EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Computer Science Department
Neuro-Cognitive Modeling

Summer Term 2022

Recurrent and Generative Neural Networks

Exercise Sheet 05

Release: June 17, 2022 Deadline: June 30, 2022 (11:59pm)

General remarks:

- Download the file `exercisesheet05.zip` from the lecture site (ILIAS). This archive contains files (Python classes), which are required for the exercises.
- Add a brief documentation (pdf) to your submission. The documentation should contain protocols of your experiments, parameter choices, and discussions of the results.
- When questions arise start a discussion in discord or contact us via email:
  Jannik Thümmel (`jannik.thuemmel@uni-tuebingen.de`)
  Sebastian Otte (`sebastian.otte@uni-tuebingen.de`)

Introduction

In this exercise sheet we will implement and train an *LSTM* and a *Transformer*-like architecture that can generate Tolkien texts on character level.

Text generation is considered a particularly data and resource hungry task. In order to reduce the complexity of the problem and make it still computationally tractable on CPU, we will use comparably small datasets and models. Accordingly, the goal of this exercise is not to produce state-of-the-art results but rather to gain experiences on text generation with neural architectures. An example of what your model might be able to produce in the end is displayed below, including a DeepL-translation to German:

Tolkien	`‘fare you well!’ said ingold; and the men made way for shadow fax, and he passed through a narrow gate in the wall.`
Model	`‘fare you well!’ said ingold; and the men made way for fax, and she the madow ax, and shrow the gathrough ate all.`
DeepL	`‘Lebt wohl!’ sagte Ingold, und die Männer machten Platz für Fax, und sie die Wiesenaxt, und die Scharfe aß alles auf.`

Excercise 1
Text Generations with LSTMs [30 points]

(a) Dataset Generation [10 points]

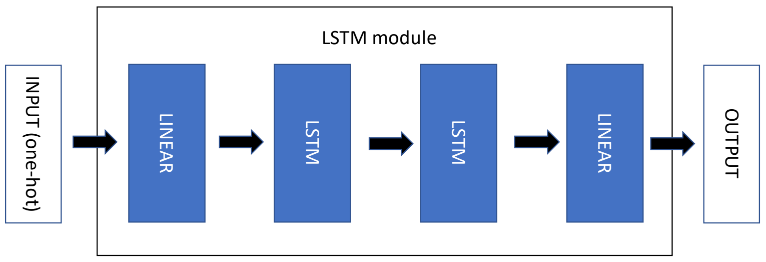
In this exercise you will process words on character level. These characters are represented as one-hot encoded vectors: a vector of the size n (size of the alphabet) that contains only *zeros*, except for the index of the letter it represents, which is a *one*. Here is an example of a one-hot encoding for the alphabet {'a', ..., 'z'}:

$$\begin{aligned} \text{`a'} &\rightarrow \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}^\top \\ \text{`b'} &\rightarrow \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \end{bmatrix}^\top \\ &\vdots \\ \text{`z'} &\rightarrow \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}^\top \end{aligned}$$

In the folder `data` you can find the file `text.txt`[1]. This file is to be converted into sequences of one-hot encoded vectors, which can be done with the Python script `data_preparation.py`. There is a prepared function `txt_to_npy` that already provides a basic framework to convert text lines into according one-hot sequences, which you to have to complete (watch for the TO-DOs in the code). We recommend to implement a function *char_to_on_hot* within the script file `utils/helper_functions.py` to conveniently reuse it later. Write the sequences as separate `sample_****.npy` files (replace the stars with a running index) into the same directory, where the file *alphabet.pny* was written to.

(b) LSTM Implementation [10 points]

Implement an LSTM network in the `models/lstm/modules.py` file, following the scheme illustrated in the figure below. A Pytorch documentation can be found here[2] (essential building blocks are `nn.LSTM` and `nn.Linear`).



(c) Model Training [10 points]

Run the `models/lstm/train.py` script and evaluate your model with the `models/lstm/test.py` script. Consider the `models/lstm/config.json` file to set particular parameters. Report on what the model learns throughout the epochs and give an interpretation why it does so.

[1]Containing a chapter of J.R.R. Tolkien's The Lord of the Rings trilogy taken from `https://ae-lib.org.ua/texts-c/tolkien__the_lord_of_the_rings_3__en.htm`
[2]`https://pytorch.org/docs/stable/index.html`

Excercise 2
Text Generation with Transformers [70 points]

In this task, you will implement the decoder part of a transformer to do the same as the LSTM above. The provided code for this exercise again contains further comments. Code section where you have to add your implementations are explicitly marked with TODOs. Relevant are the lecture slides 9 – 14 in `RecAndGenANNs_08_TransformersAndGPT3.pdf`.

(a) Multi-Head-Attention Module [30 points]

Implement the multi-head-attention module in `models/transformer/modules.py`, including the attention mechanism. Note that, in order to perform multi-head instead of single-head attention, the query, key, and value vectors have to be split into k heads, which will all perform their own attention computation on the data parts they receive, respectively.
Implementing multi-head-attention is often easier utilizing torch.einsum() [3] for the matrix multiplications and einops.rearrange() [4] for the reshaping operations.

(b) Feed-Forward Module [10 points]

In the `models/transformer/modules.py`, implement a feed forward module consisting of a ReLU layer, followed by a dropout layer and another linear layer.

(c) The Decoder [20 points]

The decoder has to be implemented in the `models/transformer/modules.py` file as well. It consists of a multi-head-attention module followed by a feed-forward module. Note that, instead of feeding the inputs straight through the different modules, the transformer works best when predicting residuals (e.g. something like `x = x + multi_head_att(x)`). You can stack multiple decoder blocks to improve the performance of the model. Note, however, that stacking increases the computation time and remember that the aim of this exercise is to understand transformers and not to produce the highest-quality outputs.

(d) Model Training [10 points]

Train and validate your transformer decoder model and contrast your findings to the results from the LSTM.
Try training your transformer without dropout. Can you explain what effect this would have?

[3] https://pytorch.org/docs/stable/generated/torch.einsum.html
[4] https://einops.rocks/api/rearrange/