Assignment 3

Deadline: 10 Dec 2023 - 11.59pm

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 overleaf template 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 them as the last questions 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.
- Submission between 00.00 am - 00.10 am of Monday 11th will be accepted with no reduction if it is the first time you submit late, 10% reduction if it is the second time, and 15% reduction if it is the third time.
- 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

1 Language models with LSTM [90/100]

The focus of this assignment is on text generation with LSTM. In particular, we will generate news titles starting from real news titles. The dataset for the assignment is the one provided by [2] and can be (freely, which is nice) downloaded from here. The data contains 42 news categories in the dataset, however, we will stick to the most represented category, which is politics. Your goal is to design a model that obtains low perplexity and generates some credible news titles. Note that the structure of the model and the choice of the hyperparameters are very relevant in this assignment. For this assignment, you can re-use part of the code of Exercise 4, but be sure the mention when this is the case.

1.1 Data (20 points)

For this part, we recommend not to waste GPU time and to do everything on your computer.

1. (2 pt) **Download data** - Download the data and read it as pandas Dataframe. Inspect the data and the columns. Filter only the news labeled with `POLITICS`. At this point, you should have 35602 sequences. Print and report here the first 3.

2. (3 pts) **Tokenization** - For each title, tokenize it at a word level and create a list containing lists of words, one list for each title. Make sure to convert all the words in lowercase. Append, at the end of each sentence, the `<EOS>` token. As you may have noticed this procedure takes a lot of time. Save the list you created in pickles the first time you properly create it and load them for the next tests. Use `pickle` format. Read this if you don't know how to use it. Print and report here the first three sentences tokenized.

3. (4 pts) **Dictionaries** - Create a list named `all_words` containing all the words exactly once and save it. As already done in Exercise 4, put at position 0 the token `<EOS>` and at position -1 the token `PAD`. Create two dictionaries named `word_to_int` and `int_to_word` representing a mapping from words (keys) to integer. At this point, you should have 33207 tokens. Save the dictionaries in `pickle`. Do this procedure just once. Report here the most common 5 words.

4. (5 pts) **Dataset class** - Create a dataset class taking as input your list of tokenized sequences and the dictionary `word_to_int`. Each item should be a tuple having as the first item the indexes of all the words of the sentence except the last one; the second one contains all the elements of that sentence except the first one. Explain here what you need to implement.

5. (6 pts) **Padding, Batches, Dataloader** - Define a function `collate_fn(batch)` that pads sequences with `PAD` until the maximum sequence size of the batch. Then, create a Dataloader that autonomously to the padding using `collate_fn`. Although an analogous task has been performed in Exercise 4, you can read more on this here. Explain here what you need to implement.

1.2 Model definition (20 points)

Define your model. It must be an LSTM-based model where you can add whatever you like. Two things are mandatory:

- Having an Embedding at the beginning,
- Using LSTM, potentially stacked.

Apart from that, you can add Dropout, fully connected layers, and whatever comes to your mind. To get full marks, motivate your choice. **Note: transfer learning is not allowed**

Don't forget to define a method named `init_state` to define the initial state. What is the main difference with RNNs?

1.3 Evaluation - part 1 (10 points)

As already discussed in class, we have two strategies to prompt sentences from our model: randomly sample from the distribution (*sampling strategy*) and sample the token with the highest probability (*greedy strategy*). Implement such strategies as described above, explaining for each point what you did (at an high level)

- (3pts) Implement a function `random_sample_next` which randomly sample the next word on $p(w_n|w_0, w_1, \ldots, w_{n-1})$. You can get some hints from Exercise 4.
- (4 pts) Implement a function `sample_argmax` which picks the word having the highest probability according to the distribution $p(w_n|w_0, w_1, \ldots, w_{n-1})$.
- (3 pts) Implement a function `sample` that takes as minimal input a prompt (some words), the model, and one of the two functions `random_sample_next` or `sample_argmax_next`. Such a function starts from a prompt and completes the sentence with word generation until the token `<EOS>` is met.

1.4 Training (30 points)

1. (15 pts) Train your model with a standard training loop. For every epoch/ every k epoch (you choose k), prompt a sentence your model generates and the perplexity value. Consider your training good when you get a loss function below 1.5. You may need to do gradient clipping. Report: (a) A plot of the loss function, explicitly showing that it goes below 1.5 (b) A plot with the perplexity values and (c) Three sentences your model generates: one after the first epoch, one in the middle of the training, one at the end of the training. Comments on what's going on.

2. (20 pts) Truncated backpropagation trough time (TBBTT) is a method in recurrent neural network training that limits backpropagation to a fixed number of time steps. It addresses computational challenges in handling long sequences by breaking them into manageable segments, named chunks. In the provided Python template, you can find a starting point for implementing training with TBBTT. Complete it by explaining here what you did/added/modify. Then, train your model using TBBTT and comment on the differences you observe with respect to standard training. Report: (a) A plot of the loss function, explicitly showing that it goes below 1.5 (b) A plot with the perplexity values and (c) Three sentences your model generates: one after the first epoch, one in the middle of the training, one at the end of the training. Comments on what's going on, also comparing to standard BBTT. Consider your training good when your loss is below 1.

1.5 Evaluation - part 2 (5 points)

Generate and report 2/3 sentences in the following way:

- Start with any prompt, e.g, "the president wants", and generate three sentences with sampling strategy.
- Start with the same prompt as above and generate three sentences with the greedy strategy.

What do you notice?

Do you think your titles are credible? Try submitting the best titles you get to some friends or colleagues along with other titles taken from the dataset. **Note: harmful content may be generated from your architecture. Reporting them here will lead to a score of 0 on the full assignment.**

1.6 Bonus question* (5 points)

In the well-known embedding word2vec [1], it is assessed that the mathematical operation "vector("King") - vector("Man") + vector("Woman") results in a vector that is closest to the vector representation of the word Queen". Could you claim the same for your Embedding? **Note**: the amount of points you get is not related to the answer yes/no to the previous question, as the goal of this assignment is not to create a good embedding. We will evaluate how do you asses this and your motivations.

Remarks

A non-exhaustive list of my hyperparameters in the base case (Training time: 10mins): Batch size = 64, Epochs = 12, Learning rate = 0.001, Optimizer = Adam, Gradient clipping used, with clip = 1, Hidden size of the LSTM = 1024, 1 single LSTM, Embedding dimension = 150.

With TBBTT I diminish the epochs to 5 and increase the LSTM size to 2048. (Training time: 10 mins)

2 Questions [5 points]

As a strategy for tokenizing our dataset, we have used plain tokenization, namely, one word = one token. This could lead to some issues, for instance, if you consider "does", "doesn't" and "not", they are all present in our vocabulary as three different items. As you may imagine, this is not the best option. Other tokenization techniques exist, *Character-level Byte-Pair Encoding* and *WordPiece*. Study these two tokenization techniques and explain in max 10 lines how they work.

3 Do you like NLP?

We initially planned the assignment with a second part - eventually dropped to not make the assignment too heavy. The idea was to use also the second part of the dataset, namely, the column `short_description` to generate a title plus a short description. If you want, you can try also this task - here, TBBTT is more than recommended.

Another interesting part of NLP is the bias in the training set, which is often the cause of the generation of offensive content. During the training, you notice that this may happen. Preventing the generation of offensive content is an active research area, and you can learn about it and try to add this part to your model - not tested by us.

We will not evaluate both of these tasks, but we are happy to check and discuss your ideas.

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[2] Rishabh Misra. News category dataset. *arXiv preprint arXiv:2209.11429*, 2022.