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 7

Regulations

Please submit your solutions via Moodle in teams of **2 students**, before the exercise group on **Wednesday, June 11th, 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-sheet07.pdf`
`Lastname1-Lastname2-sheet07.ipynb`

1 Backpropagation Through Time

Given $T \in \mathbb{N}$, a univariate RNN:

$$z_t = \varphi(uz_{t-1} + vx_{t-1}), \quad y_t = wz_t \tag{1}$$

with a differentiable function φ, where $\forall t \in \{0, \ldots, T\} : u, v, w, x_t, y_t, z_t \in \mathbb{R}$, and a squared error loss:

$$L(u, v, w) = \sum_{t=1}^{T} (x_t - y_t)^2 \tag{2}$$

Analytically find formulas for the gradients:

$$\nabla_u L, \quad \nabla_v L, \quad \nabla_w L \tag{3}$$

Find conditions for the gradients such that they neither vanish nor explode, that is:

$$\lim_{T\to\infty} \nabla_u L \notin \{0, \pm\infty\} \tag{4}$$

2 The Influence of Noise on the Data

The file `noisy_sinus.pt` contains data of 41 time steps from a two-dimensional sinusoidal oscillation with Gaussian white noise:

$$x_t = \begin{pmatrix} \sin\left(\frac{t}{10\pi}\right) + \varepsilon(t) \\ \cos\left(\frac{t}{10\pi}\right) + \varepsilon(t) \end{pmatrix}, \quad t = 0, \ldots, 40, \quad \varepsilon(t) \sim \mathcal{N}(0, 0.2) \tag{5}$$

The objective is to train a RNN to learn the underlying dynamics of data that has been corrupted by noise, without overfitting to the noise itself. If the network has too few hidden units, it may fail to capture the true dynamics; too many, and it risks overfitting by modeling the noise rather than the actual signal. Since identifying the optimal number of parameters is challenging, a common approach is to begin with a larger model and apply regularization techniques to prevent overfitting.

Using your RNN from last week on the noisy data *, find a number of hidden units where the RNN clearly overfits the data †. Compare the performance of the following strategies to mitigate overfitting:

a) **L1-Regularization**: Add $\alpha\left(\sum_j |\theta_j|\right)$ to the loss function, where θ_j are all the parameters of the model. Try different values for α, starting at 0.1. To access θ_j, use `model.parameters()`.

b) **L2-Regularization**: Add $\alpha\left(\sum_j \theta_j^2\right)$ to the loss function and try different values for α, starting at 0.1. Equivalently, you can set the `weight_decay` argument of the Adam optimizer to α.

c) **Dropout**: Set the `dropout` argument of the `torch.nn.RNN` class to $d \in (0, 1)$. This sets a fraction d of the total weights to zero during every gradient step. **Important:** Before training, call `model.train()`, and after training, call `model.eval()`. This toggles dropout on and off appropriately.

Find arguments (briefly) why each of the techniques counteracts overfitting.

*Note: Use the Adam optimizer and mini-batching.

†You can deduce overfitting from the prediction plot.