MT Solutions 5

Topics: Byte Pair Encoding, Beam Search

1 Experiments with Byte Pair Encoding (0.5 points)

You can find the necessary scripts and config files necessary to solve the exercise as well as two example modules under the branch solution of the GitHub repository under https://github.com/siri-web/mt-exercise-5.

The scripts assume the language pair EN-DE: your data might be different. Important insights:

- To test different BPE vocabulary sizes, it is necessary to train two different BPE models.
- You have to be careful to provide JoeyNMT with the correct BPE model and vocabulary, depending on the desired vocabulary size.
- The config files are very similar, except for the following points:
  - For the word-level model, there is no pre-computed vocabulary file. Instead, there is an upper limit of 2k words for both the source and target language, i.e. 4k words in total. Here we also apply moses as a pretokenizer, as there is no subword tokenization taking place. Additionally, for the word-level model the embeddings cannot be tied, as the source and target vocabulary differ.
  - For the bpe-level models, we instead specify the respective codes and vocabulary file, which were acquired by running the `learn_bpe.sh` script. Of course we also need to change the level from `word` to `bpe` as well as setting the `tokenizer_type` to `subword-nmt`.
- If you'd like to see the translation in BPE-format, the parameter `postprocess` can be set to `False` under `testing`. Normally, `JoeyNMT` will undo the BPE-splits internally automatically.

Our results in terms of BLEU are as follows:

	use BPE	vocabulary size	BLEU
(a)	no	2000	8.2
(b)	yes	2000	16.9
(c)	yes	4000	17.7

Observations:

- In our case, the BPE model with a 4k vocabulary performed best on our testset. This is in line with our expectation that one of the two BPE models would score best.
- Looking at the translations, the translation created with the word-level 2k model contains many `<unk>` symbols, while the translations created with the BPE models do not contain such symbols.
- Due to the many `<unk>` symbols in the translation, we can conclude that in a proper human evaluation, the word 2k model would certainly be rated as the worst model.
- When comparing 2k BPE model and the 4k BPE model manually, the 2k model seems to be performing slightly better. This shows that while BLEU scores are a good first indicator of the translation quality of a model, manual evaluation is still crucial and necessary.

Please note that the different ways in which we let Joey preprocess the data lead to a different number of effective updates to the model in two epochs. This is because we defined the batch size in terms of tokens, and if tokens are subwords there are naturally more batches that can be built with the same training data. For a serious academic experiment, this would need to be taken into account.

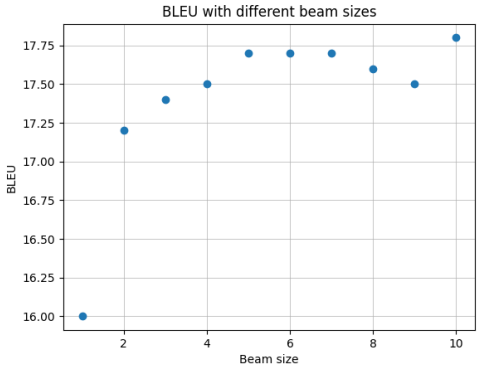
A second caveat is that, like in exercise 4, we are training models for very few updates only. This is quite unusual: usually, models are trained until their performance on the dev set does not improve anymore. If models are trained properly, the BLEU scores, translations and conclusions might be different.

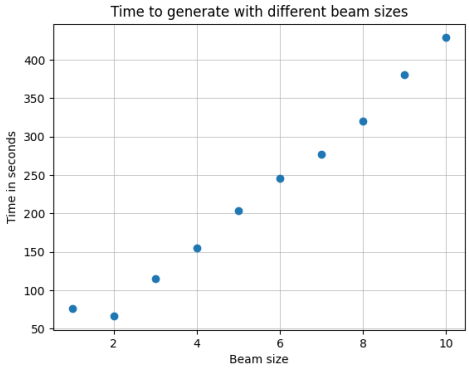
2 Impact of beam size on translation quality (0.5 points)

To translate with different beam sizes, we added the script `beam_size_bleu.sh`[1]

For each beam size from 1 to 10, the script first creates a copy of the config file to change the beam size setting with a `sed` command, then translates and evaluates as usual. It is a little annoying that with Joey the config needs to be modified to change the beam size at test time.

Here are two plots showing the development of the BLEU score and the time it takes to generate the translations in seconds. They were generated using the `beam_size_plot.py` script:





What the first graph shows is that given our model parameters and training data (for the language pair NL-EN), there is quite a large gap in translation quality between beam sizes 1 and 2. There are some further improvements up to the peak at beam size 5, 6 and 7 in this case. Larger beam sizes do generally not bring any improvement, on the contrary, they lead to a reduction in translation quality With only one outlier at beam size 10. This means that for a production system, we would probably use beam size 5 for decoding since it is nearly peak performing and faster than higher beam sizes.

The results somewhat contradict our expectation that larger beam sizes will always produce better translation quality, which is known as the *beam problem/beam search curse*. One reason for this is that with an increasing beam size the translations tend to become shorter. And indeed, research[2] has shown that the beam problem can largely be explained by the *length bias/brevity problem*. Therefore, one way to maintain translation quality with increasing beam sizes (at least to a certain degree) is to perform (stronger) length normalization.

The time it takes to translate increases nearly exactly linearly with beam size. It therefore makes sense for the kind of setup we had here to opt for the lowest sufficiently well performing beams size, since the increased time effort will bring diminishing or even worse returns (say 8 or 9).

[1] Credit for the orginal script goes to Emma van den Bold, thanks Emma!
[2] `https://www.aclweb.org/anthology/W18-6322.pdf`