MT Exercise 5

Topics: Byte Pair Encoding, Beam Search

Due date: Tuesday, May 28 2024, 13:59

Submission Instructions:

- **Submission file format: zip**
- Please follow our file naming convention: `olatusername_mt_exercise_xx.zip`, for instance `mmuster_mt_exercise_05.zip`
- **Submit URLs to Github forks:** Unless otherwise specified, all of your work should be committed to your Github repositories. Your submission to OLAT can simply be a text file containing links to those repositories. Submit everything in the same zip folder.
- Please submit via the exercise submodule on OLAT. Submission is open only until Tuesday, 14:00.
- You can work alone or in groups of two. If you submit as a group, please include both usernames in the submission..

Alternative Instructions:

If you would like to avoid creating a Github account, or making your work available in a public repository, here are some alternative submission instructions.

- Do not create a Github account. Instead of forking repositories to your account (ignore the instructions in the exercise below that tell you to do this), simply clone our repositories to your local machine.
- Commit your changes locally, without pushing them to a remote repository.
- **Submit ZIP folders of repository files:** For each of your Github repositories, create a ZIP folder with all files. Move to the local copy of the repo on your machine, then zip everything, including a very important hidden folder called `.git`. On a Unix machine, for example:

```
cd [your local copy of the repository]
zip -r repo.zip .
```

1 Experiments with Byte Pair Encoding (0.5 points)

In this exercise, you will experiment with different ways of constructing a vocabulary used by NMT systems. Specifically, you will investigate what the impact of byte pair encoding (BPE) is on translation quality, compared to a whole-word vocabulary. The goal is for you to experience first-hand why we are using a subword vocabulary in almost all cases.

Vocabularies for NMT systems

For an NMT system, the vocabulary (the set of all symbols[1] that can appear in sequences) must be defined ahead of time, before you start training. The vocabulary size will dictate the shape of the embedding matrices and the softmax output matrix that eventually computes a distribution over the entire target vocabulary at each decoding step.

Important: Typical NMT systems do not define a vocabulary for you. They simply assume that either

(a) you preprocessed the training, validation and test data in such a way that it only contains symbols that should be part of the vocabulary. The tools very commonly assume that you already separated whatever should represent symbols with whitespace characters. Or

(b) you prepared a vocabulary file ahead of time that the NMT training code can load, without constructing a vocabulary from the training data.

If you do not do (a) or (b), then there might also be a vocabulary threshold for the maximum number of symbols allowed. The actual number of unique words in a typical training corpus is far too large to put in an NMT vocabulary: for instance, having a softmax output matrix for more than 100k target symbols is problematic.

If a vocabulary threshold T is used, all symbols occurring in the training data are sorted by frequency, and only the T most frequent symbols are kept. All other symbols are replaced with `UNK`, or a similar string for unknown symbols. This has consequences for translation quality, as you have learned in a previous lecture.

Byte pair encoding (BPE)

An alternative to defining symbols to be entire words and using a vocabulary threshold is constructing a vocabulary of *subwords*. Some words are split up into two or more parts, and the split is marked with a special symbol, such as `@@`. Here is an example:

```
Resumption of the session -> BPE -> Re@@ sumpt@@ ion of the session
```

The purpose of `@@` is to be able to easily remove splits and restore the usual word boundaries after translation.

BPE is a procedure that needs to be trained on a specific text corpus. The training algorithm roughly does the following:

[1] We are using the word *symbol* since sequences can consist of different kinds of units, such as: words, characters, subwords – or even bytes.

1. Start with an initial vocabulary V that contains all unique single characters that occur in the corpus
2. Go through the corpus and look for all occurrences of character bigrams, sort them by frequency. Add the most frequent character bigram as an additional symbol to V.
3. Do 2. again, only that now you consider bigrams of all symbols in V, not only the initial set of all characters.
4. Repeat 3. until V reaches the desired size.

Two great advantages of this procedure are 1) that the size of the resulting vocabulary is fixed and decided by the user, and 2) that because we are always keeping in the vocabulary all single characters, any arbitrary string can be encoded with a BPE model[2]: in the absolute worst case, we will segment an unexpected string into all of its characters. **There are no `UNK` tokens in training data preprocessed with BPE.**[3]

To find out more about BPE we recommend you read a seminal paper by Rico and other researchers from Edinburgh[4] or look at the open-source code[5].

1.1 Experiments with different vocabularies

To get an intuition for the impact of decisions regarding the NMT model vocabulary, you will train several systems: a word-level one, and two BPE models.

Setting up

As always, our starting point is a repository of shell scripts:

`https://github.com/siri-web/mt-exercise-5`

Before you do anything else, fork this repository to your own Github account. Clone your forked repository in the desired place, make a new virtualenv and activate it, then install the hotfixed JoeyNMT[6] version in editable mode like in exercise 4, as well as sacremoses. You will need sacremoses for tokenizing later on.

Data

This time, we are giving you more freedom to choose the data for your experiments. There is still a script to download data automatically: `download_iwslt_2017_data.sh`. This will create a folder with data for five different languages, giving you many translation directions to choose from. The data is taken from the multilingual task of the IWSLT 2017 evaluation campaign[7]. The format is one sentence per line, and XML tags and leading whitespaces have already been

[2]One notable exception: single characters that did not occur in the training data are still unknown.
[3]Making sure that this is true is a great sanity check before starting a training run!
[4]See `https://www.aclweb.org/anthology/P16-1162.pdf`.
[5]See `https://github.com/rsennrich/subword-nmt`.
[6]See `https://github.com/siri-web/JoeyNMT-hotfixed`.
[7]See `https://sites.google.com/site/iwsltevaluation2017/TED-tasks`.

removed. Choose whichever direction you like, **except** German to English (de-en), meaning your are not allowed to use the same direction as in exercise 4.

Preprocessing, training and evaluation

This time, you are in charge of applying preprocessing steps, training models and evaluating them. Instead of simply executing commands in the shell until they work, please make sure to write reusable scripts. Additional instructions and/or recommendations:

- Sub-sample your parallel training data to 100k sentence pairs to make training manageable on consumer grade machines.
- For preprocessing, the only step we recommend you do for all models is tokenization. There is no need for truecasing, normalization or any kind of cleaning. Remember that JoeyNMT has built in functionality for this, study the sample config we provided and have a look how it was handled in the last exercise.
- **For the word-level JoeyNMT model**, we recommend to not build vocabulary files before training, but use the config fields `voc_limit` to set a vocabulary threshold for both languages. To be fair, you should perhaps use the same vocabulary size for each language. You can only share input and output embeddings (tied embeddings=True) if you use the exact same vocabulary. For word-level models, this would be wasteful, so set this to False.
- **For the BPE-level JoeyNMT model**, we recommend to build a single (= the same for both languages) vocabulary file before training. Do not use `voc_limit` in the config, instead use `voc_file` to point to the vocab file.
- For training, the scripts repository contains an example JoeyNMT config with settings that work well for low-resource tasks. Some of the fields contain `"?"` – this indicates that you should fill in the blank.
- Evaluate the models on the test set, with case-sensitive BLEU. Also, you should manually check the translations.

And some detailed advice regarding BPE preprocessing:

- **To learn a BPE model**, use the argument `--total-symbols` to ensure the vocabulary is of the exact size you specify with the argument `-s`. Without this argument, the vocabulary size is approximate since the set of single characters is not taken into account.
- Study the best practices[8] for Subword NMT and make sure to follow them for learning and applying BPE.
- JoeyNMT can apply BPE by itself, but you have to **learn** a model first to provide the codes and vocabulary files in the training config. Be aware that even though Joey applies BPE by itself during training, the provided vocabulary must be in BPE-form, so you need to apply BPE yourself beforehand to be able to provide the entire joint vocabulary in the required format. Joey does not need the vocabulary counts but just the tokens, so you may remove them.

[8] See `https://github.com/rsennrich/subword-nmt#best-practice-advice-for-byte-pair-encoding-in-nmt`.

Experiments to perform

Choose 1 language pair and translation direction, and stick with it for all experiments. For the translation direction of your choice (**except** de-en), compare (a) a model trained on entire words with a vocabulary threshold to (b) a model with a subword vocabulary. As the vocabulary size for (a) and (b), we recommend to use 2k.

(c) Train one more model with BPE data, with a different desired vocabulary size for training the BPE model. Investigate the difference in BLEU on the test set. To summarize, we expect the following minimal experiments:

	use BPE	vocabulary size	BLEU
(a)	no	2000	
(b)	yes	2000	
(c)	yes	?	

In addition to computing BLEU scores, also look at the translations manually.

Instructions for Submission

For this exercise,

- For the translation direction of your choice, perform experiments as stated above. Investigate the difference in BLEU on the test set, but also describe how translations differ if you look at them manually.
- Describe your changes, additional shell scripts and other findings such as a table with your BLEU results in the README of your fork of `mt-exercise-5`. It is also possible to submit your findings (except for scripts and other forms of code) as a PDF document.
- **Important, read very carefully: Document all of your changes and additions by committing them to your forked repositories. This means that we should be able to clone your forked repositories, and there should be instructions for us to install your custom code and train your models. We will not consider commits that are made to those repositories after the submission deadline.**

Submission: A text file with links to your forked repositories, and, depending on which type of submission you prefer for your findings, a PDF document.

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

In this exercise, you will investigate the impact of beam size on translation quality. This will help you develop an intuition for useful beam sizes for future experiments, and a nice side effect is that you get to practice your visualization skills!

Beam search and beam size

Beam search is the most widely used search algorithm in machine translation. In a nutshell, it allows to build up several hypotheses (potential translations) in parallel, without having to commit to one of them immediately. But since the space of possible sequences in the target language is quite enormous, beam search is only feasible if we keep in memory the top K most probable partial hypotheses at each decoding step.

This introduces a hyperparameter that users of NMT systems need to set: beam size K. Also, there is a trade-off regarding runtime: using a beam size of K instead of 1 means that we perform almost K times the amount of computation to translate. This makes beam size an important hyperparameter for inference.

Creating a graph of beam size versus BLEU

Using a different beam size very likely leads to a different BLEU score. Use your best model trained in the first exercise above and do the following: translate the test set 10 times, each time varying the beam size. Produce one graph to visualize the impact of beam size on the BLEU score and another one for the impact on time taken to generate the translations.

Decide on a kind of graph that in your opinion works well for this kind of data. Use any programming language of your choice to produce a graph.

Instructions for Submission

For this exercise,

- Submit both the code used to produce the graphs as well as the graphs themselves, for instance as a PNG images.
- Given your graphs, describe the impact of beam size on BLEU and time. Give your personal take on which beam size you would choose in the future.
- Commit the code, image and text to a repository such as your fork of `mt-exercise-5`. It is also possible to submit your findings (except for scripts and other forms of code) as a PDF document.

Submission: A text file with a link to your forked repository, and, depending on which type of submission you prefer for your findings, a PDF document.

In case of problems or if exercises are unclear please post in our OLAT forum.
Good luck!