Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Alan Akbik

Berlin, 24.05.2024
Lehrstuhl Maschinelles Lernen

Introduction to Natural Langauge Processing

Exercise 5

Submission: Hand your homework until 03.06.2024 (Monday), 23:59 via Github Classroom. Each exercise is worth a total of **10 points** and you need **80%** (of total points over all exercises) to be admitted to the final exam. Sometimes, there are optional tasks that will give you extra points. Exercises in Github Classroom should be completed and submitted individually. However, we encourage you to work on the problems in teams. Please note the information available in Moodle: `https://hu.berlin/nlp24-moodle`.

Github Classroom: `https://classroom.github.com/a/frklxFXf`

Task 1 (LM preparation) — 3 + 3 = 6 points

In this task, we are preparing an LSTM implementation for future training as a language model in PyTorch. As you know from the lecture, LSTM solves a problem of vanishing gradients by introducing a memory cell and a neural gating mechanism that controls which information is erased/written/read, which becomes especially important when trying to generate longer text sequences.
You are provided with most of the data-loading logic, together with an LSTM constructor. You will need to:

(a) Implement data processing that is specific to the language modeling task. We rely on the data preparation logic similar to the one in the previous exercise. But now it allows for both character- and word-level tokenization. In order to complete the data processing, you need to complete the `make_index_vectors()` method in `data_util.py`. This method performs one-hot encoding of the inputs or targets that are then fed into the LSTM.

(b) Implement the `forward()` pass of the LSTM in `model.py`. Inside the forward pass, you will need to calculate the loss as well.

Note: we will be training on mini-batches in this exercise, therefore pay attention to sequence padding and tensor dimensions!

Task 2 (Small LM training) — 2 points

In this task, we will be training an LSTM on a small dataset containing verbalized mathematical equations. You are provided with the training loop implementation in `train.py`. The main method in `train.py` showcases how to utilize the configuration from `config.py` to configure your model and training hyperparameters. We expect you to execute at least two training runs: one using character-level tokens, and one using word-level tokens.

- Execute a trainig run on the `data/sanity_check.txt` dataset first to check if your implementation from Task 1 works as expected before training on the target dataset.

- Then, after you've made sure that everything works fine, proceed to train your models on the `data/equations.txt` dataset.

Training on a smaller dataset can be done locally on the CPU within a reasonable time frame, and thus you don't need to worry about computational costs too much, apart from the cases when you drastically increase the amount of hidden layers or the hidden layer size.
Nevertheless, you will definitely be able to see an impact of different tokenization strategies on the training time.
Play around with different hyperparameters to find the best performing combination. The learning rate is always drawn from a uniform distribution depending on your Github username that you pass on to the `generate()` method in the configuration (e.g. refer to how it is called from the main method of `train.py`). The results of your runs are recorded automatically in the `runs/` folder. Aim for the final test perplexity of under 10 for the word-level model, and under 2 for the character-level model.
Summary:

- Train at least 2 LSTMs on the `data/equations.txt` dataset: one using character-level tokens, one using word-level tokens
- Learning rate is fixed based on your Github handle, but you are free to choose other hyperparameter values
- Try to achieve the test perplexity of under 10 with the word-level model, and under 2 with the character-level model

Task 3 (Large LM training) **2 points**

In this task, we are training an LSTM on a larger dataset consisting of motivational quotes: `data/motivational_quotes.txt`. Again, you will need to execute two training runs, but this time using two different mini-batch sizes, namely 4 and 32 (keep them fixed).
When it comes to training on a larger dataset with longer text sequences, a GPU is often required in order to complete the training within a reasonable time frame. In case you don't have access to a GPU, we suggest you to use Google Colab for this exercise, as it provides free access to computing resources, including GPUs. For more details regarding Google Colab setup and code execution, please refer to the exercise slides.
Again, similarly to Task 2, we encourage you to play around with different hyperparameters to find the best performing combination. However, given the constraints posed by Google Colab, don't go too overboard with the amount of hyperparameters that you test. Aim for the final test perplexity of less than 500 for your best run.
Summary:

- Train at least 2 LSTMs on the `data/motivational_quotes.txt` dataset: one with batch size 4, one with batch size 32
- Learning rate and batch size are fixed, but you are free to choose other hyperparameter values
- Make sure that you pass your model and inputs/targets to the GPU device to enable GPU training
- Try to achieve the test perplexity of under 500 (doesn't matter with which batch size you achieve this)