Shared task: Lab 1 – NMT baseline system

Maksym Del <maksym.del@ut.ee>   Mark Fishel <fishel@ut.ee>

18. September 2017

In this lab session, you will learn to do the following:

1. Use git & github for the shared task
2. Prepare parallel text corpus for MT
3. Train baseline MT model
4. Translate a document with OpenNMT-py and trained baseline model

At the end of this document, you will find guidelines on what to do next to successfully pass the **first milestone**.

Git and GitHub usage

We prepared a separate pdf instruction for you on this topic. You can find it in the course materials github repository.

Data preparation for MT

This tutorial assumes that we work with translation direction Estonian --> English.

In practice, you begin with some raw data in **parallel text format** (**parallel corpus**), or even with several datasets like this (**parallel corpora**) that come from different sources. In order to prepare data for NMT training, do the following:

1. Concatenate corpora to form one big text corpus
2. Split corpus into training, dev, and test sets
3. Tokenize all the sets
4. True-case all the sets
5. Filter out empty and strange sentence pairs from the training (and dev) set
6. Apple BPE to all the sets

In order to do steps 2-4 we use Moses scripts; for the step 5, we employ external Subword NMT software package. We put all the scripts needed in the OpenNMT-py's *tools* folder.

Enter *data* folder, and follow the steps below:

1. Concatenate all in-folder corpora to form one big text corpus:

```
cat raw/*.en > demo-all.en
cat raw/*.et > demo-all.et
```

2. Split corpus into training, dev, and test sets:

   - corpus should be shuffled before any splitting
   - you can use the Unix commands *paste, shuf, sed -n and cut*
   - use *command_name --help* to see info about the command
   - the sizes are not pre-defined, the test set has to be at least 50'000 tokens and the dev set -- twice less. For now, let us use 100 for testing and dev, and 8800 for training (to complete the lab faster)

```
# tab-join and shuffle et and en corpora
paste demo-all.{et,en} | shuf > mixed-data.both

# split corpus
sed -n 1,100p mixed-data.both | cut -f 1 > test.et
sed -n 1,100p mixed-data.both | cut -f 2 > test.en
sed -n 101,200p mixed-data.both | cut -f 1 > dev.et
sed -n 101,200p mixed-data.both | cut -f 2 > dev.en
sed -n 201,9000p mixed-data.both | cut -f 1 > train.et
sed -n 201,9000p mixed-data.both | cut -f 2 > train.en
```

3. Tokenize all the sets:

```
for f in {test,dev,train}.{en,et}
do
   ../OpenNMT-py/tools/tokenizer.perl < $f > tok-$f
done
```

Note: if you get message “Permission denied”, execute the line below:

```
chmod +x ../OpenNMT-py/tools/*.perl
```

4. True-case all the sets:

   - first train the true-casing models for both files:

```
../OpenNMT-py/tools/train-truecaser.perl --model en-truecase.mdl --
corpus tok-train.en

../OpenNMT-py/tools/train-truecaser.perl --model et-truecase.mdl --
corpus tok-train.et
```

- then do true-casing itself:

```
for lang in en et
do
   for f in {test,dev,train}.$lang
   do
      ../OpenNMT-py/tools/truecase.perl --model $lang-truecase.mdl <
tok-$f > tc-tok-$f
   done
done
```

5. Filter out empty and strange sentence pairs from the training set:

```
../OpenNMT-py/tools/clean-corpus-n.perl tc-tok-train en et cleaned-tc-
tok-train 1 100
```

Note: because OpenNMT-py serializes dev set as well, you will have to clean the dev set as well if training on big data.

6. Apple BPE to all the sets:

At this point, we represent our sentences as a sequence of words. However, if we train NMT model using words as basic units, we will face a couple of difficulties. First, because Seq2Seq NMT models output probability distribution over **vocabulary** of basic units, they can became very computationally inefficient when the size of the vocabulary is big. That is why we **cut vocabulary** to 10.000 – 100.000 most frequent words. However, since we use limited number of words, arises question of the efficiency of our vocabulary: model treats different forms of words as completely different words (e.g. “play” vs “played” vs playing). Note, that using stemming, as a preprocessing step for MT, probably will significantly decrease model performance.

Idea of **subword units** is one solution to the **open vocabulary** problem. Byte Pair Encoding (BPE) is one technique to learn subword units. For details about subword units segmentation see here.

- first learn BPE:

```
cat cleaned-tc-tok-train.et cleaned-tc-tok-train.en
|/gpfs/hpchome/your_username/shared-task-2017/demo/OpenNMT-
py/tools/subword-nmt/learn_bpe.py -s 10000 > eten.bpe
```

- then apply BPE:

```
for lang in et en
```

```
  do
# apply to train
     /gpfs/hpchome/your_username/shared-task-2017/demo/OpenNMT-
py/tools/subword-nmt/apply_bpe.py -c eten.bpe < cleaned-tc-tok-
train.$lang > bpe.cleaned-tc-tok-train.$lang

# apply to dev
      /gpfs/hpchome/your_username/shared-task-2017/demo/OpenNMT-
py/tools/subword-nmt/apply_bpe.py -c eten.bpe < tc-tok-dev.$lang >
bpe.tc-tok-dev.$lang

# apply to test
     /gpfs/hpchome/your_username/shared-task-2017/demo/OpenNMT-
py/tools/subword-nmt/apply_bpe.py -c eten.bpe < tc-tok-test.$lang >
bpe.tc-tok-test.$lang

done
```

At this point, we have our BPE segmented corpora ready to be fed to the NMT end-to-end model.

Training NMT model with OpenNMT-py

Firstly, login to some gpu server (falcon1 or falcon2). Secondly, we have to load PyTorch and other packages required to run OpenNMT-py. In order to do this, execute following command from shell:

```
module load python-2.7.13
```

Then enter *OpenNMT-py* folder, and follow the steps below:

7. Build vocabulary from the training set and covert it into PyTorch format:

```
python preprocess.py -train_src ../data/bpe.cleaned-tc-tok-train.et -
train_tgt ../data/bpe.cleaned-tc-tok-train.en -valid_src
../data/bpe.tc-tok-dev.et -valid_tgt ../data/bpe.tc-tok-dev.en -
save_data ../data/rdy -src_vocab_size 10000 -tgt_vocab_size 10000 -
seed 123
```

We use validation data to tune model hyper-parameters and evaluate training progress.

After execution of this command, the following files are generated:

demo.src.dict: dictionary of source vocabulary *demo.tgt.dict:* dictionary of target vocabulary

demo.train.pt: serialized PyTorch file containing vocabulary, training and validation data

8. Train the model:

First, run following command to see gpu load:

```
nvidia-smi
```

Look for some free gpu, take its index (see first column), and run the command below with the -gpuid parameter specified:

```
python train.py -data ../data/rdy -save_model ../model/demo-model –
gpuid id
```

It runs default model with 2-layer LSTM and 500 hidden units on both encoder and decoder. We will talk about model hyper-parameters in details later in this course.

NB! Regarding running this (and other) commands directly from shell: do not try it at home. We broke this rule on this lab exceptionally for the educational purposes. HPC Center of UT do not allow running processes without the queue system on the new GPUs. This is to prevent people from accidentally destroying other people's jobs. If such processes are found on the nodes, they will be killed immediately, and you will not be able to finish your baseline model on big data. Instead, you should use SLURM (see the end of the document for more details). It is also helpful, since usually you want to run big processes in background and save commands in .sh files anyway.

You will see how validation perplexity changes. OpenNMT-py regularly makes model checkpoints (in the model directory).

9. Translate:

Now you have a trained model, and you should use in order to perform inference (decoding/translation) of the source sentences. We take the model with the best validation perplexity.

```
python translate.py -model ../model/demo-
model_acc_34.27_ppl_107.99_e13.pt -src ../data/bpe.tc-tok-test.et -
output ../data/hyps.en -gpu ind
```

10. Evaluate

You might notice, that translations in data/hyps.en are poor. It is due to the small amount of parallel corpus we used. At home, you will have a change to train a model almost on all available et-en data. Anyway, let us quantitatively evaluate how bad we are with our translations. From the *data* folder execute:

```
perl ../OpenNMT-py/tools/multi-bleu.perl bpe.tc-tok-test.en < hyps.en
```

You should get about 10 BLEU points. For the actual baseline model, you will get at least thrice as much. You will learn more about evaluation scores and error analysis in future during this class.

Training your baseline on big data

In this section, we provide you with guidelines on what to do in order to pass your first milestone.

Firstly, enter *your project repository data folder*, and take the actual raw data you will train your model on (~20.000.000 sentences):

```
cp –r /gpfs/hpchome/maksym95/shared-task-2017/data/* .
```

Then all you have to do is to repeat the tutorial above (prepare data, train the model, translate the sentences, and report results).

However, there are few **important aspects** you have **to follow**:

1) Use SLURM manager in order to run all your GPU and CPU tasks (e.g. subword splitting, model training, or inference). Please study and use it appropriate way. All the information can be found here, and in particular here. (see also NB part of “8. Train the model”)
2) Create tasks and issues, and attach them to milestones on *github project board* so we know what you have done, are doing, and will be doing. It will influence how we grade your team (more info in the pdf from “Git and Github usage” section).
3) Whatever command you run (you do it with SLURM), specify all the parameters in a separate config file, one file per command. Usually, frameworks allow to specify config file in many places. It helps you, us, and your teammates know exactly what you ran, and improves reproducibility of your work. E.g. you should execute preprocess.py file from step 7 (build vocabulary) like this:

```
python preprocess.py –config preprocess-demo.cfg
```

where preprocess-demo.cfg is a text file containing all your hyper-parameters, one hyper-parameter per line.

4) Save all the commands you run into .sh files to keep track of what you run (you will have to do it anyway with SLURM).
5) Commit and push all your .sh and .cfg files to your team’s GitHub repository. Write meaningful commit messages. Commit frequently. It will influence how we grade you (however, sometimes you need only 1-2 commits for the milestone, so do not overdo here).
6) In order to submit, just push report in pdf or .md format to the folder *milestone1* from the *results* directory (in your team’s repository). See a separate pdf file from “Git and GitHub usage” section, on detains on how to submit.
7) Model trains for about a week, so start early, keep calm, and good luck.