Lab on MT Data Processing, Evaluation and NMT Systems

Machine Translation – Summer 2023

June 14, 2023

1 Getting Started

The purpose of this lab is to train and evaluate a neural machine translation system (NMT) based on OpenNMT.[1,2] We will use the pytorch version (OpenNMT-py) and train a system in Google Colab[3] using a python notebook. If you are familiar with GPU clusters and have one of them available you can do the same (without the Colab restrictions) there. Notice that this is only a class exercise. In SMT, Moses was used for both research and industry. In NMT, nobody would use a notebook on Colab as we will do here. Marian (especially adequate for industry), Fairseq, Sockeye or even OpenNMT are good options but they need to be installed in a GPU cluster.

We will use the same data pre-processing and evaluation software as in the PB-SMT lab with only one addition as neural models work with subunits as basic tokens. In the following, the parts in grey are exactly the same we did in the first lab. You can reuse the first steps we did to prepare the corpus.

Before coming to the first session, please explore Google Colab and prepare your account. Also download or install Subword-NMT[4] for BPE tokenisation.

2 (Neural) Translation Engine for Literature

2.1 Getting the data

Parallel corpora can be found in evaluation campaigns' web sites (WMT, NIST, IWSLT, etc.) as we have seen in the lectures or the open parallel corpus OPUS site. Explore the latter:

```
http://opus.lingfil.uu.se/
```

For this session, we will use the English–French edition of the *Books collection*, a set of copyright-free books aligned by Andras Farkas. Download them (download plain text files, MOSES/GIZA++), extract the files and explore the content:

```
machine:~/pln/teaching/literature$ wc -lw Books.en-fr.*
  127085  2715071 Books.en-fr.en
  127085  2642300 Books.en-fr.fr
  127085   531211 Books.en-fr.ids

machine:~/pln/teaching/literature$ tail -n 2 Books.en-fr.*
==> Books.en-fr.en <==
The corpses lay all night, spread out contorted, on the dining-room floor, ...
And for nearly twelve hours, in fact until the following day at about noon, Madame Raquin, ...

==> Books.en-fr.fr <==
Les cadavres restèrent toute la nuit sur le carreau de la salle et manger, ...
Et, pendant près de douze heures, jusqu'au lendemain vers midi, Mme Raquin, ...
```

[1] https://opennmt.net/
[2] https://github.com/OpenNMT/OpenNMT-py
[3] https://colab.research.google.com/
[4] https://github.com/rsennrich/subword-nmt

```
==> Books.en-fr.ids <==
en/Zola_Emile-Therese_Raquin.xml.gz fr/Zola_Emile-Therese_Raquin.xml.gz s1246.0 s1246.0
en/Zola_Emile-Therese_Raquin.xml.gz fr/Zola_Emile-Therese_Raquin.xml.gz s1246.1 s1246.1
```

The ids file tells us to which novel corresponds each parallel fragment. Use this information to extract all the fragments corresponding to *The Great Shadow* by Arthur Conan Doyle. We will use this novel for in-domain tuning and testing purposes (top 1000 lines for testing, bottom 859 for tuning). All the other novels (remove this one!) are used for training.

```
machine:~/pln/teaching/SMT/corpus$ grep Shadow Books.en-fr.ids | wc -lw
   1857    7921

machine:~/pln/teaching/SMT/corpus$ wc -lw Books.*.??
  125226  2674214 Books.train.en
  125226  2600120 Books.train.fr
    1000    22930 Books.test.en
    1000    23551 Books.test.fr
     859    17927 Books.dev.en
     859    18629 Books.dev.fr
```

Let's go back to OPUS and download also *WMT-News*, a parallel corpus of News Test Sets provided by WMT for testing purposes. The last 3002 lines correspond to the test set of 2014 (the last year that included the English–French pair). We will use this data to test our system in an out-of-domain framework.

```
machine:~/pln/teaching/SMT/corpus$ wc -lw news2014.test.*
  3002  62333 news2014.test.en
  3002  68148 news2014.test.fr
```

2.2 Pre-processing the Data

This is probably the most important part of the process that fully depends on you. Look carefully at the data. Which is the encoding of the file? Text is homogeneous? Same punctuation marks? Strange characters? In our case, we are using data that has already been prepared for MT, we are in a controlled experiment, but this is not the usual case. Get used to doing the whole process!

We are using Moses, which has its own scripts for **cleaning** the texts. Look at the folder `mosesdecoder/scripts/tokenizer`. Using the next scripts (both source and target) never harms:

```
perl replace-unicode-punctuation.perl < set.en > set.norm1.en
perl normalize-punctuation.perl -l en < set.norm1.en > set.norm2.en
perl remove-non-printing-char.perl < set.norm2.en > set.norm3.en
```

where *set* applies to the training, development and the two test sets.

The final file is ready to be **tokenised**. In this lab we will use a classical tokeniser. You will learn about BPE, SentencePiece and other types of subunit tokenisation later in the course and we will use them in the second lab. These are different approaches that appear from different needs (size of the vocabulary!). The (language-dependant) classical tokenisers are available as moses scripts too:

```
perl tokenizer.perl -l en -no-escape -threads 4 < set.norm3.en > set.tok.en
```

Optionally, we can **truecase** the text or, at least, lowercase it. Moses also has tools for training a truecaser and a recaser. Look at the folder `mosesdecoder/scripts/recaser`.

```
perl train-truecaser.perl --model tcModel.Books.en --corpus Books.train.tok.en
```

Unless there is a reason, you should train the truecaser with large monolingual corpora. Here, we will only use the two sides of our parallel corpus. Look at the model: it is a frequentist list of the tokens with their capitalisation in our corpus:

```
machine:~/pln/teaching/SMT/corpus/truecaser$ more tcModel.Books.en
  patronne (1/1)
  Parvis (24/28) parvis (4)
  quarreling (6/6)
  ...
```

After training one model for each language, we want to apply it to all the tokenised sets:

```
perl truecase.perl --model ./truecaser/tcModel.Books.en < set.tok.en > set.tc.en
```

What we have done until now is word tokenisation, each word is a basic translation unit, a token. In neural systems, one can only learn a limited number of vocabulary tokens, initially because of restricted hardware resources (GPU memory). Those tokens not available in the vocabulary would be marked as "unknown" or "unk" and, therefore, they would not have a translation. To solve this issue, instead of working with words as basic tokens, we will work sub-words. That is, we tokenise the input further, a word such as `quarreling` will be converted into `quarr@@ el@@ ing` for example. How to split words is also learned (in a frequentist way!) from the training data.

In this exercise, we will use *Byte-Pair-Encoding* (BPE) as implemented by Ricoh Sennrich in Subword-NMT. BPE starts from single characters and merges the most frequent pairs of characters in a corpus to create a new subunit. The process is repeated until the desired number of merge operations are performed. With BPE, common words form a single unit while rare words are split into subunits as seen above.

Let's install Subword-NMT (you can also simply download it as these are python scripts):

```
pip install subword-nmt
```

Read the explanations in the Subword-NMT repo[5] to fully understand how it works. We will use their *best practice advice* to bpe our data:

```
subword-nmt learn-joint-bpe-and-vocab --input {train_file}.L1 {train_file}.L2   \
    -s {num_ops} -o {codes_file} --write-vocabulary {vocab_file}.L1 {vocab_file}.L2

subword-nmt apply-bpe -c {codes_file} --vocabulary {vocab_file}.L1   \
    --vocabulary-threshold 50 < {file}.L1 > {train_file}.BPE.L1
subword-nmt apply-bpe -c {codes_file} --vocabulary {vocab_file}.L2   \
    --vocabulary-threshold 50 < {file}.L2 > {train_file}.BPE.L2
```

which in our case corresponds to first training the joint model:

```
subword-nmt learn-joint-bpe-and-vocab --input Books.train.tc.en Books.train.tc.fr   \
     -s 32000 -o bpe/model.bpe --write-vocabulary bpe/vocab.en bpe/vocab.fr
```

and afterwards applying the learnt model (`model.bpe`) to the training, development and test sets. This example shows how to apply it to the training data, the same applies for the other sets.

```
subword-nmt apply-bpe -c bpe/model.bpe --vocabulary bpe/vocab.en   \
     --vocabulary-threshold 50 < Books.train.tc.en > Books.train.bpe.en
subword-nmt apply-bpe -c bpe/model.bpe --vocabulary bpe/vocab.fr   \
     --vocabulary-threshold 50 < Books.train.tc.fr > Books.train.bpe.fr
```

Let's look how a segment of the training data looks like once it is bpe'd:

```
machine:~/pln/teaching/NMT/corpus$  head Books.train.bpe.en
...
we left that part of the country nearly fifteen years ago and shall certainly never
go back to it .
we were living in the building of the H@@ i@@ gh@@ er E@@ l@@ em@@ ent@@ ary C@@ l@@
as@@ ses at S@@ ain@@ te@@ -@@ A@@ g@@ a@@ the 's S@@ cho@@ o@@ l .
my father , whom I used to call M. Seurel as did other pu@@ p@@ ils ,
...
```

[5] https://github.com/rsennrich/subword-nmt

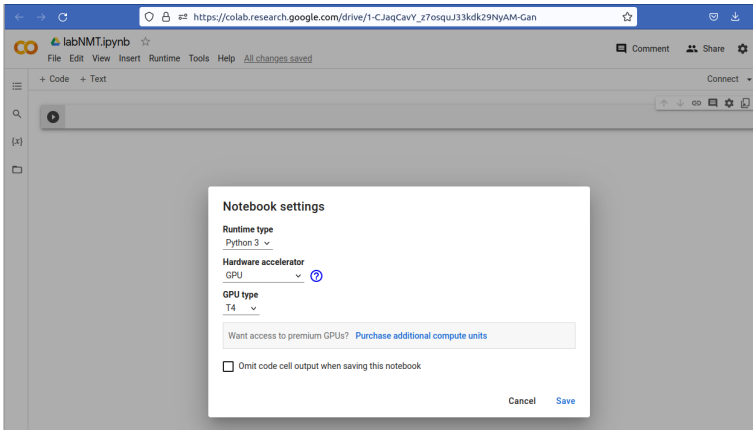
And we are done with the (new!) pre-processing. This is the corpus we are going to use for training, tuning and testing our translation engine:

```
machine:~/pln/teaching/NMT/corpus$  ls *bpe.??
Books.dev.bpe.en  Books.test.bpe.en  Books.train.bpe.en  news2014.test.bpe.en
Books.dev.bpe.fr  Books.test.bpe.fr  Books.train.bpe.fr  news2014.test.bpe.fr
```

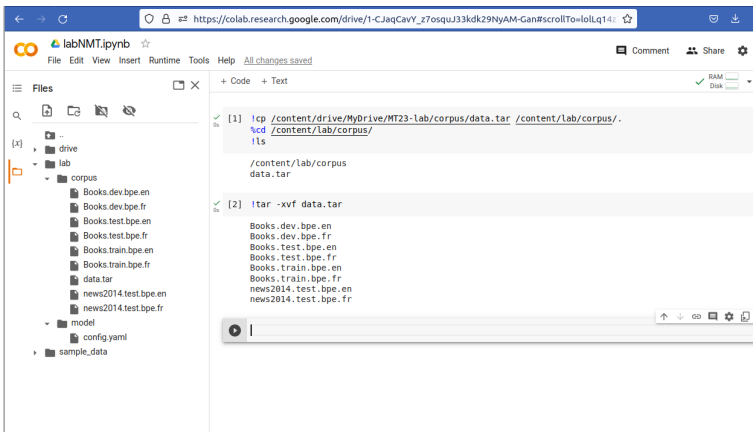
2.3 Training

We will use Colab for training. Colab is a Google interface that allows to run python and bash commands using Jupyter-like notebooks. The main advantage is that everything is installed on their side and GPUs are available. The main disadvantage is that a session lasts at most 12 hours and everything that is not downloaded is lost once the session is closed. So, remember to save your data and/or models to Google Drive or to your machine and take into account that for more extensive experiments you should not use notebooks and you should install your preferred NMT system in a server with GPUs. Some NMT systems such as Marian NMT allow translation in CPUs too.

First, go to Colab using your Google account, create a new notebook, give it a name and *very important*, go to a runtime environment with GPUs: Runtime menu > Change runtime type: Hardware accelerator with GPU:



Now, let's import the final result of our pre-processing steps. Open the Files box on the left, mount your Google Drive in case your data is there or upload the data (training, dev, tests and config file) from your computer. In the example below, Google Drive is mounted on `/content/drive` (the default) and the folders for this lab have been created in `/content/lab`:



An NMT training is very different to what we did with the SMT system. SMT is a combination of modules: language model, alignments, translation model, tuning... Everything is learnt from data,

there is no hyperparameter tuning and once the pipeline is built, there is almost no need for human (your!) intervention. NMTs are end-to-end systems, there is no pipeline. However, these systems have lots of hyperparameters (batch size, learning rate, dropout, etc) that need to be explored and in this case human (your!) expertise is needed.

OpenNMT summarises all the hyperparameters and functionalities needed for training the system in a config file in YAML format. Look at our example:

```
# config.yaml

## Where the samples will be written
save_data: run

# Training files
data:
    corpus_1:
        path_src: /content/lab/corpus/Books.train.bpe.fr
        path_tgt: /content/lab/corpus/Books.train.bpe.en
        transforms: [filtertoolong]
    valid:
        path_src: /content/lab/corpus/Books.dev.bpe.fr
        path_tgt: /content/lab/corpus/Books.dev.bpe.en
        transforms: [filtertoolong]

# Vocabulary files, generated by onmt_build_vocab
src_vocab: run/source.fr.vocab
tgt_vocab: run/target.en.vocab

# Vocabulary size - should be the same as in BPE/sentence piece
src_vocab_size: 32000
tgt_vocab_size: 32000

# Filter out source/target longer than n if [filtertoolong] enabled
src_seq_length: 150
src_seq_length: 150

# Tokenization options
src_subword_model: source.model
tgt_subword_model: target.model

# Where to save the log file and the output models/checkpoints
log_file: train.log
save_model: /content/lab/model/model.fr2en

# Stop training if it does not improve after n validations
early_stopping: 4

# Default: 5000 - Save a model checkpoint for each n
save_checkpoint_steps: 500

# To save space, limit checkpoints to last n
keep_checkpoint: 5

seed: 3435

# Default: 100000 - Train the model to max n steps
# Increase to 200000 or more for large datasets
# For fine-tuning, add up the required steps to the original steps
train_steps: 3000

# Default: 10000 - Run validation after n steps
valid_steps: 500
```

```
report_every: 100

# Number of GPUs, and IDs of GPUs
world_size: 1
gpu_ranks: [0]

# Batching
bucket_size: 262144
num_workers: 0  # Default: 2, set to 0 when RAM out of memory
batch_type: "tokens"
batch_size: 4096   # Tokens per batch, change when CUDA out of memory
valid_batch_size: 2048
max_generator_batches: 2
accum_count: [4]
accum_steps: [0]

# Optimization
model_dtype: "fp16"
optim: "adam"
learning_rate: 2
warmup_steps: 1000     # Default: 4000 - for large datasets, try up to 8000
decay_method: "noam"
adam_beta2: 0.998
max_grad_norm: 0
label_smoothing: 0.1
param_init: 0
param_init_glorot: true
normalization: "tokens"

# Model
encoder_type: transformer
decoder_type: transformer
position_encoding: true
enc_layers: 6
dec_layers: 6
heads: 8
hidden_size: 512
word_vec_size: 512
transformer_ff: 2048
dropout_steps: [0]
dropout: [0.1]
attention_dropout: [0.1]
```

These values correspond to a standard transformer seq2seq model with some adaptations[6] to take into account that we are training with a small corpus. Download the template config.yaml from the course webpage, fill the missing values and upload it to Colab, we will use it for training.

Before starting the training itself, we need a last detail, we need to build the vocabulary in the appropriate format for OpenNMT (tokens with their frequency). We do this in Colab from the folder model:

```
# Build Vocabulary
# -config: path to your config.yaml file
# -n_sample: use -1 to build vocabulary on all the segment in the training dataset
# -num_threads: change it to match the number of CPUs to run it faster
!onmt_build_vocab -config /content/lab/model/config.yaml -n_sample -1 -num_threads 2
```

We go to the folder where the vocabs are stored (as requested in config.yaml) to see how they look.

[6] `train_steps`: for datasets with a few millions of sentences, consider using a value between 100000 and 200000, or more!; `valid_steps`: 10000 can be good if the value train_steps is big enough; and `warmup_steps`: try 4000 and 8000 values for large datasets.

```
!head -n 50 target.en.vocab

,211247
the 141567
. 114430
" 73543
of 69392
and 67515
...
not 16825
s@@ 16493
! 15394
my 15062
...
which 13687
t@@ 13624
this 13159
```

We are ready to run the training with the command `onmt_train` and the configuration file. With the configuration above, this will take 35 minutes.

```
!onmt_train -config config.yaml
```

The output of this step is the model, `model.fr2en_step_3000.pt` in our case. You will find a model after every 500 steps as this is indicated in the config file, and only the last 5 models have been saved. The `train.log` file created also in the model folder describes the network used for training but not the evolution during training (perplexity, accuracy, etc.) that is shown in Colab in the stdout. Copy the stdout in a file, it will be useful later.

2.4 Decoding

We can now use the model trained above to translate our test sets. We can do this with the command `onmt_translate`:

```
!onmt_translate -model /content/lab/model/model.fr2en_step_3000.pt   \
    -src /content/lab/corpus/Books.test.bpe.fr  \
    -output /content/lab/trads/Books.testTrad.bpe.fr2en.en -gpu 0 -min_length 1

!onmt_translate -model /content/lab/model/model.fr2en_step_3000.pt   \
    -src /content/lab/corpus/news2014.test.bpe.fr   \
    -output /content/lab/trads/news2014.testTrad.bpe.fr2en.en -gpu 0 -min_length 1
```

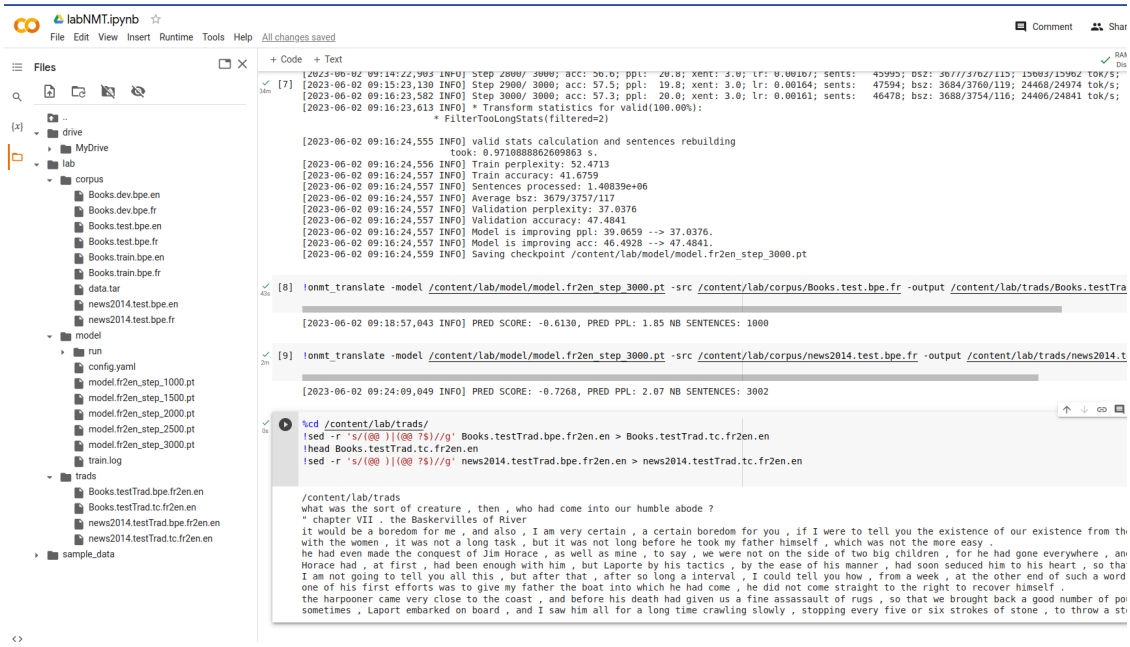
These translations are bpe'd, truecased, and tokenised:

```
!head -n2 Books.testTrad.bpe.fr2en.en
what was this sort of love which had come to lay in our humble d@@ re@@ ss@@ ing-@@ room ?
" chapter V@@ I@@ I@@ I . the G@@ over@@ nor
```

We need to undo all these steps to convert the text into "human-written" text before being evaluated. The only step that differs from the SMT lab is that first we need to remove the `@@` introduced by the BPE model. A simple `sed` command does the job:

```
!sed -r 's/(@@ )|(@@ ?$)//g' Books.testTrad.bpe.fr2en.en > Books.testTrad.tc.fr2en.en
!sed -r 's/(@@ )|(@@ ?$)//g' news2014.testTrad.bpe.fr2en.en > news2014.testTrad.tc.fr2en.en
```

At this point we download everything produced in Colab (either directly or copying it first to your Google Drive) because from now on we can reuse the post-processing and evaluation pipelines that we did in the SMT lab. The Colab folder will look something similar to:



You need to download the models and `train.log` (as you will need them again to complete the questionnaire) and the `trads` folder. It is advisable that you download the notebook itself for reference, you will have to repeat some of the steps.

Our translations are still truecased and tokenised, so we go to the `trads` folder in your computer and call Moses scripts for post-processing as we did before. Detruecase and detokenise (do it for both test sets):

```
perl  /home/cristinae/soft/mosesdecoder/scripts/recaser/detruecase.perl <
      Books.testTrad.tc.fr2en.en > Books.testTrad.tok.fr2en.en

perl  /home/cristinae/soft/mosesdecoder/scripts/tokenizer/detokenizer.perl -en <
      Books.testTrad.tok.fr2en.en > Books.testTrad.fr2en.en
```

2.5 Evaluation

Let's finally evaluate your system. This is an important aspect very difficult in MT. Nothing can really substitute a manual evaluation of your translations, but this is a very expensive evaluation (both in time and money) that cannot be used during the development of a system. Instead, automatic evaluation is used.

We will use `sacreBLEU`.[7] In their own words "SacreBLEU (Post, 2018) provides hassle-free computation of shareable, comparable, and reproducible BLEU scores." In fact, it also implements chrF and TER; and statistical significance tests!

Install it, this one is fast:

```
pip install sacrebleu
```

and look at all the options with `sacrebleu -h`. We will calculate the three metrics with 95% confidence intervals via bootstrap resampling.

```
sacrebleu ../corpus/Books.test.en -l fr-en -i Books.testTrad.fr2en.en -m bleu chrf ter
          --confidence -f text

BLEU|nrefs:1|bs:1000|seed:12345|case:mixed|eff:no|tok:13a|smooth:exp|version:2.2.0
```

[7] `https://github.com/mjpost/sacrebleu`

```
    = 18.1 (μ = 18.0 ± 0.8) 53.2/24.6/12.4/6.6
    (BP = 0.999 ratio = 0.999 hyp_len = 26435 ref_len = 26469)

chrF2|nrefs:1|bs:1000|seed:12345|case:mixed|eff:yes|nc:6|nw:0|space:no|version:2.2.0
    = 40.0 (μ = 40.0 ± 0.6)

TER|nrefs:1|bs:1000|seed:12345|case:lc|tok:tercom|norm:no|punct:yes|asian:no|version:2.2.0
    = 71.5 (μ = 71.5 ± 1.0)
```

We are below the PB-SMT system! Remember: BLEU $= 19.1 \pm 0.8$, chrF2 $= 43.4 \pm 0.6$, TER 69.1 ± 1.1. We can do better. The system has not converged for example, and the hyperparameter are just a first guess, no manual intervention. You will do a few more things in the Questionnaire.

As before, this works because you only have a system, but confidence intervals are not really informative. In general, you would have a baseline and several variants of your system. And now you have, the PB-SMT and the NMT translations are two variants of your fr2en translation system. In that case, p-values are good indicators of difference in quality among systems. `SacreBLEU` allows you to obtain confidence intervals together with p-values via paired bootstrap resampling. Use the `--paired-bs` option for this instead of `--confidence`.

Keep in mind that BLEU/chrF are lexical metrics that count the number of n-gram matches. Two translations can be perfect, or at least convey the same meaning, without having any n-gram in common. Calculating 3 metrics instead of only one is already a first approach to minimise the impact. Using test sets with multiple references is another (expensive) approach. But whenever it is possible (may not be available in the language you need), use also semantic metrics.

Let's install COMET[8] for this:

```
pip install unbabel-comet
```

The first time you run it, COMET will download the model (2.32G). Run it to obtain the score at system level using:

```
comet-score -s ../corpus/Books.test.fr -t Books.testTrad.fr2en.en
            -r ../corpus/Books.test.en --quiet --only_system
```

Disclaimer: This is slow and might need more resources than the ones in a standard laptop. You can install COMET in Colab and evaluate your translations there. Take into account that doing this will also consume your free GPU time, so wait until the end just in case.

Unfortunately, COMET does not provide statistical significance tests yet, they have to be implemented independently and we are not doing this here.

Your engine has been optimised for translating novels. Since the corpus is small and only this kind of text has been used for training, other genres will have really low quality. What happens with `news2014.testTrad.fr2en.en`? You cannot compare scores across different test sets, but the values can give you an idea. You will find online in several papers how big MT systems perform on the news2014 test set.

[8] `https://unbabel.github.io/COMET/html/index.html`

3 Questionnaire

1. Look for the following sentence in your training corpus and write its output after the BPE tokenisation:
   `When he was gone, I turned to the boy, whom they called Xury, and said to him, "Xury, if you will be faithful to me, I'll make you a great man; but if you will not stroke your face to be true to me"-that is, swear by Mahomet and his father's beard-"I must throw you into the sea too."`

2. We have used BPE to obtain subunits, but there are other methods available. SentencePiece[9] for instance takes a different approach that allows to get rid of the language dependant word tokenisation that we have done until now. Explain how SentencePiece works and comment the differences with BPE as implemented in Subword-NMT.

3. Install SentencePiece, train it with your raw clean training data (`Books.train.norm3.en` and `Books.train.norm3.en`) and apply it to the training, development and test sets. Write the sentence of Question 1 after the process.

4. Translate *Books.test.fr* and *news2014.test.fr* into English with the system trained in the lab. We call this system the baseline NMT system. Recover the translations from the PB-SMT lab (they are also available in the course webpage). Use sacreBLEU and COMET to calculate scores for the translations of both systems. When using sacreBLEU, use the `--paired-bs` option to obtain p-values and confidence intervals. Report the results. Is the baseline NMT system significantly better than the PB-SMT one?

5. Let's improve the baseline NMT system (I). After 3000 steps, the system has probably not converged yet (you don't see a minimum in perplexity). Extend the training until convergence. This implies increasing the number of steps and setting an appropriate `early_stopping` value. Plot the evolution of the accuracy and perplexity in validation (seen in the stdout during training) along the whole training.

6. Let's improve the baseline NMT system (II). Ensemble Decoding. During translation (not training!), instead of adding one model/checkpoint to the `-model` argument, add multiple checkpoints. For example, try the three last checkpoints. Does it improve quality of translation? —The evaluation here is the same as in Question 10 for this system.— Does it affect translation speed?

7. What's the Adam optimiser? Compare it briefly to AdamW.

8. What is *dropout* and why is it useful? Where (in which parts of the transformer) is applied according to the parameters you use in the baseline system?

9. Let's improve the baseline NMT system (III). Hyperparameter tuning. We have used reasonable values for all the parameters in the config file, but these parameters should be tuned for every system. Try to explore changes to significant parameters such as the size of the vocabulary, the learning rate, the optimiser, the number of warmup steps, the size of the transformer (layers, dimensions, etc). Or train a system with the SentencePieced corpus instead of the BPEd one. You won't be able to do an exhaustive exploration on Colab, but try at least 3 changes with respect to the baseline system. Report the configurations you explore with their corresponding accuracy and perplexity in validation.

[9] `https://github.com/google/sentencepiece`

10. Extend the table of Question 4 with the translations obtained with the improved systems of questions 5, 6 and 9. As before, use sacreBLEU with the `--paired-bs` option to obtain p-values and confidence intervals. Is the best NMT system now significantly better than the PB-SMT one?

11. Compare and comment the results obtained by the PB-SMT system and the best NMT one. Find examples in the translation outputs that show the characteristics of SMT and NMT systems. Justify the answer.

4 Submission

You have two weeks from the last day of the lab to write the report. Give it to us with

- your name and matriculation number in EVERY page
- questions 1 and 4 starting a NEW page

And please, be CONCISE, get to the point. Don't make the text unnecessarily long.