Assignment 4

Deadline: 17 January 2024 - 10.00am

By the deadline, you are required to submit the following:

Report. Prepare a single PDF file that presents clear and concise evidence of your completion of each of the listed tasks. **Use the LaTeXtemplate available on iCorsi**.

Source code. A Python script **using the template available on iCorsi**, performing each of the specified tasks. If a task is accomplished in the code but not documented in the report, it will be considered incomplete. Therefore, make sure to thoroughly report all your work in the submitted report. **Jupyter notebook files will not be accepted**.

General hints. The question marked with * is more challenging, and we recommend possibly leaving it as the last question to solve. To obtain the maximum grade, not only should all exercises be completed correctly, but the plots must also be clear, have a legend, and have labels on the axes and the text should be written in clear English. For saving plots in high quality, consider using the command `matplotlib.pyplot.savefig`. **Clarity of plots/text and code will account for a total of 5 points**.

Submission rules: As already discussed in class, we will stick to the following rules.

- Code either not written in Python or not using PyTorch receives a grade of 0.
- Later submission will get the following penalty:
  - 17th January, after 10 am: -10%
  - 18th January: -15%
  - 19th January: -20%
  - 20th January: -100% (Assignment not accepted)
- If plagiarism is suspected, TAs and I will thoroughly investigate the situation, and we will summon the student for a face-to-face clarification regarding certain answers they provided. In case of plagiarism, a score reduction will be applied to all the people involved, depending on their level of involvement.

- If extensive usage of AI tools is detected, we will summon the student for a face-to-face clarification regarding certain answers they provided. If the answers are not adequately supported with in-person answers, we will proceed to apply a penalty to the evaluation, ranging from 10% to 100%.
- GPU usage is recommended for this assignment

Conversational model with Transformers

In this assignment, we will train a conversational model.

We will leverage the Cornell Movie Dialogues Corpus [1] — a rich dataset containing movie character dialogues.

You are asked to define and train a conversational model using transformers in PyTorch

Remark 1: The assignment is pretty tough, but we retain that it is impossible to learn PyTorch without hands-on practice. For this reason, the total amount of points is equal to 155 (120 of practical questions + 5 points for quality of writing/code + 30 bonus points).

Remark 2: This dataset has been widely used. Before blindly starting a task, search on Google what's going on (tutorial, blog post, even papers, . . .).

Remark 3: Training language models is a hard task, that requires serious computational resources and clever architectures. It's ok if the answer you get seems not to make that much sense, as this is a small project. This is also what happens in many tutorials you can find online.

1 Data (40 pts)

As mentioned, we will use Cornell Movie – Dialogues Corpus, a large collection of conversations extracted from raw movie scripts.

Note: We recommend not using GPUs / Not wasting GPU time for this task. After the first training loop, you may want to go back to this Section and revise some choices.

1. (5 pts) Download the folder `data.zip` from iCorsi. These files are part of the Cornell Movie Dialogues Corpus. Inspect a couple of lines of the files and comment on the content in a few sentences

2. (5 pts) Create a list of pairs, each of them containing pairs of sentences (pairs of string at this level). Note that some conversations are not simple question-and-answer conversations, but, at the end of the day, we want pairs. If one conversation contains more than two sentences, for instance, 4 sentences, namely (s_1, s_2, s_3, s_4), you can either decide to add just the first two, namely (s_1, s_2) or all the pairs $(s_1, s_2), (s_2, s_3), (s_3, s_4)$. Comment on your choice.

3. (10 pts) Tokenize the sentences at a word level. At this time, you may want to do more involved tokenization w.r.t Assignment 3, for instance, create one token for each punctuation symbol, such as ?., !, and remove other types of punctuation. If this is the case, you can use the function

`clear_punctuation` already available in the template. Don't forget to append at the end of each sentence the token `<EOS>` and at the beginning of each answer - *just for the answers!* - the token `<SOS>`. Comment on your choices.

4. (3 pts) Remove all the pairs where there exists a sentence that is longer than a certain length `max_length`. In this case, it could help plot the length distribution to decide a suitable threshold. Comment on your choices.

5. (0 pts) This process should be fast, however, we recommend saving the sentences in pickle format.

6. (3 pts) Count the words in your corpus and eliminate all the words below a certain threshold - it could help to plot the distribution of the frequency to decide a suitable threshold. Then, eliminate all the pairs having at least a sentence containing one unknown word, namely, a word you have eliminated. This process may take a lot of time (15 minutes on my computer). In the meantime, read the assignment until the end. **Note:** This is not what you want to do in practice. In practice, you want to replace each unknown word with a dedicated token `<UNK>`. However, for the purpose of this assignment, simply eliminating the sentence is easier.

7. (1 pts) **Strongly recommended**: Save the list thus obtained in pickle.

8. (3 pts) You should come up with many sentences, e.g., 60'000. After fixing a seed, we recommend randomly sampling a subset of them, e.g., 10'000. This could help speed up the training. Comment on your choices.

9. (5 pts) Modify the class `Vocabulary` as requested in the code to obtain a Vocabulary. Comment on what you did.

10. (5 pts) Modify the class `Dataset` as requested in the code to obtain a Dataset and hence, define a Dataloader object. Don't forget to add a collate function that automatically pads the sequences in each batch. Comment on what you did.

2 Model & Tools for training (35 pts)

Part of the code is ready-to-use and you are just required to complete some parts.

1. (2 pts) In a maximum of 5 lines, comment on the implementation of the layer `PositionalEncoding`.

2. (10 pts) Now, focus on the class `TransformerModel` and complete the `__init__` method. Comment here on what you did.

3. (5 pts) Complete function `create_padding_mask` that masks the token `<PAD>`. It takes as input a tensor and returns a boolean tensor having `True` in the `<PAD>` token and false otherwise.

4. (13 pts) Complete and comment on the function `forward`, especially focusing on the usage of all the masks.

5. (5 pts) Explain why the masks we have defined contain `True / False` and not `0, -inf` as discussed in class.

3 Training (35 pts)

Note: in the training pipeline you have three "players":

- The input sentence, that must end with `<EOS>`.
- The output sentence that is the input of the decoder, which should start with `<SOS>` and not have `<EOS>`.
- The output sentence that is the output of the decoder - the one we want to predict, which should end with `<EOS>` and not having `<SOS>`.

To make it even more clear, you can have a look at Figure 6 of this tutorial.

1. (15 pts) Implement a training pipeline with a standard training loop. Considering (a) using a scheduler and (b) starting with the hyperparameters available in the "Remark" section. Consider your training good when your validation loss is below 1.5. You can put this training pipeline in a function named `train`. Comment here on what you did for training. Report a plot of the training and validation loss, explicitly showing that the goal has been achieved. **Note**: the 1.5 threshold is not mandatory. You can get it in a few epochs if you remove the `ignore_index` requirement for the `<PAD>` token. (If you do it, explain why and its implications). However, it is still mandatory to show a training and validation loss function. But you have to report (a) a decreasing loss function; (b) No overfitting; and (c) At least 8 epochs. This choice can lead to meaningless sentences: in this case, it's ok to have them

2. (20 pts) Define another training pipeline in a function named, for instance, `train_ga`, where you replace standard loss computation with gradient accumulation. You can check Lecture 4 for some hints on how to do it. Comment here on what you did, with a special focus on any improvement you get with respect to the standard training loop. Consider your training good when your validation loss is below 1.2. **Note**: the 1.2 threshold is not mandatory. You can get it if you remove the `ignore_index` requirement for the `<PAD>` token. (If you do it, explain why and its implications). However, it is still mandatory to show a training and validation loss function. But you have to report (a) a decreasing loss function; (b) No overfitting; and (c) At least 8 epochs. This choice can lead to meaningless sentences: in this case, it's ok to have them. Report a plot of the training and validation loss, explicitly showing that the goal has been achieved. **Remark:** gradient accumulation is particularly effective in training transformers (large models) because it allows you to have smaller batches per time and accumulate the gradient many times. To make it clear, you can consider a batch size equal to 2, accumulate the gradient 32 times, and make it "equivalent" to consider a batch size of 64.

This allows you to either use a larger vocabulary or a bigger model, as you consume less memory on your GPUs for data. For all these reasons, take particular care on which is the suitable value of the loss function to report.

4 Evaluation (10 pts)

Generate some answers to your questions by implementing different search strategies.

- (3.5 pts) Implement greedy strategy: at each step, pick the word that is the most likely to be the next one.
- (3.5 pts) Implement top-k sampling strategy: after your model generates a probability distribution over your vocabulary, pick just the top k values and sample the next word among them. This allows you to prevent sampling very rare words/outliers - they are possible, even though with a low probability.
- (3 pts) Test both greedy and top-k sampling strategies on three different input sentences. Report here the 3 different input sentences plus 6 answers and comment on what's going on.

Note: harmful content may be generated from your architecture. Reporting them here will lead to a score of 0 on the full assignment.

5 Bonus questions* (30 pts)

1. (5 pts - Easy) Define a function `train_ga_hf` that replaces your implementation of Gradient Accumulation with the HuggingFace function `Accelerate`. You can read more here.
2. (10 pts - Medium) A commonly done practice when implementing Transformers is to implement separately the encoder and decoder parts, using for instance the functions `nn.TransformerEncoder` and `nn.TransformerDecoder`. Create a new class `TransformerSeparate` that implements the model in this way. Take particular care of how the forward function works in this case.
3. (15 pts - Medium/ Hard) You were required to implement two strategies for generating sequences from your trained model. Implement instead Beam Search. This is a tough task, as a non-efficient implementation of beam search is time-consuming in terms of memory.

Remarks

Some technical suggestions

- In Kaggle, when you set the model in the evaluation mode, it raises some errors. While we try to understand why, you can consider reporting the evaluation loss without putting the model in evaluation mode. If you do so for technical issues, please state it explicitly, also declaring why this choice is not ideal and should not be used in practical cases.

- Point 1.6 can take a lot of time if your Python version is smaller than Python 3.10.4
- In the forward method of the Transformer model, consider adopting the following lines according to your PyTorch version:

```
# For torch 2.1.1
tgt_mask = self.transformer.generate_square_subsequent_mask(tgt.size(1),
dtype=torch.bool).to(tgt.device) # (T, T)

# For torch 2.0.0
tgt_mask = self.transformer.generate_square_subsequent_mask(tgt.size(1)).to(
tgt.device) # (T, T) tgt_mask tgt_mask.bool()
```

Some suggestions on the hyperparameters choice

- For the standard training, I have used a batch size of 32, an initial learning rate of 0.0001 (with a scheduler). I have randomly sampled 20'0000 pairs. The parameters of the models are $d_{model} = 512$, 6 encoder and decoder layers, 2048 for the feedforward neural network, 8 heads, and a dropout probability of 0.2.
- For the training with accumulated gradient, I have used a batch size of 2 and accumulated the gradient 32 times, an initial learning rate of 0.0001 (with a scheduler). I have randomly sampled 30'0000 pairs. The parameters of the models are $d_{model} = 512$, 6 encoder and decoder layers, 2048 for the feedforward neural network, 8 heads, and a dropout probability of 0.3.

References

[1] Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.