Dynamical Systems Theory in Machine Learning & Data Science

lecturers: Daniel Durstewitz
tutors: Christoph Hemmer, Alena Brändle, **Lukas Eisenmann**, Florian Hess
WS2024/25

Exercise 9

To be uploaded before the exercise group on January 8, 2025

Prerequisites

In this exercise we will train Recurrent Neural Networks (RNNs) to predict the evolution of the chaotic Lorenz-63 system. We will train the RNNs using Backpropagation through time (BPTT). To this end, we will use the deep learning framework PyTorch, which provides all the tools we need. Hence, install PyTorch on your machine or coding platform at hand. A detailed installation guide is provided on the PyTorch website. We recommend installation via the Anaconda (or Miniconda) Python distribution.

1 Learning to predict the Lorenz system using RNNs

Consider a common RNN architecture of the form

$$\boldsymbol{z}_t = \boldsymbol{F}(\boldsymbol{z}_{t-1}, \boldsymbol{x}_t) = \phi(\boldsymbol{W}\boldsymbol{z}_{t-1} + \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{h}) \tag{I}$$

$$\hat{\boldsymbol{y}}_t = \boldsymbol{G}(\boldsymbol{z}_t) = \boldsymbol{B}\boldsymbol{z}_t + \boldsymbol{b}, \tag{II}$$

where $\boldsymbol{F}$ is an RNN and $\boldsymbol{G}$ its linear output layer with overall parameters $\theta = \{\boldsymbol{W} \in \mathbb{R}^{M\times M}, \boldsymbol{C} \in \mathbb{R}^{M\times N}, \boldsymbol{h} \in \mathbb{R}^{M}, \boldsymbol{B} \in \mathbb{R}^{N\times M}, \boldsymbol{b} \in \mathbb{R}^{N}\}$ and nonlinear activation function ϕ. In this exercise, we will train RNNs on the chaotic Lorenz-63 system. To train any RNN, we will proceed the following way: Given some training data in the form of a $T \times N$ dimensional time series $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$, we will arbitrary sample small sequences of length τ (hyperparameter) $\boldsymbol{x}_{t:t+\tau} := \{\boldsymbol{x}_t, \ldots, \boldsymbol{x}_{t+\tau}\}$ and try to predict the sequence propagated one step forward in time $\boldsymbol{y}_{t:t+\tau} := \boldsymbol{x}_{t+1:t+\tau+1}$. That is, the model is trained to perform one-step-ahead predictions: Given ground truth data $\boldsymbol{x}_t$ as input, the RNN updates its hidden state $\boldsymbol{z}_{t-1}$ according to $\boldsymbol{F}$ and then $\boldsymbol{z}_t$ is mapped back to the data domain using the linear output layer $\boldsymbol{G}$, yielding prediction $\hat{\boldsymbol{y}}_t$, which should ideally equal the (one step forwarded) ground truth data $\boldsymbol{y}_t$. Providing ground truth data during training to guide the dynamics of the RNN is generally referred to as *teacher forcing*. Our objective is to minimize the mean-squared-error (MSE) loss function between ground truth targets and predictions

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{y}_{t:t+\tau}, \hat{\boldsymbol{y}}_{t:t+\tau}) = \frac{1}{\tau N}\sum_{t=1}^{\tau} \|\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t\|_2^2 \tag{III}$$

To minimize this error, we employ (mini-batch) stochastic gradient descent (SGD). To this end, multiple sequences will be packed into a single tensor, and will be processed in parallel. The gradient used to perform the update is then the gradient averaged over the entire batch. The necessary gradients w.r.t. to all parameters will be computed using BPTT.

The file `sheet9_template.ipynb` provides code to load in data of the Lorenz-63 system, found in `lorenz_data.pt`. It also provides a training routine to train an RNN on the provided data in the manner discussed above.

TASKS

1. Would the RNN be able to learn the dynamics of the Lorenz-63 attractor if ϕ is a linear function? Explain.

2. Pytorch provides a implementation of the vanilla RNN introduced above through the class `nn.RNN`. Make yourself familiar with the structure by looking at the docs. Train multiple RNNs on the Lorenz system using $\phi = tanh$ or $\phi = ReLU$ varying the number of hidden units/neurons M. Choose a suitable sequence length ($\tau \in [30, 200]$) and the batch size to be $B = 32$. Train until the loss converges to a plateau, and plot the losses (use a logarithmic y-axis) for different M against the number of parameter updates. What do you observe?

3. To know whether our RNNs really have captured the dynamics of the attractor, we need to check whether the model provides accurate short-term forecasts, as well as capturing the dynamics of the limit set (long-term behaviour). To test the former, take a window from the Lorenz data of length $T = 1000$. Use the model with the lowest loss after training from the previous task and assess predictions multiple steps ahead. To this end, initialize the hidden state of the RNN with zeros. 'Warm up' the dynamics by teacher forcing the RNN for 100 time steps, that is, perform one-step-ahead predictions providing the ground truth data of the previous time step as input to the model, i.e. $\boldsymbol{z}_t = \boldsymbol{F}(\boldsymbol{z}_{t-1}, \boldsymbol{y}_{t-1})$ (this is the same that is done to form predictions during training!). Now, use $\boldsymbol{z}_{101}$ as the initial hidden state and iterate the RNN for the remaining T-100 time steps forward in time, by feeding the model its own predictions, $\hat{y}_t = G(F(\boldsymbol{z}_{t-1}, \hat{\boldsymbol{y}}_{t-1}))$. Compute the MSE between predictions and ground truth data, and plot the MSE against time steps excluding the warm up phase. Report the x-axis in units of the Lorenz-63's Lyapunov time ($\tau_{Lyap} = 1 \ / \ (\Delta t \cdot \lambda_{max}) = 1 \ / \ (0.01 \cdot 0.906) \approx 110$). For how many Lyapunov times does your RNN manage to accurately predict the system? Why can't the predictions align indefinitely?

4. To qualitatively check the long-term behaviour, generate a trajectory from your RNN by initializing the hidden state to zeros, and providing a random (yet sensible) initial condition $\boldsymbol{x}_1$ to the model to produce $\hat{\boldsymbol{y}}_1$. Again iterate on the RNN predictions for $T = 100,000$ time steps. Plot the ground truth attractor (the dataset) in 3d state space and overlay the model generated one. By eye-balling, argue whether your model has captured the long-term attractor dynamics.

Happy holidays and a wonderful new year!