Computer Science Department
Neuro-Cognitive Modeling

Winter Term 2024/2025

Recurrent and Generative Neural Networks

Exercise Sheet 04

Release: December 02, 2024 Deadline: December 16, 2024 (11:59am)

General remarks:

- Download the file `exercisesheet04.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) and model checkpoints** 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:
  Jan Prosi (`jan-gerhard.prosi@uni-tuebingen.de`)
  Matthias Karlbauer (`matthias.karlbauer@uni-tuebingen.de`)

1 Introduction

In this exercise, we will predict two-dimensional waves as a spatiotemporal process that circularly expands from a point source outwards, as visualized in Figure 1.

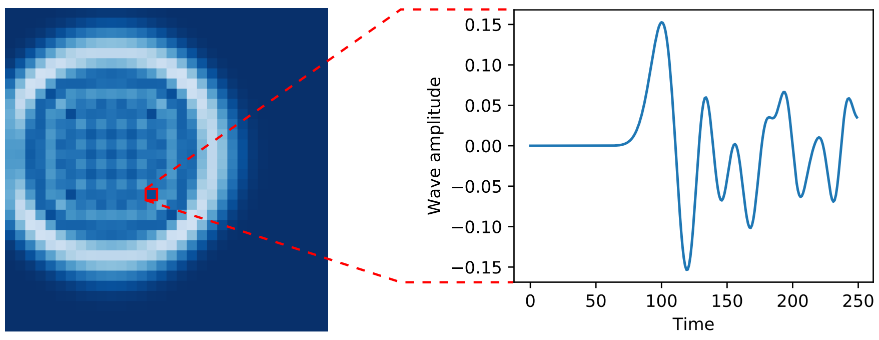

Figure 1: Two-dimensional wave propagating over time through space.

Exercise 1: ConvLSTM [20 points]

Beyond modeling the two-dimensional wave, we use this exercisesheet to learn a bit about the `hydra-core` Python package, which facilitates the book keeping of training deep learning models by offering a handy command line interface and a convenient dumping of configurations. The command line provide facilities to modify architecture, data, training configurations and other without having to touch the actual code. For each trained model, a hidden `.hydra` directory is created to run an evaluation with the exact same configurations as used for training.

When operating on the TCML cluster, please do not compute on the main node but execude according `.sbatch` files. You can use the singularity container provided in `/common/share/LectureRecAndGenANNs/exerciseshee04/`

(a) Data Generation [0 points]

For data generation, please refer to the data generation section in the `README.md` in the downloaded zip. For convenience, you may paste the content of the `REAMDE.md` file into a markdown viewer, such as `https://markdownlivepreview.com/` and consider the hints and tips to exemplarily visualize and animate the wave. Following the instructions, you will generate train, validation, and test wave data with two different velocities.

(b) Predicting Slow vs. Fast Waves [20 points]

We provide you with an implementation of the `ConvLSTM` model and ask you to train it on the slow and fast dynamics. Please refer to the `README.md` for information about how to specify the slow or fast wave dynamics data for training.

Report your findings by providing exemplary plots and explaining the success of the `ConvLSTM` model and discuss reasons why the model fails to simulate the dynamics if it does so.

Exercise 2: UNet [80 points]

The U-Net we are considering here consists of an encoder and a decoder. The encoders and decoders are set up from encoder and decoder modules. Each encoder/decoder module performs a down/up scaling operation as well as non-scaling operations using ConvNeXt Blocks. During the down/up sampling operations the spatial dimensions are down/up sampled whereas the channel number is increased/decreased. This exercise focuses on the UNet implementation and training, the ConvNeXt Blocks come pre-implemented in the provided code.

The code for the model allows to adjust the following network properties:

- `c_in`: Number of input channels. In our implementation we concatenate the input frames along the channel dimension. The number of input frames is given by `context_size`, which can be adjusted in the data config.
- `c_out`: Number of output channels, which is 1 as we have one output frame.

The subsequent lists determine the encoder properties and are reversed to determine the decoder properties. The length of the lists determines the number of encoder/decoder modules and allow you to adjust the model depth (number of hierarchies), width and scaling operations.

- `c_list`: Input and output channels for the corresponding encoder/decoder module. (Therefore its length is the number of encoder/decoder modules + 1)

- `spatial_factor_list`: The spatial factor that the corresponding encoder/decoder module down/up scales with.
- `num_block_list`: Number of subsequent non-scaling ConvNeXt Blocks in the corresponding encoder/decoder module.

For the UNet we use a different dataset for training and validation even though it is sampled from the same data as we used for the ConvLSTM. For testing we stick to the `wave.yml`, that we also used for the ConvLSTM. The configurations of the dataset that we will be using are given by `wave_unet.yml`. The description can be found in the file. One sample of the dataset consists of an input of length `context_size` and a target of length `max_rollout_steps` and is sampled from all time steps across the different wave time series.

One-Step-Ahead Training[50 points]

1. Write the code for the UNet.
   - In `models/u_net/u_net.py` you can find the code for the UNet. The core functionality of the classes are missing. Please complete the code. Do not stick to the architecture of the original UNet by Ronneberger et al. too closely but adjust it according to the needs of the task.
   - Include a drawing of your U-Net architecture with arbitrary hyperparameters.
2. Train the model.
   - Use the slow wave data with `skip-rate 1`
   - Adjust the model architecture, configurations of the dataset and the training. Play around with the network architecture by adjusting the hyper parameters until it converges well and produces a reasonable evaluation.
     - Feel free to prototype on less data and smaller models before up-scaling for your final run. Keep an eye on the batch size to optimize vram usage.
     - The loss should decrease by at least one order of magnitude. For now only train on one-step-ahead predictions, do not implement the closed-loop training.
3. Discuss the training and evaluate the model.
   - Evaluate the model with `evaluate.py`
   - Show and discuss the loss convergence as well as the evaluation of the model before you fine-tuned the configurations and hyper parameters vs after the fine-tuning. You can run `tensorboard --logdir outputs` and open `http://localhost:6006/` in a browser to inspect and contrast training and validation curves of all models in the `outputs` directory.
   - For your fine-tuned model, briefly discuss the wave amplitude plot and the dynamics in the animation before and after closed-loop evaluation. Compare to the evaluation of the ConvLSTM.
   - List the hyper parameters as well as data and training configurations that you consider most important. Briefly describe how you adjusted them and their effects.

Closed-Loop Training[30 points]

1. Implement closed-loop predictions within the training of the model.
    - First of all, continue to train on the slow wave data.
    - Come up with a smart way to integrate closed-loop training by adjusting the `rollout_steps` parameter. Feel free to change the parameter during training.
    - Shortly describe how you trained the model closed-loop.
    - Briefly compare the evaluation of your model that has been trained closed-loop to the one that has not been trained closed-loop.
2. Train and evaluate the model on the fast wave data (`skip rate 3`)
    - If the model does not converge well, do some readjustments.
    - Briefly compare the evaluation of the model to the evaluation of the ConvLSTM on the fast wave data.