Time Series Models: From Statistics to AI

lecturer: Daniel Durstewitz
tutors: Lukas Eisenmann, **Christoph Hemmer**, Alena Brändle, Florian Hess, Elias Weber, Florian Götz
SS2025

Exercise Sheet 6

Regulations

Please submit your solutions via Moodle in teams of **2 students**, before the exercise group on **Wednesday, June 4th, 2025**. Each submission must include **exactly** two files:

- A `.pdf` file containing both your Jupyter notebook and solutions to analytical exercises. The Jupyter notebook can be exported to pdf by selecting **File** → **Download as** → **pdf** in JupyterLab. If this method does not work, you may print the notebook as a pdf instead. Your analytical solutions can be either scanned handwritten solutions or created using LaTeX.
- A `.ipynb` file containing your code as Jupyter notebook.

Both files must follow the naming convention:
`Lastname1-Lastname2-sheet06.pdf`
`Lastname1-Lastname2-sheet06.ipynb`

Getting Familiar with PyTorch

In this exercise, you are going to train a Recurrent Neural Network with PyTorch. In order to do this, you need to download and install PyTorch first, by following the instructions on
`https://pytorch.org/get-started/`.

We provide a template notebook (`rnn_template.ipynb`) for implementing the RNN, the training algorithm and the evaluation procedure.

1 Understanding an RNN

The aim of this exercise is to capture a simple dynamical system (in this case a sinusoidal oscillation) with a recurrent neural network (RNN). Therefore, define a latent RNN model of the following form:

$$z_t = \tanh(Cx_{t-1} + Wz_{t-1} + h) \tag{1}$$
$$\hat{x}_t = Bz_t + c \tag{2}$$

where W, C, B are weight matrices and h and c bias vectors. The observations are denoted by x_t, the prediction of the RNN $\hat{x}_t$ and the latent state z_t. Depict the RNN graphically (in a flow chart).

2 Training the RNN

The file `sinus.pt` contains data of 41 time steps from a two-dimensional sinusoidal oscillation:

$$x_t = \begin{pmatrix} \sin\left(\frac{t}{10}\pi\right) \\ \cos\left(\frac{t}{10}\pi\right) \end{pmatrix}, \quad t = 0, \ldots, 40 \tag{3}$$

Complete the template for model and training loop given in `rnn_template.ipynb`. In the last section of the script, generate a **freely** predicted trajectory $\{\hat{x}_t\}$ using the trained mmodel, which is then plotted. The new trajectory must be 5 times longer than the original one and should give you an impression if the model learned the oscillation correctly. Try to find the minimum number of hidden units in the RNN (dimension of z_t) such that it reconstructs the dynamics to satisfaction.[*]

3 Optimizing Training

Gradient descent updates the parameters θ with gradient g and learning rate λ: $\theta \leftarrow \theta - \lambda g$. Observe the influence of the learning rate on the dynamics of the learning process:

1. Plot the losses as a function of number of gradient steps and vary the learning rate in gradient descent. How does the loss behave depending on the learning rate?
2. A scheme to speed up learning is to use momentum which keeps a moving average over the past gradients: $v \leftarrow \alpha v - \lambda g$, $\theta \leftarrow \theta + v$. How do the dynamics change when the learning rate is adapted with momentum?
3. How does the adaptive learning rate of the Adam (Kingma and Ba, 2014) optimizer perform (`torch.optim.Adam`) in contrast to stochastic gradient descent (SGD)?
4. Can you identify bifurcations in the learning dynamics from eye-balling the loss curve? Plot the generated trajectories before and after a bifurcation to observe how the optimization changes the network dynamics.

4 Mini-batching

A common strategy to improve training is mini-batching. From the original data set, the model is trained on a randomly drawn subset in every epoch. For time-series $\{x_t\}_{t=1,\ldots,T}$, that means drawing multiple (not just one) sub-sequences $\{x_t\}_{t=s,\ldots,s+l-1}$ of length l that start at random time-points s.

Implement mini-batching into your training procedure by implementing epoch-wise drawing of sub-sequences, and concatenating them along a new dimension in the input vector to RNN. In pytorch, the dimensionality of batched data tensors for RNNs conventionally is (Time, Batch, Features). Be careful not to mess this up. You should notice improved stability during training.

[*]Around 1000-2000 epochs should suffice to reconstruct the oscillation.