Exercise 7 - **Assignment 3**
Language Modeling with RNNs

Deep Learning Lab
Due: Sunday 4 December 2022, 10:00 pm (time in Lugano)

November 9, 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.

1 Preliminaries and Reading Comprehension [28/100 points]

Similarly to Exercise 6, in this assignment, you will work with character-level language models based on recurrent neural networks (RNNs). As has been presented in the lecture (Sec. 3.2), an RNN language model is trained to predict texts from left to right one token at a time. While the primary role of a language model is to estimate the probability of a sentence, such a model can be also used to *generate* texts recurrently by generating one character at a time according to its output distribution and feeding it back as the input to generate the next character.

Importantly, for this assignment, we provide an initial code that implements all basic components you need to build a RNN language model and generate texts from it (available on iCorsi; and commented in class). In this Sec. 1, your main task is to read and understand the provided code, while in Sec. 2, you'll run experiments using the provided code. Finally in Sec. 3 you'll introduce some modifications to the code.

1.1 Text data

Many public domain books are available from Project Gutenberg. Download "Aesop's Fables (A Version for Young Readers)" from here. For simplicity, we'll assume that no pre-processing is needed.

1. (4 pt) Report the following properties of the text data: number of characters, number of unique characters, and number of lines in the file. No need to provide code (e.g., you can use simple bash script to compute these statistics). Also report one property you can recognize when you look at the text (it can be anything: usage of capitalized letters, structure of the text, etc...)

2. (**Bonus**, 2 pt) We repeat: no text preprocessing is needed for this task. But if you were asked to do some text preprocessing, what would you do? (1 or 2 sentences)

1.2 Dataloader/Batch Construction (Repeated from Exercise 6)

Consider a very long string representing the whole book (the text file you downloaded). Backpropagating gradients through an RNN which is unrolled as many times as there are characters in that long string (backpropagation through time; BPTT) will require too much memory. Instead, the string must be broken down into smaller text chunks. Chunking should be done such that the first token for the current chunk is the last token from the previous chunk. The first token of the chunk is the first input token to be fed to the language model, while the last token of the chunk is the last target token to be predicted by the model within a given chunk. We'll then train the model by *truncated backpropagation through time* (which was also covered in the lecture, Sec. 3.2), i.e.:

- We limit the span of the backpropagation to be within one chunk.
- We initialize the hidden state of the RNN at the beginning of the chunk by the last state from the previous chunk. We thus can not randomly shuffle chunks, since the RNN states need to be carried between consecutive text chunks.

We train the model on a batch of text chunks such that multiple chunks are processed in parallel.

An implementation to read a raw text file and to create a batch generator which is compatible with the description is included in the helper code. No coding is thus needed here. Answer the following questions which test your reading comprehension of the provided code.

1. (2 pt) In the method `get_idx` of class `Vocabulary`, explain why there is a `if` branch (max. 2 sentences).

2. (4 pt) In `Vocabulary`, you should be able to recognize two dictionaries `id_to_string` and `string_to_id`. What are the keys and values for each of these dictionaries?

3. (1 pt) Regarding `LongTextData`, what statistic do you obtain when you call `__len__`? (max. 1 sentence/a few words)

4. (1 pt) Same question for `ChunkedTextData`: what statistic do you obtain when you call `__len__`? (max. 1 sentence/a few words)

5. (**Bonus**, 10 pt) In the method `create_batch` of `DataBatches`, find the following two lines:
   `padded = input_data.data.new_full((segment_len * bsz,), pad_id)`
   `padded[:text_len] = input_data.data`
   Explain what these two lines do. *Hints: You should try to answer the following questions. What is `input_data.data`? What does `new_full` do? Can you describe what kind of object the result of the first line `padded` is, and how it looks like? How does `padded` look like after the second line?* (max. 6 sentences)

6. (3 pt) Another question on `create_batch` of `ChunkedTextData`: in the loop at the end, under the branch `if i == 0:`, what is the shape of `padded[i * bptt_len:(i + 1) * bptt_len]` in terms of input arguments `bsz` and `bptt_len`? (max. 1 sentence)

7. (3 pt) Same question for `padded[i * bptt_len - 1:(i + 1) * bptt_len]` in the `else` branch: what is its shape in terms of input arguments `bsz` and `bptt_len`? (max. 1 sentence)

1.3 Modeling, Training, and Decoding

The helper/initial code implements a standard RNN model, the corresponding training loop, and the greedy decoding algorithm that can be used for text completion. Your task is to answer the following questions to demonstrate your understanding of the provided code.

1. (2 pt) Notice that we apply `.detach()` to the hidden states of RNNs. Explain why we do this (1 or 2 sentences).
2. (2 pt) Explain why `ignore_index=0` is given as an argument to `nn.CrossEntropyLoss` (1 or 2 sentences).
3. (2 pt) What is the input shape expected by `self.rnn` in `RNNModel`. Express it in terms of sequence length N, batch size B, and character embedding size D.
4. (2 pt) What is the output(s) of `self.rnn`. If it contains any tensors, express their shape in terms of sequence length N, batch size B and RNN hidden dimension H, and number of RNN layers L.
5. (2 pt) The function `complete` implements the greedy decoding algorithm. Notice that inside the training loop, we call it with the argument "Dogs like best to." Explain why we do this inside the training loop? (1 or 2 sentences).

2 Running Experiments Using the Initial Code [30/100 points]

In this section, your task is to run the provided code and report your observations.

1. (2 pt) While the main goal of this section is to run experiments using the initial code, let's introduce one tiny modification. **Modify** the code such that you monitor your model's training *perplexity* instead of the loss. The perplexity of a language model p for a text w_1^N (with a start token w_0) is defined as:

   $$\text{Perplexity} = (\prod_{n=1}^{N} p(w_n|w_0^{n-1}))^{-\frac{1}{N}} = \exp(-\frac{1}{N}\sum_{n=1}^{N} \log p(w_n|w_0^{n-1})).$$

   From this equation, you should recognize that the perplexity can be computed by simply **applying the exponential to the cross-entropy loss in PyTorch** with the default parameters.

2. (20 pt) Train an RNN language model using the following hyper-parameters:

   - an input embedding layer with a dimension of 64
   - one LSTM layer with a dimension of 2048
   - a BPTT span of 64 characters
   - the Adam optimizer with a learning rate of $5e^{-4}$.
   - with a gradient clipping with a clipping threshold of 1.0. This can be done by adding the following line between the gradient computation (typically `loss.backward()`) and the parameter update (typically `optimizer.step()`):
     `torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)` where `model` is your PyTorch model. Note that this is also **already implemented** in the provided code.

   We do not make use of any validation data: you can consider the model to be well trained when its training perplexity is below 1.8. You should be able to achieve this performance in less than 10 minutes on a GPU.

   Report the evolution of both perplexity and text generation quality. For perplexity, you must provide a plot. For generation quality, provide one example of text generated by the model at three different stages (roughly beginning/middle/end) of training. Introduce any necessary modifications to the coded to do this.

3. (8 pt) Once your model is trained, run greedy decoding using the following prompts using a decoding steps of (more than) 500:
   (a) A title of a fable which exists in the book.
   (b) A title which you invent, which is not in the book, but similar in the style.
   (c) Some texts in a similar style.
   (d) Anything you think might be interesting.

For each case, **report** the input prompt of your choice and the output you obtain from the model. Provide a concise global comment (max 4 sentences discussing e.g. Is the output meaningful? Is the model capable to produce novel texts?...)

3 Extending the Initial Code [40/100 points]

Now you will implement a few modifications to the helper code.

1. (20 pt) The language model in the initial code is based on standard RNN (`nn.RNN`). **Implement** an **LSTM** language model using PyTorch `nn.LSTM`. **Introduce** any modifications needed to the training and decoding pipelines so that you can train and evaluate LSTM language models.

2. (5 pt) Using exactly the same hyper-parameters as above, except the learning rate that you should set to 0.001. Report the training evolution and the best perplexity you obtain. You should achieve a training perplexity value below 1.03.

3. (10 pt) The helper code implements the greedy algorithm. **Modify** the code to add an option to allow sampling during decoding; i.e. instead of taking the argmax according to the model's output distribution, you sample randomly from that distribution. Some hints were presented in the lecture.

4. (5 pt) Using a well trained LSTM language model, compare greedy decoding and sampling using the same prompt. Show the results for two prompts that are:
   (a) A title of a fable which exists in the book.
   (b) A title which you invent, which is not in the book, but similar in the style.

   Is sampling a good idea?

5. **(Bonus, 5 pt):** Be creative! Use other types of texts (e.g. the Python code from your last assignment), to train an LSTM language model. Report at least four prompts and the corresponding output text generated by the model using the greedy decoding. Comment.

4 Questions [2/100 points]

Provide a **brief** answer to the following questions **in your own words** and add them to your report. You can find more information in the relevant subsections of e.g. chapter 10 of the Deep Learning Book.

1. (1 pt) What is the perplexity of a language model that always predicts each character with equal probability of $1/V$ where V is the vocabulary size? (max. one word)

2. (1 pt) When training an RNN using backpropagation through time, one may encounter the problem of vanishing gradient, i.e., the values of the gradients propagated through a long sequence (of transformations) become very small. Why is that an issue? (max. 1 sentence: It is an issue because ...)