm-nmtX
```

Replace USER and PASSWORD below with your assigned user and project name, replace X with your assigned machine.

The data for this session is available at `data/shared_folder/multi30k`.

Copy all files to your working directory. The dataset has been already split into training, validation and test set. You should therefore have four training files, four validation files and four test files. Decide which of the four languages you will use as your target language. You should be able to judge the correctness of the target output. You will use the three remaining languages as source language and will train three translation systems: source1 $\rightarrow$ target, source2 $\rightarrow$ target, and source3 $\rightarrow$ target.

To end your session at the end of your work use `exit`.

2 Pre-processing

As first step, you need to preprocess all your data. The preprocessing procedure usually consists of the three following steps:

- The **normalization** step simplifies the punctuation of the text by using re-writing rules (e.g., remove extra spaces, normalize different types of quotation marks, etc.).
- The **tokenization** step splits the sentences in individual tokens, deciding the boundaries of each word (for example detaching a comma from a word).
- The **truecasing** step defines the proper capitalization of words.

The scripts for preprocessing are available in the folder `shared_folder/preprocessing-tools`.

The following command normalizes the punctuation of the Czech validation data:

```
normalize-punctuation.perl -l cs < val.cs > val.norm.cs
```

Repeat this command for all twelve files. The `-l` parameter expects the two-letter code of the language the data is written in.

The following command tokenizes the Czech validation data:

[1] https://github.com/OpenNMT/OpenNMT-py

[2] `https://github.com/multi30k/dataset/tree/master/data/task1/raw`

```
tokenizer.perl -l cs < val.norm.cs > val.tok.cs
```

Run the command for all twelve files, again setting the `-l` parameter according to the language. Inspect the resulting files to make sure that they correspond to your expectations..

Question: What happens if you set the `-l` parameter wrongly, e.g. tokenize the German data with the English settings?

Once all the files are tokenized, we first need to train a truecaser model on the training data of each language. The Czech truecasing model is trained with the following command:

```
train-truecaser.perl -corpus train.cs -model truecasing.cs.model
```

Repeat this procedure for the three other languages.

We can now apply the truecasing models to all the tokenized data. The command for the Czech validation data is as follows:

```
truecase.perl -model truecasing.cs.model < val.tok.cs > val.true.cs
```

Run this command for all twelve files, choosing the appropriate truecasing model. Inspect the resulting files to make sure that they correspond to your expectations.

If everything worked correctly, you should have twelve *.true.* files. The training files should contain 29,000 lines each, the validation files 1,014 lines each, and the test files 1,000 lines each.

3 Data conversion

The OpenNMT-py training procedure expects all training and validation data in a binary format. The onmt_preprocess script performs this data conversion. This script is part of the OpenNMT-py distribution.

The preprocessing script for a Czech-to-English model looks like this (one single line):

```
onmt_preprocess -train_src train.tok.cs -train_tgt train.tok.en -valid_src val.tok.cs
-valid_tgt val.tok.en -save_data data-cs-en
```

Run this command for all three translation directions that you have chosen.

If everything worked fine, you should have the following files in your folder for each language pair:

```
data-lang1-lang2.train.0.pt
data-lang1-lang2.valid.0.pt
data-lang1-lang2.vocab.pt
```

4 Training

After the preprocessing step, we can finally training a NMT model for each language pair. The training procedures requires the access to an actual GPU.

The following command is used for training a NMT model:

```
onmt_train -data data-cs-en -save_model model-cs-en -train_steps 10000
-save_checkpoint_steps 2000 -valid_steps 2000 -global_attention none -input_feed 0
-dropout 0.0 -world_size 1 -gpu_ranks 0 -layers 1 -rnn_size 256 -word_vec_size 256
```

The parameters represent the following settings:

- `-data` and `-save_model` refer to the file names (without extensions) of the input data and the model files.
- `-train_steps`, `-save_checkpoint_steps` and `-valid_steps` determine how many passes through the training data the model makes, how many intermediate model it saves, and how often it applies the validation data. The training process takes a batch of sentences for each training step and uses them to learn how to translate. Every 2,000 steps, the toolkit will save a model on your working directory. The last one (10,000) in our case will be most likely the optimal model. There is no need to change these parameters for this exercise.
- `-global_attention`, `-input_feed`, `-dropout` are related to the model type. This time, we use a simple encoder-decoder model as seen in class. We will experiment with other model types later.
- `-world_size` and `-gpu_ranks` determine how many GPUs to use. These settings should not be changed.
- `-layers` determines the number of recurrent layers in the model. `-rnn_size` determines the number of dimensions of each recurrent layer, and `-word_vec_size` determines the number of dimensions of the embedding layer (the first, non-recurrent layer of the model). You can experiment with different parameter settings later.

Repeat the training process for all three translation directions.

5 Testing

After training we can finally test our models and evaluate them. The `OpenNMT-py` `onmt_translate` script takes text file as input and produces the translation in another text file.

Remember how we trained the models: The models have been trained on preprocessed data, so we also need to feed them with preprocessed test data, and we will also get the output in preprocessed form. Create another SLURM script and write the following command in it:

```
onmt_translate -model model-lang1-lang2_step_10000.pt
-src test_2016_flickr.true.lang1 -output pred_2016_flickr.true.lang2 -gpu 0
```

Check the files and look at the `pred_2016_flickr.true.lang2` file to make sure that it corresponds to your expectations.

You have to detruecase and detokenize the translation output. This can be done with a single command:

```
detokenizer.perl -u -l lang2 < pred_2016_flickr.true.lang2 >
pred_2016_flickr.detok.lang2
```

The `-u` parameter makes sure that the first letter of each sentence is uppercased, which corresponds to detruecasing.

Compute the BLEU score of your detokenized translation output (with respect to the reference translation) using one of the scripts of assignment 1. Fill the following table with your BLEU scores (you will only fill one column of the table, depending on the target language you have chosen):

Source lang.	Target language			
	EN	DE	CS	FR
EN				
DE				
CS				
FR				

Questions: How do the BLEU scores vary according to source language? Does the ranking correspond to your expectations? Look at the output of the three models. Do you spot any differences in translation between models? Can you say why? Remember that in this scenario the training and testing data are of the same type, the only difference is the source language.

6 Examining the training process

When training the translation models, you saved five intermediate models (2000, 4000, 6000, 8000, and 10000). In the preceding exercise, we used the last one for testing, assuming that it would be the best. Here, we look at the intermediate models to see how the translation quality evolves over time.

Choose one language pair for this exercise. Use the `onmt_translate` command to evaluate each of the intermediate models Save the outputs in different files.

Look at the output of the five models (no need to detokenize or to compute BLEU scores).

Questions: What do the translations look like with the first model? Can you spot clear improvements in translation quality over time?

7 Modifying the network parameters (optional)

Choose one language pair for this exercise and train new models, modifiying the following 4 parameters: `-layers`, `-rnn_size`, `-word_vec_size`. Only modify one parameter at the time. Test your new models on the test set, detokenize it and compute BLEU scores.

Questions: Can you spot any impact on translation quality? Which parameter influences translation quality most?

8 Submission

Submit a PDF file in Moodle by Wednesday June 9th, 11:00pm. You should document your progress in the assignment. Answer the given questions but also note your observations and impressions.