Assignment 3: Machine Translation

This assignment accompanies the lectures on machine translation in the Advanced Natural Language Processing course. Its goals are as follows:

- Familiarise yourself with training a neural machine translation (MT) system.
- Replicate MT experiments from a recent academic paper.

1 MT System

In this exercise, you will replicate some of the experiments in one of the submissions to the shared task on Unsupervised MT and Very Low Resource Supervised MT at WMT 2022. We are targeting the *Low Resource Supervised Translation* part of the task. The task is about translation between Upper and Lower Sorbian, two Slavic minority languages spoken in Eastern Germany.

This assignment runs over two weeks, and it is up to you to manage your time. Even though the MT system you will train is small, a full training run takes several hours, and you should also expect some congestion on the HPC cluster if all of you launch your experiments at the same time. I would therefore suggest that you prioritise setting up the training runs so that they can run while you are busy with other parts of the exercise, and between the exercise sessions. You are also welcome to talk to each other to coordinate experiments and share your experience.

- ► Start by looking at the general description of the task:
  `https://www.statmt.org/wmt22/unsup_and_very_low_res.html`

Specifically, your task is to replicate the experiments on translation from Upper Sorbian from Lower Sorbian described in the following paper:

> Edoardo Signoroni and Pavel Rychlý (2022). MUNI-NLP Systems for Lower Sorbian-German and Lower Sorbian-Upper Sorbian Machine Translation @ WMT22. In: Proceedings of the Seventh Conference on Machine Translation (WMT), Abu Dhabi, pp. 1111–1116.
> `https://aclanthology.org/2022.wmt-1.109/`

We only focus on the language pair Upper Sorbian (hsb) to Lower Sorbian (dsb). There are some problems with the German-Sorbian datasets for the shared task, so I don't recommend working with those.

Note: Even though the system setup is quite straightforward, it's not so easy to replicate the published results exactly. If you can't quite get there with your baseline, decide for yourself if it's more interesting to continue improving your baseline, potentially at the cost of leaving out some of the later exercises, or move on to the next part with a less-than-perfect baseline.

Like Signoroni and Rychlý, we work with the sequence-to-sequence modelling toolkit `fairseq`. On the ITU HPC cluster, there is a Anaconda Python environment that includes all the packages you need for this exercise. Setting up a working environment was not entirely straightforward, so I strongly recommend you work with the one we we've provided. Here's how to activate it:

```
module purge
module load Anaconda3
source activate /opt/itu/condaenv/cs/anlp2023-mt
```

Note: Loading the environment can take a very long time on the `front` node. Try logging in to a computational node instead.

Please read the documentation of fairseq and subword-nmt to find out what arguments to pass to the two tools:
`https://fairseq.readthedocs.io/en/latest/index.html`
`https://github.com/rsennrich/subword-nmt`

2 Data Preparation and Baseline Training

- Download and unpack the hsb-dsb datasets from the shared task website linked above. You will need the "Parallel Upper Sorbian ↔ Lower Sorbian Data". The hsb (Upper Sorbian) and dsb (Lower Sorbian) parts of the training corpus are in two separate files. The validation and test sets are in a tarball.
- Train BPE vocabularies with 4000 operations on the training data using `subword-nmt` and apply those to all datasets. (4000 operations will give you a vocabulary of a bit more than 4000 words. That is okay.)
- Using `fairseq-preprocess`, prepare the data for a baseline experiment with your BPE-processed data. You will need to pass the arguments `--source-lang`, `--target-lang`, `--trainpref`, `--validpref`, `--testpref` and `--destdir`. Other arguments you may find useful include `--bpe`, `--bpe-codes`, `--srcdict` and `--tgtdict`.
- Train a baseline system using `fairseq-train`, following the description in Section 2.4 and Table 1 of the Signoroni paper. Start with `--arch transformer` and add arguments for the hyperparameters described in the paper. Looking at Table 2 of the paper, observe that the *t-bpe* configuration outperforms *t-opt-bpe* across all language pairs and directions, so you may want to aim for that. However, *t-bpe* training as described may require too much GPU memory, and you may consider taking inspiration from the smaller *t-opt-bpe* to make the model fit on the GPUs we have. In particular, reducing the batch size via the `--max-tokens` parameter has an immediate impact on GPU memory consumption (potentially at a cost in system performance).

- ► Use `fairseq-generate` to translate the test data with the best checkpoint found during training. The output of `fairseq-generate` contains a lot of information in one file. You can use
  `grep '^H' fairseq.out | cut -f3- >system.out` and
  `grep '^T' fairseq.out | cut -f2- >reference.out`, respectively, to extract the MT system output and the reference translation from fairseq's output. Note that the output of `fairseq-generate` isn't in order unless you sort it again by the sentence number in the first column.
- ► Decode the BPE encoding by replacing all occurrences of '`@@ `' with the empty string and use `sacrebleu` with default settings to compute the BLEU score of your translation.
- ► During training, keep an eye on the development of the training statistics: Training and validation loss and perplexity (ppl), in relation to the learning rate (lr). When you see large jumps in perplexity, you might try to investigate what happens by obtaining translations for the corresponding epochs and computing BLEU scores or inspecting the output. Even if, like me, you can't read Sorbian, you might still be able to recognise obvious problems in the MT output.

3 System Development

Signoroni and Rychlý describe two methods, with which they achieved (minor) improvements in system performance: High Frequency Tokenisation and Data Diversification.

- ► Read the description of both methods in the sections below and in Sections 2.1 and 2.2 of the Signoroni paper and then choose *one of the two methods* to implement in your system. You don't need to do the other one.

3.1 High Frequency Tokenisation

High Frequency Tokenisation (HFT) is a subword splitting method that the authors claim works better in low-resource scenario. If you pick this method, you need to implement the tokeniser and detokeniser and integrate it in your system by disabling the subword splitter in fairseq and using your own tool instead. You can tell fairseq to use your own tokenisation by passing the flag `--tokenizer space` to `fairseq-preprocess`.

- The HFT method is described in Section 2.1 of the Signoroni paper.
- I suggest leaving out the special character transformations (Figure 1 in the paper) initially and concentrating on the implementation of word splitting.
- Consider using `collections.Counter` to keep track of the subword counts you need to compute.

- As part of the algorithm, you need to find the best possible subword split of a word for a given vocabulary. You can use dynamic programming to compute this efficiently: Process the word letter by letter and store the best split for each position. For any given position, look at all the prefixes for which you've already computed a split and check if the remaining part that takes you to the current position is in the vocabulary.
- Even if you use dynamic programming, this part of the algorithm is rather slow. You can speed up the process by creating a dictionary containing the splits of words you've already seen.

► Implement HFT, train a new system and test how it performs with respect to your own baseline.

3.2 Data Diversification

Data Diversification is described in Section 2.2 of the paper. This method requires generating translations with several MT systems (or checkpoints) to increase the variety of the test set. If you decide to implement this method, I suggest you team up with some of your fellow students and share the translations of the baseline systems each of you independently trained, so that you don't need to train a whole bunch of MT systems for each of you. You could also use several checkpoints of one MT system, but make sure that you only include checkpoints with reasonable performance. Test by evaluating their BLEU score and comparing to that of your best checkpoint.

► Implement Data Diversification, train a new system and test how it performs with respect to your own baseline.