MT Exercise 4

Topic: Layer Normalization for Transformer Models

Due date: Tuesday, May 16 2023, 14:00

Submission Instructions:

- **Submission file format: zip**
- Please follow our file naming convention: `olatusername_mt_exercise_xx.zip`, for instance `mmuster_mt_exercise_04.zip`
- **Submit URLs to Github forks:** 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 Understanding Code: LayerNorm in JoeyNMT (0.5 points)

In this exercise, you will examine two different ways to apply *layer normalization* for Transformer models in `JoeyNMT`: **pre-norm** vs. **post-norm**. In order to explore the difference between these two strategies, you first need to understand how layer normalization is currently implemented in `JoeyNMT`.

1.1 Background

As you have seen during the lecture, using layer normalization can improve the model's translation quality, increase training stability and speed up convergence. The original paper introducing the Transformer architecture (Vaswani et. al. 2017) [1] uses a "post-normalization" approach, meaning that the layer normalization is done **after** the input has gone through the sublayer.

Wang et. al. (2019)[2] discuss an alternative architecture where the input is normalized **before** passing it through the sublayer. Their experiments show that pre-norm is more efficient and beneficial to translation quality for deep Transformer models, i.e. models with many Encoder/-Decoder layers stacked on top of each other.

1.2 Instances of LayerNorm in `JoeyNMT`

Your task is to closely study the `JoeyNMT` code and find all instances of layer normalization. We recommend that you use Vaswani et. al. (2017) as a starting point to know where you have to look. Furthermore, have a look at the slides from lecture 9 for some example pseudo-code illustrating the difference between pre- and post-norm. In Wang et. al. (2019) you can find a more detailed account of pre-norm and its benefits in their experiment setup, which can be helpful for understanding the motivation behind pre-norm.

Your task is to write some 'missing documentation' for layer normalization in `JoeyNMT`. Where and how are pre- and post-norm implemented? What is the default behaviour? What are the specific differences between the two options and how do you control them?

Important: Make sure to clearly specify the python script and relevant code for each instance of layer normalization in `JoeyNMT`.

Submission: A PDF document with the descriptions of all instances of layer normalization.

[1] `https://arxiv.org/abs/1706.03762)`

[2] `https://arxiv.org/abs/1906.01787`

2 Implementing Pre- and Post-Normalization (0.5 points)

In the second part of this exercise, you will use your newly gained expertise of `JoeyNMT` to test the two different versions of layer normalization. If you solved the first part correctly, you should have noticed that `JoeyNMT` uses a mix of pre- and post-norm per default. Your task is to train and compare two models, each using pre- and post-norm respectively.

2.1 Preparations

As with previous exercises, we want you to set up your code and experiments in a predefined way to make them easily understandable and reproducible for us.

Low-resource setting

We would like you to be able to train systems on your own laptops.[3] For this reason, we use small NMT systems and very little data. Using very little data makes it a *low-resource setting*. More precisely, we use only 100k sentence pairs for training: this means translation quality is not great!

Many of the hyperparameters of our baseline models are chosen to work well specifically in low-resource situations.

Baseline model

To reduce the amount of computation necessary, we have trained a baseline model for you and provide the results to you, so you do not have to train them yourself. You can find the model configuration in the `config` directory. We have included the logs from training the baseline model in the same zip folder as these instructions:

- `deen_transformer_regular.yaml`: Configurations for the baseline model. Note that it consists of 4 encoder layers and 1 decoder layer.
- `baseline.log`: Logging output from training the baseline model.

Setting up

As in previous exercises, our starting point is a repository of shell scripts:

`https://github.com/moritz-steiner/mt-exercise-4`

Fork this repository to your own Github account. Then clone your forked repository in the desired place.

```
git clone https://github.com/[your username]/mt-exercise-4
cd mt-exercise-4
```

[3] We trained our baseline model *deen_transformer_regular* on CPU on a M1 Pro with 10 cores, which took a bit more than 2 hours. As a reference: During this training, around 6500 tokens per second were processed. On a entry level laptop CPU, this number might be less than a fourth. Consider this for your time management, knowing your own hardware.

Next create a new virtualenv, and activate it. Make sure you have python3.10 installed on your system, as this is the version JoeyNMT was tested with and therefore the one used for the virtualenv.

```
./scripts/make_virtualenv.sh
source venvs/torch3/bin/activate
```

Then clone Moses into the tools folder:

```
./scripts/download_moses.sh
```

The data you will be working with is provided within the data directory. The data is parallel, cleaned and tokenized, but no BPE was applied to it yet. This will be done internally with the help of the codes3200.bpe file, as well as the joint vocabulary found in the shared_models directory. So this time, you don't have to modify the texts in any way.

The reason that we give you readily prepared data this time is that we use very little data only, and train for very few updates: tiny details in preprocessing (such as sort order differences between different operating systems) can make an enormous difference.

The data we are using is a subset of 100k segments from this data set[4], and we are translating from German to English.
The preprocessing (that, to emphasize, you do not have to run yourself) involved applying truecasing to some of the corpora, learning a BPE model and extracting vocabularies.

Installing JoeyNMT

Please make sure you have activated your **virtualenv** before installing JoeyNMT.

This time we also have to fork JoeyNMT, specifically a slightly modified version that should work better for this exercise:

```
https://github.com/moritz-steiner/joeynmt-hotfixed
```

Please fork it to your own Github account, **renaming** it again to simply joeynmt and then clone your forked repository of JoeyNMT in the desired place.

Important: Do not clone your fork of JoeyNMT into your fork of mt-exercise-4 (or vice versa). The reason for this is that Github does not allow nested repositories, so if try to push a repository that contains another repository, you will get an error.

```
git clone https://github.com/[your username]/joeynmt

# change directory
cd joeynmt
```

[4] http://data.statmt.org/rsennrich/wmt16_factors/

Due to some recent updates in several Python packages, directly installing `joeynmt` will cause problems later on. You therefore need to install the specific versions manually first. Execute the following commands:

```
pip uninstall setuptools
pip install setuptools==59.5.0
pip install torch==1.13.1
```

Now, install `JoeyNMT` in **editable** mode:[5]

```
# install JoeyNMT in editable mode
pip install -e .
```

Training two models

> Please do not change any hyperparameters unless we specifically ask you to modify them. Otherwise, it is no longer possible to compare the baseline results with your own results.

While setting up this exercise, you have cloned your fork of the `mt-exercise-4` repository, so you already have all of your data for training, validation and testing in place.

Make the necessary changes you worked out in the first part of this exercise to train two different models: one with pre-norm, one with post-norm.

We have again included a script that takes care of training a model. Before you run the script:

- If you have a GPU, edit the `JoeyNMT` configuration to enable `use_cuda` and take a look at the modifications we applied to `joeynmt/scripts/training.py` on line 573 to 583 to optimize for CPU training. This will take some modification and experimentation on your part but the time savings may be worth the effort.
- If you have a different number of cores on CPU, change the script `train.sh`.
- Make sure to change the model name where appropriate, so that you don't overwrite any model or log files. Note that the `.yaml` file names the model directory using the specified name within the `.yaml` file.

Then, run the script like this for each model:

```
./scripts/train.sh
```

Visualizing your results

From training the two models you should now have a log file for each of them. Additionally, we have provided you with the training log for the baseline model. Now do the following:

- Extract the **validation perplexities** from each of these log files. The perplexity on the validation set is reported after 500 steps, so several times during one epoch.

Validation ppl	**Baseline**	**Prenorm**	**Postnorm**
500			
1000			
1500			
2000			
...			

- Create a table containing the validation perplexities of the three models (see below).
- Create a line chart on the basis of this table to visualize the results. Use a clearly distinguishable colour for each model.

Discussion

Discuss your results using the line chart. In your discussion, make sure to comment on the following points:

- Given that there is a difference in the training progress for the three models, can you think of a reason for it?
- In what way does our setup differ from Wang et. al. 2019? How could that have influenced our results?

Note: Which variant is most successful depends on the hyperparameter settings (e.g. the number of layers) and the amount of training data, among other things. We performed these experiments to show that the choice of configuration (regarding the layer normalization strategy) can make a difference, but we cannot conclude from them that one variant is categorically better.

[5] This time we are setting the flag `-e` for `pip install`. This sets the installation to *editable mode* and allows us to edit the code in the installation directory without having to reinstall/update `JoeyNMT` after each edit.

Summary and Instructions for Submission

For this task,

1. Compare two alternative model versions with `JoeyNMT`: pre-norm and post-norm.
2. Train a model with each version using the same hyperparameters as for the baseline model.
3. Visualize the validation perplexities and discuss the results.
4. Push any changes to `JoeyNMT` and `mt-exercise-4` to the corresponding forks of the repositories. You can either include your findings of this exercise in the README of your fork of `mt-exercise-4` or submit them in the PDF document from the first task.

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 (if not already described on the exercise sheet). 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.

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