INF265 - Weeks 14-15: LSTM-based character level language models

Pierre Gillot   Natacha Galmiche

April 19th 2021

1 Introduction

1.1 General instructions

Every two weeks you will be given an assignment related to the associated module. There are 3 weekly group sessions available to help you complete the assignments, you are invited to attend one of them each week (please choose one group session and stick to it unless exceptional case). Attendance is not mandatory but recommended. However, **assignments are graded and not submitting them or submitting them after the deadline will give you no points**. The grading system is detailed here: `https://mitt.uib.no/courses/27468/pages/general-information`.

- **Format: Jupyter notebook (single file)**
- **Deadline: Sunday 2nd May, 23:59**
- **Submission place:** `https://mitt.uib.no/courses/27468/assignments/44439`

1.2 Overview

In this assignment, we aim to design a simple character level language model in order to be able to generate new text similar to the text trained on during the learning stage. More specifically and as implied by the name, character level language models operate at the level of characters (as opposed to words): given a fixed length sequence of characters, these models are trained to guess what could be the next character. Owing to the sequential nature of the task, these models pair well with the so-called recurrent neural networks, a class of neural networks able to process sequential input by updating their internal state. Of particular interest are LSTMs (acronym for long short-term memory), a category of recurrent neural networks designed around the notion of "gates" in order to better control the gradient flow and to mitigate the vanishing of the gradient problem. In that sense, the connection between LSTMs and recurrent neural networks is comparable to that between residual networks and traditional feed-forward networks.

In a nutshell, the main goal of this assignment will be for you to learn how to set up a whole pipeline with limited guidance in order to solve a natural language processing task.

1.3 Useful resources

- Stanford lecture on RNNs: `https://youtu.be/6niqTuYFZLQ`
- Project Gutemberg: `https://www.gutenberg.org/`

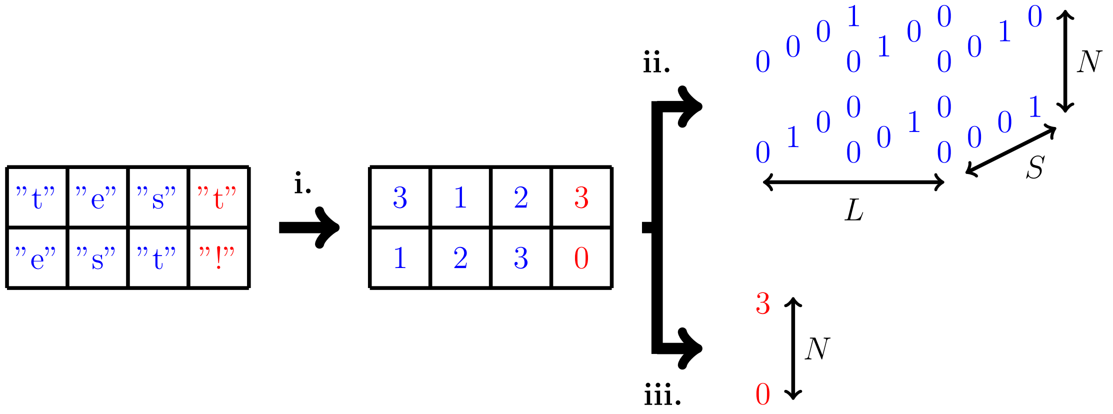
2 Exercise

1. Download a text document that you like. The project Gutemberg is a good place to obtain legally interesting books !
2. Load and clean up the text. You will remove all accents, force all letters to be lower case, remove all occurrences of "\n", "\t" and "\r" and make sure every two consecutive words in the text are separated by a single space " ". You can also remove part (or all) of the punctuation.
3. Create a character-based vocabulary from the preprocessed text: every appearing character will have a unique integer ID. You can use a dictionary structure to store the vocabulary.
4. Generate the data (X, y) for a character level language model:
   (a) Decide of a sequence length L.
   (b) For every sub-sequence formed from $L + 1$ consecutive characters in your text (with stride 1):
      i. Convert every character from that sub-sequence into its corresponding integer ID.
      ii. Convert the L first integer IDs into one-hot vectors (whose size will be equal to the size of the vocabulary) and store them in the features matrix X.
      iii. Store the last integer ID in the labels vector y (remember that in pytorch, labels do not need to be one-hot encoded).

   Assuming N is the number of such sub-sequences and S is the size of the vocabulary, the above procedure should yield two tensors X and y of respective shape (N, L, S) and $(N,)$.

   Let us illustrate this with a toy example. We consider the following text: "test!". The vocabulary is {0: "!", 1: "e", 2: "s", 3: "t"} thus $S = 4$. We choose to fix the sequence length to $L = 3$, which results in $N = 2$

sub-sequences ["t", "e", "s", "t"] and ["e", "s", "t", "!"]. The data would therefore be (X, y) as returned by the following procedure:



Hint: Extracting the sub-sequences can be done efficiently using the 'torch.nn.functional.unfold()' method. The one-hot encoding can be vectorized as well, using the 'torch.scatter()' method in order to fill a zeros tensor.

5. Shuffle then split the data between train and validation sets. You probably want to keep most of the data available for training.

6. Implement a LSTM classifier designed to predict the next character from a sequence of L consecutive characters. A good starting point would be to stack a 'torch.nn.LSTM()' module followed by a 'torch.nn.Linear()' layer. **Hint:** For this task, we need a "many-to-one" type of LSTM.

7. Implement the training loop. You will choose carefully the loss function and a metric that are suitable for the task. Your code will print the loss and the chosen metric at the end of every epoch, both for the train and the validation sets.

8. Finetune your model (for simplicity, no model selection pipeline required in this exercise): you will play with the hyperparameters of your model, including (non-exclusively) the sequence length, the hidden size of the LSTM module, the number of layers in the LSTM module, the batch size and the weight decay.

9. Train your model and analyze its performance.

10. Use your trained model to generate new text:

    (a) Choose a "seed sequence" consisting of L characters of your choice (among the characters in the vocabulary).

(b) Encode the seed sequence the same way you encoded X in Question 4 (in this case, $N = 1$).

(c) Pass your encoded seed sequence through your trained model.

(d) Predict the next character as the argmax of the softmax activation on the output.

(e) One-hot encode the predicted character.

(f) Update the seed sequence: remove the encoded character in first position of the encoded seed sequence, then add the encoded predicted character in last position of the encoded seed sequence.

(g) Repeat (c)-(f).

11. What problem seems to occur with the previous procedure ?

12. Modify the procedure described in Question 10: rather than predicting the next character as the argmax of the softmax activation on the output, you will instead sample it from the probability distribution given by the softmax activation on the output.

13. Entertain us by generating amusing text !