EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Computer Science Department
Neuro-Cognitive Modeling

Winter Term 2024/2025

Recurrent and Generative Neural Networks

Exercise Sheet 03

Release: November 18, 2024   Deadline: December 02 , 2024 (11:59am)

General remarks:

- Download the file `exercisesheet03.zip` from the lecture site (ILIAS). This archive contains files (Python classes), which are required for the exercises.
- Ideally, the exercises should be completed in teams of two students. Larger teams are not allowed.
- **Add a brief documentation (pdf) to your submission**. The documentation should contain your **names**, **matriculation numbers**, and **email adresses** as well as protocols of your experiments, parameter choices, and a discussion of the results.
- When questions arise, please start a forum discussion in ILIAS or contact us via email:
  Manuel Traub (`manuel.traub@uni-tuebingen.de`)
  Matthias Karlbauer (`matthias.karlbauer@uni-tuebingen.de`)

Introduction

In this exercise sheet, we will compare *TCN*, *LSTM*, and *Transformer*-like architectures for Tolkien text generation 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 [40 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 size n, i.e., the 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.npy* was written to.

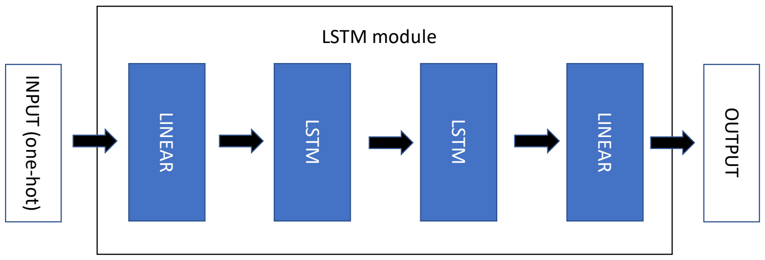
(b) Text Generation [30 points]

Learn to predict the next character with three different architecture types you have already implemented previously: LSTM, TCN, and Transformer. In all cases, you may use your own implementation or take the one we provide. Experiment with different model sizes and learning rates and report your observations, also touching on differences across architectures.

Please include the checkpoints of the trained models in your submission.

LSTM

Put together an LSTM network in the file under `models/lstm/modules.py`, 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`).



[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`

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 what the model learns throughout the epochs, i.e., what the model does after a few epochs and later into the training, and give an interpretation why it does so.

TCN

Run the `models/tcn/train.py` script and evaluate your model with the `models/tcn/test.py` script. Consider the `models/tcn/config.json` file to set particular parameters. Report what the model learns throughout the epochs, i.e., what the model does after a few epochs and later into the training, and give an interpretation why it does so.

Transformer

Run the `models/transformer/train.py` script and evaluate your model with the script located at `models/transformer/test.py`. Consider the `models/transformer/config.json` file to set particular parameters. Note that the transformer we provide you with does not implement positional embedding. Report what the model learns throughout the epochs, i.e., what the model does after a few epochs and later into the training, and give an interpretation why it does so.

Excercise 2 [60 points]

In this exercise, we will train autoencoders to compress 64×64 images of faces into a low-dimensional latent state and subsequently reconstruct the images from that latent state. We will compare non-regularized autoencoders versus beta variational autoencoders. Due to the model and dataset size, we strongly recommend using graphics cards for this exercise (in particular, for training the models). For this exercise, you will need `numpy==1.23`, `torch`, `torchvision`, and `tqdm`. See **environment_setup.txt** to setup an according conda environment.

Disclaimer: The models trained here are not state-of-the-art. Therefore, please do not expect reconstructions of very fine detail and quality.

(a) Model training [20 points]

Train autoencoders with different values of beta, for example, $\beta \in [0, 4, 20]$, to visualize the effect of the KL divergence term in the variational loss. A model can be trained using the command
`python main.py --beta 0 --ckpt_name vae_beta_0`

Please include the checkpoints of the trained models in your submission.

(b) Generate Images from Random Latent Codes [10 points]

Compare the different autoencoders in terms of the images that they reconstruct from random latent codes. For this, you can generate latent codes $z \sim \mathcal{N}(0, 1)$ and forward those through the decoder of the model. Please complete the `generate_from_latent` function in the Solver class of `solver.py`. To evaluate a trained model called `vae_beta_0` on this task, call

```
python main.py --task random_generation --ckpt_name vae_beta_0
```

Please report how the images differ for varying β-values and provide brief interpretations and explanations.

Hint: Make sure to call `torch.sigmoid()` on the output of the decoder.

(c) Latent State Analysis [20 points]

Investigate which latent neuron corresponds to what feature in the space of the reconstructed images. For example, one particular neuron could be responsible for changing the hair style or the hair color of the generated person. Please complete the `latent_analysis` function in the Solver class of `solver.py`. To evaluate a trained model called `vae_beta_0` on this task, call

```
python main.py --task latent_analysis --ckpt_name vae_beta_0
```

Please contrast your observations for varying β-values and provide brief interpretations and explanations.

(d) Rotate Face [10 points]

Given some image as input, we want to rotate the person shown on that image along the z-axis, e.g., simulating pictures taken from different angles at the same height. Please complete the `rotate` function in the Solver class of `solver.py`. To evaluate a trained model called `vae_beta_4` on this task, call

```
python main.py --task rotate --ckpt_name vae_beta_4
```