LMU: Fakultät für Physik

Lecture:
When Machine Learning meets Complex Systems

PD Dr. Christoph Räth, Sebastian Baur, Daniel Köglmayr, Joel Steinegger

Exercise Sheet 3

(Please prepare your answers until Wednesday, December 3rd)

Exercise 3.1: Networks Create the following networks:

- A regular ring lattice with $N = 250$ nodes, where each node is connected to its 4 nearest neighbors.
- An Erdös-Rényi random network with $N = 250$ nodes and an edge probability of $p = 0.01$.
- A small-world network with $N = 250$ nodes, 4 edges per node and a rewiring probability of $p = 0.1$.
- A scale-free network with $N = 250$ nodes where each new node is connected to 2 nodes of the existing network (with preferential attachement).

Visualize each network in a sensible, preferably interactable, way. Calculate and plot the distribution and the mean value of the node degree, clustering coefficient and average path length.

Exercise 3.2: SINDy The SINDy algorithm (sparse identification of nonlinear dynamics) is a simple but powerful method for reconstructing the governing equations of a dynamical system. It was orginally introduced by Brunton et al. Here we will implement the most basic version of SINDy and apply it to reconstruct and predict the Lorenz System.

a) Load the time series $\mathbf{X}$ of the Lorenz system from moodle and quickly plot it to convince yourself that you have loaded the data correctly. It has shape (t, d) with $t = 10000, d = 3$. Alternatively you can also create your own dataset using the Runge Kutta implementations from previous sheets. Time step size is $\Delta t = 0.002$ and system parameters are $\sigma = 10, \rho = 28, \beta = 8/3$.

b) For the purpose of this exercise we'll now pretend to not know what system was used to generate this time series but we'll assume that it was generated by some dynamical system following a (stationary and deterministic) governing equations of the form

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)).$$

The vector $\mathbf{x}(t)$ denotes the state of a system at time t, and the function $\mathbf{f}(\mathbf{x}(t))$ represents the dynamic constraints that define the equations of motion. Further, we'll guess that these equations can be approximated by combining a set of polynomials of up to second order. Doing so turns every data point $\mathbf{x}_i = (x_i, y_i, z_i)$ into a so called feature vector $\mathbf{r}_i$, here given ass:

$$\mathbf{r}_i = (x_i, y_i, z_i, x_i^2, z_i^2, y_i^2, x_i y_i, x_i z_i, y_i z_i),$$

which incorporates the original data point and all possible combinations of second order terms. Write function that turns any three dimensional data point into such a feature vector. By applying this function to every data point, turn the whole data set $\mathbf{X}$ of shape $(10000, 3)$ into our feature matrix $\mathbf{R}$ of shape $(9999, 9)$

c) As a next step we need to define our target data. As we want to reconstruct an ODE, we want to have $\frac{d}{dt}\mathbf{x}(t) \approx \frac{\Delta \mathbf{x}}{\Delta t}$ as our target data. Produce for every $\mathbf{x}_i$ in $\mathbf{X}$ its $\Delta \mathbf{x}_{i+1} = \mathbf{x}_{i+1} - \mathbf{x}_i$ and store it in a target matrix $\mathbf{Y}$ (shape $(10000, 3)$). (Δt is not of interest for now.)

d) We now have our input data $\mathbf{X}$, our best guess for what equations could be part of the system's governing equations $\mathbf{R}$ and our target data $\mathbf{Y}$, the thing that these equations in $\mathbf{R}$ are supposed to produce when given an input from $\mathbf{X}$. Now we just have to figure out the connections between them:

$$\mathbf{X} \xrightarrow[\text{function set}]{\text{polynomial}} \mathbf{R} \xrightarrow{\quad ? \quad} \mathbf{Y}$$

The basic idea for how to do this is to find to find a matrix $\mathbf{W}_{out}$ that is optimized to 'predict' the next value of the time series by multiplying the the feature vector of time i $\mathbf{r}_i$ and $\mathbf{W}_{out}$, such that

$$\mathbf{W}_{out}\mathbf{r}_i \simeq \Delta \mathbf{x}_{i+1}. \tag{1}$$

I.e. we are trying to solve

$$\mathbf{W}_{out} = \arg\min_{\mathbf{W}} \sum_i \|\mathbf{W}\mathbf{r}_i - \Delta \mathbf{x}_{i+1}\|. \tag{2}$$

This is solvable using linear algebra, namely by using least squares regression. In practice though it turns out that adding a penalty term to results of regression decreases overfitting substantially. Hence we'll actually solve

$$\mathbf{W}_{out} = \arg\min_{\mathbf{W}} (\sum_i \|\mathbf{W}\mathbf{r}_i - \Delta \mathbf{x}_{i+1}\| + \alpha \|\mathbf{W}\|. \tag{3}$$

where α is the coefficient for the L2 penalty. This type of regression is called ridge regression. $\alpha = 1e{-}8$ happens to work well here. Luckily for us, ridge regression has already been implemented in many different python packages. You can for example use the sklearn library to find $\mathbf{W}_{out}$ with

```
from sklearn.linear_model import Ridge
regressor = Ridge(alpha=1e-8)
regressor.fit(R,Y)
W_out = regressor.coef_
```

Note that the last entry of $\mathbf{R}$ has no target data, hence you'll want to shorten $\mathbf{Y}$ correspondingly.

e) To figure out what equations SINDy has actually reconstructed, divide $\mathbf{W}_{out}$ by Δt (in our case Δt=0.002) and have a look at it. Can you identify matrix elements which look similar to the σ, ρ and β parameters of the initial Lorenz equations? Using this, write down your best guess of the Lorenz equation parameters. Congrats you just found a system's generating equations solely by looking at the data!

f) As this is just an exercise sheet we already know the true governing equation, hence it's easy to see if the parameters we found actually agree with the real ones. In practice, you typically don't know the correct equations, you'd need to use different methods to see if the equations you have found are actually correct. One of the simplest ways to do this for SINDy is to use our results to directly predict the system.
You can do so easily by taking a data point $\mathbf{x}_i$, turning it into a feature vector $\mathbf{r}_i$ and then multiplying it with the optimized $\mathbf{W}_{out}$ to get $\Delta\mathbf{x}_{i+1}$ and from that $\mathbf{x}_{i+1}$

$$\begin{aligned}
\mathbf{x}_i &= (x_i, y_i, z_i), \\
\mathbf{r}_i &= (x_i, y_i, z_i, x_i^2, z_i^2, y_i^2, x_i y_i, x_i z_i, y_i z_i), \\
\Delta\mathbf{x}_{i+1} &= \mathbf{W}_{out}\mathbf{r}_i \\
\mathbf{x}_{i+1} &= \Delta\mathbf{x}_{i+1} + \mathbf{x}_i.
\end{aligned}$$

Implement the above in a for loop for 10000 predictions steps and visually compare it with the real Lorenz system.

Exercise 3.3 Reservoir Computing: TBD