Time Series Analysis & Recurrent Neural Networks

lecturer: Daniel Durstewitz
tutors: Georgia Koppe, Leonard Bereska
WS2019/2020

Exercise 7

To be uploaded before the exercise group on December 11th, 2019

Task 1. Discrete-time non-linear dynamics: fixed points, stability and bifurcations
Consider the univariate non-linear map

$$x_{t+1} = f(x_t, a, b) = a \cdot x_t + b \cdot tanh(x_t)$$

1. Plot the *return plot* of this map. By inspecting the plot say how many fixed points you expect the system to have and comment on their stability. Do this for the parameter values: I) $\{a, b\} = \{1, 3\}$, II) $\{a, b\} = \{0.5, -2\}$, III) $\{a, b\} = \{0.5, 3\}$ and IV) $\{a, b\} = \{1, 0\}$.

2. For all parameter sets specified above, plot the trajectory of the system when starting from the initial conditions $x_0 = -10$, $x_0 = -0.5$, $x_0 = 0$, $x_0 = 0.5$, $x_0 = 10$.

3. Confirm your intuitions by computing the fixed points (numerically) and their stability (analytically) for the parameter set III (fixed points: see equation 9.4, chapter 9.1.1. For numerical solutions you can use `scipy.optimize.fsolve` for python or `fzero` for matlab).

4. Plot the *bifurcation graph* as a function of $a \in [-7, 2]$ (x-axis) for $b = 5$ in which you display stable fixed points x^* in the range of $x^* \in [-5, 5]$ (y-axis). Solve this task by numerical simulations, i.e., for each value of a considered i) initialize the system from, e.g., 100 different initial conditions x_0 (with $x_0 \sim U(-5, 5)$), and ii) plot the value x_T (with T large enough to allow transients to settle), where $x_T \approx x^*$.
   How would you interpret the bifurcation graph at $a = -1.8$?

Task 2. Training an RNN in 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/`.
Now, consider a recurrent neural network with N neurons:

$$x_t = \Phi\left(\mathbf{A}x_{t-1} + I_t\right)$$
$$\Phi(y) = \tanh(y)$$

where $x_t \in \mathbb{R}^{N\times 1}$ is the output of the network at time t, $\mathbf{A} \in \mathbb{R}^{N\times N}$ the weight matrix and $I_t \in \mathbb{R}^{N\times 1}$ the input at time t.
You are given the following mapping from inputs to outputs:

$$I_3 = (1, 0, 0, 0, \ldots)^T \rightarrow \hat{x}_7 = (\bullet, \bullet, 1, 0, \bullet, \ldots)^T$$
$$I_3 = (0, 1, 0, 0, \ldots)^T \rightarrow \hat{x}_7 = (\bullet, \bullet, 0, 1, \bullet, \ldots)^T$$

where I_t is the input at time step t, $\hat{x}_t$ is the requested output of the network at time step t, and the dot $\bullet$ indicates that no specific output is requested for that unit at that time step (this means that only units 1 and 2 receive input while only units 3 and 4 have requested outputs, while all other units are

considered hidden units). The total length of the time series is $T = 10$.
The code in `working_memory_RNN.py` implements an RNN training procedure for this scenario in PyTorch. There, we present the network with a number M of mini-batches of size m, where the order of the trial types (either mapping I or mapping II) is randomized.

Make yourself familiar with the PyTorch framework by carefully studying the code provided and changing parameters. For $N = 5$, examine the convergence of the error function and the performance of the network after apparent convergence (recall/test phase) for different learning rates from the range $\alpha \in [10^{-5} \ldots 1]$ (explore a couple of different values). Compare that to training in networks with $N = 10$ units. Change the activation function to the rectified linear function (ReLU) and do a comparison.