Exercise 7 - **Assignment 3**

Deep Learning Lab
Due: Thursday 9 December 2021, 10:00 pm (time in Lugano)

November 16, 2021

1 Language Modeling with RNNs [94/100 points]

In this assignment, you will implement a character-level language model using recurrent neural networks (RNNs). As it 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 evaluate the "correctness" of any sentence by providing its probability, 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 an input to generate the next character. If you are interested in learning more about such an idea, an accessible introduction is available here. We also remind you that preliminaries, as well as, some helper code (uploaded on iCorsi) for this assignment have been presented in the lecture (see slides towards the end of Sec. 3.2).

1.1 Text data

Many public domain books are available in text format 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. (5 pt) Report important properties of the data: character vocabulary size, number of characters in the file, number of lines in the file, and any other interesting observations about the structure of the text (e.g. usage of capitalized letters, contractions, paragraphs, line lengths, 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? (max. 2 sentences)

1.2 Batch construction

We consider a very long string representing the whole book (the text file you downloaded). Backpropagating gradients through an RNN which is enrolled 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. The length of the chunk is thus the BPTT span.

- 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.

To improve efficiency, we train on a batch of text chunks such that multiple chunks are processed in parallel. **The implementation for reading a raw text file and creating a batch generator which is compatible with the description above has been provided and commented in the lecture (and uploaded on iCorsi). 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. (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: 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 of the code, 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 Model and Training

1. (10 pt) **Implement** an LSTM language model with an input embedding layer, multiple LSTM layers using nn.LSTM, and a final softmax classification layer to predict the next character.

2. (20 pt) Our goal is to generate texts using the trained LSTM language model. To be more specific, we will provide a *prompt* (i.e. a segment/beginning of a text) to the language model, and let the model complete the text (for a length that you specify). **Implement** a greedy decoding algorithm for such a completion, i.e. at each step, take the character with the highest probability according to the model, and use it as the input to the model in the next step. We recommend you to implement this not as a part of the `forward` function of the model, but as a function which calls the `forward` function. Remember: hints were presented in the lecture/helper code.

3. (5 pt) To the implementation above, 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. Again, hints were presented in the lecture/helper code.

4. (10 pt) **Implement** the code for training. As has been described in point 3, the hidden states of the recurrent layers must be passed from one batch to the next, while not allowing the error signal to propagate across batches during training. For that you can apply `detach` function to the hidden state tensors. Make sure that your code allows you to monitor the following measures for every n training steps (with a reasonable choice of n):

   - The perplexity of your model on the training data (on the batch level).

- The model's text generation ability, by decoding always from the same prompt of your choice (e.g. "Dogs like best to" or the beginning of a story in the training set).

The perplexity of a language model p for a text w_1^N (with a start token w_0) is defined as:

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

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

5. (6 pt) Train an LSTM language model. We commend 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 0.001.
   - 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)` assuming that you defined your PyTorch model as `model`.

   We do not make use of any validation data: you can consider the model to be well trained when its training perplexity is below 1.03. You should be able to achieve this performance after about 10 minutes on a GPU. Report the evolution of both perplexity and text generation quality.

6. (4 pt) Report what you observe by trying two other values for learning rate and BPTT span independently (i.e. four more training runs in total).

7. (7 pt) Once your model is trained, run greedy decoding using the following prompts:
   (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.

   Report the output you obtain for each case. Provide a concise global comment (max 4 sentences. discussing e.g. Is the output meaningful? Is the model capable to produce novel texts?...)

8. (3 pt) Using the trained model, run decoding with sampling using prompts (a) and (b) from the previous question. Provide a concise global comment (max 2 sentences).

9. **(Bonus, 5 pt):** Be creative! Use other types of texts (e.g. the Python code from your last assignment), and re-run the experiments above. Comment.

2 Questions [6/100 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. (2 pt) In this assignment, we looked at the next-word prediction in text as a sequence prediction problem. Give two other examples of sequence prediction problems that are not based on text. (max. 2 sentences)

3. (3 pt) What is the vanishing/exploding gradient problem? And why does this affect the models ability to learn long-term dependencies? (max. 4 sentences)

Submission

You should deliver the following by the deadline stipulated on iCorsi3:

- **Report**: a single *pdf* file that clearly and concisely provides evidence that you have accomplished each of the tasks listed above. The report should not contain source code (not even snippets). Instead, if absolutely necessary, briefly mention which functions were used to accomplish a task.
- **Source code**: a single Python script that could be easily adapted to accomplish each of the tasks listed above. The source code will be read superficially and checked for plagiarism. Therefore, if a task is accomplished but not documented in the report, it will be considered missing. Note: Jupyter notebook files are not acceptable.

Please carefully read the instructions above to prepare your submission. Failure to stick to these rules may result in reduction of points.