Exercise 8 - **Assignment 4**
Mathematical Problem Solving with Transformers

Deep Learning Lab
Due: Sunday 15 January 2022, 10:00 pm (time in Lugano)

November 18, 2022

Submission Instruction

You should deliver the following by the deadline:

- **Report**: a single *pdf* file that clearly and concisely provides evidence that you have accomplished each of the tasks listed below. The report should not contain source code (not even snippets). Instead, if absolutely necessary, briefly mention which functions were used to accomplish a task. All figures should have a caption, and there should be a text that refers to it (see the Latex documentation if you are not familiar with this). Please also make sure that the section numbering in your report exactly follows that of the assignment.

- **Source code**: a single Python file (*.py*) that can be used to accomplish each of the tasks listed above. The source code will be read superficially and checked for plagiarism. Importantly, if a task is accomplished in the code, but not documented in the report, it will be considered missing. Therefore, please make sure to report everything in the report. Note: Jupyter/Colab notebook files (*.ipynb*) are not accepted.

Please carefully read the instructions above to prepare your submission. Failure to stick to these rules may result in reduction of points. As stressed in class, it is strictly forbidden to exchange your code with others or copy texts/code from the internet. The score of 0 will be given to all students involved in such cases.

NB: You should use a GPU for this assignment.

In Sec. 4 of the lecture slides, encoder-decoder models have been presented as models for solving sequence-to-sequence problems. In particular, the PyTorch implementation of Transformer models has been commented. In this assignment, you will train such a model to solve mathematical problems. The following sections will guide you to implement each component of the system step by step. While you should try to solve the following tasks (more or less) in the given order, **we recommend you to read the entire text before you start solving the problem**, so that you can get an overview of the problem you will be solving as well as the deliverable you are asked to provide in your report. Remember, **a helper code notebook is available on iCorsi**.

1 Problem Setups and Preliminaries

We are interested in solving basic mathematical problems using neural networks. The problems we consider are subsets of the DeepMind mathematical dataset[1]. Each example consists of a question and the answer. The dataset contains different *modules*, corresponding to different problem categories. Here are a few examples:

[1]https://github.com/deepmind/mathematics_dataset

- Module: *numbers - place value*
  Question: "What is the hundred millions digit of 217883211?"
  Answer: "2"
- Module: *comparison - sort*
  Question: "Put -4, 2/5, 0, 1, 132 in decreasing order."
  Answer: "132, 1, 2/5, 0, -4"

In this assignment, we treat this problem as a character-level sequence-to-sequence mapping problem: the encoder reads the question as the input sequence, and the decoder outputs the answer sequence. You will train separate models for a few selected modules of the mathematics dataset.

Download the pre-processed dataset from iCorsi3. Note that each of the modules consists of the following files: `train.x` containing the question part of the training data, and `train.y` corresponding to the answer. Analogously, you should find the **validation** set with the prefix `interpolate`. We do not consider any test set in this assignment. Unless otherwise indicated, **use the *numbers - place value* module** to answer to all following questions.

1. (8 pt) Report the number of sentences and characters in the training and validation sets. Also report average question and answer lengths in terms of number of characters for each set (No need to provide the code you used for this question).

2 Dataloader

The implementation of dataset and dataloader you need for this task was presented in the lecture and it can be downloaded from iCorsi. No coding is thus needed here. We also note here that loading the data might take some time (e.g., more than 5 min). Answer the following questions about the corresponding code.

1. (1 pt) The provided implementation automatically produces the vocabulary to be used by the model while reading the dataset. Does the implementation use the same vocabulary for the input (source) and output (target) of the model by default? (max. 1 sentence)
2. (1 pt) Recognize that the vocabulary contains the so-called unknown token, `unk`. By reading the implementation of `Vocabulary`, find out and explain its role (max. 1 sentence).

3 Model

The helper code already contains a working implementation of a Transformer encoder-decoder model. You should be able to recognize various components of the Transformer model we studied in class. Note also that in addition to the usual `__init__` and `forward` methods, it implements separate class methods to generate different masks (remember the hints presented in the lecture).

1. (5 pt) Check how `self.transformer` of the class `TransformerModel` is used in its `forward` function: `out = self.transformer(...)`. Notice also how `self.encoder` and `self.decoder` are defined in the `__init__` function. In fact, instead of forwarding `self.transformer` as is done in the current code, one could forward `self.encoder` and `self.decoder` separately. **Implement** a function `forward_separate` that implements this second option that makes use of `self.encoder` and `self.decoder` instead of `self.transformer` in the forward computation of the model. *Hint: you can start by copying the provided `forward` function, and replace the line corresponding to forwarding of `self.transformer` by lines containing forwarding of `self.encoder` and `self.decoder`. The purpose of implementing this function is to help you understand how one can separately make use of encoder and decoder components on the Transformer. For the rest of the problem, you can simply use the originally provided `forward` function.*

4 Greedy search

Your task here is to implement the greedy search algorithm for your Transformer model by following the hints and by answering the questions below. Your implementation should allow to process multiple sequences in a batch in parallel.

1. (15 pt) *Hint for implementing the main part of the algorithm*: as is pointed out in the previous section, you can forward separately your encoder and decoder which are sub-modules of `nn.Transformer`. In a search algorithm, the encoder needs to be forwarded only once for each batch, while the decoder needs multiple forwarding: at each decoding step, the model prediction must be extracted, concatenated to the current decoder input sequence, and fed as the new input to the model in the next decoding step. Thus, something in the form: `dec_input = torch.cat([dec_input, predicted])` along the correct axis, is needed. Also, you should actively make use of already implemented helper functions for creating masks: `create_src_padding_mask` and `create_tgt_padding_mask`.

2. (*Bonus*, 5 pt): Criticize concisely the implementation of `nn.Transformer` when considering its usage in a search algorithm.

3. (5 pt) *Stopping criteria*: You need to specify the stopping criteria for your search algorithm. You should use the following two criteria. The first stopping criterion is when the model outputs the end-of-sentence token, `eos`. Second, as you will see in the next section, an output of your model will be counted as a correct answer, only if all tokens match the true answer sequence. Thus, to make it simple, you can also stop decoding when the length of the output sequence exceeds that of the target sequence (note: you should otherwise never use any information from the evaluation target sequence!).

4. (5 pt) *Batch mode evaluation*: For fast evaluation, you have to carry out search in batch mode. You can terminate the search, once the stopping criteria are met for all sequences in the batch.

5 Accuracy computation

1. (5 pt) Implement a function which computes the accuracy (the number of correct answers divided by the total number of questions); a model prediction is counted as correct, only when the entire output sequence (all characters) matches that of the correct answer. You might want to modify the output format of the greedy algorithm from the previous section, depending on what's the most convenient for your evaluation function.

6 Training

Finally, you implement the training pipeline.

1. (2 pt) Which loss function do you use? Why? (1 sentence)

2. (15 pt) Implement the training code. Remember that the *decoder* component of the model is similar to language models; you should carefully define the input tensor to be fed to the decoder and the target label tensors to be given to the loss. During training, you have to keep track of both training and validation losses, as well as accuracy on the validation and training data subsets, every n training steps (with a reasonable choice of n). It might also be helpful for you to print some model prediction example every n steps.

3. (5 pt) *Gradient accumulation.* It is often helpful to use a large batch size to train Transformers. In order to simulate a large batch size, while working with limited GPU memory, you can accumulate gradients for a couple of steps i.e. update parameters (by calling `optimizer.step()`) only every k steps (e.g. assuming the batch size is 64, if you only update every 10 batches, your effective batch size is 640). Be careful where you call `optimizer.zero_grad()`.

7 Experiments

1. (5 pt) Using the training pipeline you implemented in the previous section, train a model for the module **numbers - place value** using the following hyper-parameters:

   - Effective batch size of about 640 (for example by setting the batch size to be 64 and by accummulating gradients for 10 steps).
   - Learning rate of 1e-4 with the Adam optimizer.
   - Gradient clipping rate of 0.1 (i.e. `torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)` has to be called before updating the parameters.)
   - Model with 3 encoder layers, 2 decoder layers, hidden dimension of 256, feed-forward dimension of 1024, using 8 attention heads.

   You should be able to see the training loss going down quickly. As a rough reference, the validation accuracy should reach over 90% after 5 min when training on a GPU (and 100% after 10 min).

2. (15 pt) Report/Plot your training curves (train/validation loss/accuracy) at least until you reach 90% validation accuracy. Print, check, and report 3 example questions from the validation set and the predictions of your trained model.

3. (8 pt) *Hyper-parameter tuning*: Can you reduce the model size e.g. number of layers, hidden layer size,..., without obtaining degradation in terms of accuracy? Try at least two more hyper-parameter configurations and report the corresponding hyper-parameters and training curves.

4. (5 pt) Run experiments on the module **compare - sort**. Try to achieve above 90% accuracy on the validation set. Report the results similarly to question 7.2 above. As a rough reference, using the hyper-parameters from 7.1 above, this should be possible after about 3 hours of training. By training for longer, you should be able achieve over 95% accuracy. You can also report intermediate performance, if the performance is reasonably good. Is the task harder? How does the task fundamentally differ from **numbers - place value**?

5. (*Bonus*, 5 pt): Run experiments on **algebra - linear 1d** module, report results and comment. We note that this module is harder: it will require more time for models to achieve good performance, you can also report intermediate performance, if the performance is reasonably good.