Deep Learning in Practice

Homework 5

In this assignment, the goal is to imitate Rössler attractor dynamic system using Deep Learning, The approach we chose to solve this problem is the following:
We'll build a feedfoward network model that takes as input $w_t = (x_t, y_t, z_t)$ the 3D coordinate of the point at time t , and outputs $w_{t+dt} = f_\theta(w_t)$ the 3D coordinate of the point at time $t + dt$, in this case:

$$f_\theta : \quad \mathbb{R}^3 \to \mathbb{R}^3$$

Once we find the optimal parameter θ, we will compare the trajectories obtained by the true Rössler attractor (RosslerMap.full_traj) with the ones generated by the network, i.e. $(w, f_\theta(w), f_\theta(f_\theta(w)), f_\theta(f_\theta(f_\theta(w))), ...)$.

1 The Approach

1.1 The dataset

To find the appropriate parameter θ of the network, we will sample trajectories using the function *ROSSLER_MAP.full_traj*, we used this function to sample *1,999,800* paires of $(w, f(w)$ (we generated 200 trajectories each one with 10000 points and we picked $dt = 10^{-2}$), we split the data set into training set (90%) and validation set(10%) to evaluate the model and keep the one with the smallest validation loss. then we fed the training set to the network to train the parameters of the network. For the **Batch size**, we used $Batch\_size = Niter - 1$ in order to train at each epoch on a whole trajectory.

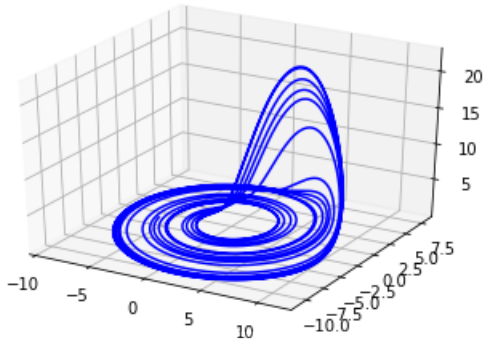


Figure 1: Rössler attractor trajectory

1.2 The Network Architecture

For the architecture of the model, we used a feedforward network with 5 hidden layers, (64,256,256,64,3) respectively the number of neurons in each layer. we apply the *ReLu* activation function between each layer except the last one. The table below presents a summary of the model used:

Layer (type)	Output Shape	Number of Parameters
Linear-1	[1, 64]	256
Linear-2	[1, 256]	16,640
Linear-3	[1, 256]	65,792
Linear-4	[1, 64]	16,448
Linear-4	[1, 3]	195

Table 1: Models Architecture

1.3 Loss function

The loss we used for the training of the network is the **Mean Square Error**, for each sample (w_t, w_{t+dt}) in the training set, we try to minimize the $\mathbf{L}_2$ norm between the predicted position by the network $f_\theta(w_t)$ and the real one w_{t+dt}.
For the optimizer we used **ADAM** algorithm with a learning rate $lr = 10^{-3}$. We trained the model for many epochs (approximately 40 epochs), and we kept the one with the lowest MSE on the validation set. The figure below shows an example of a trajectory generated by the model and the real trajectory:

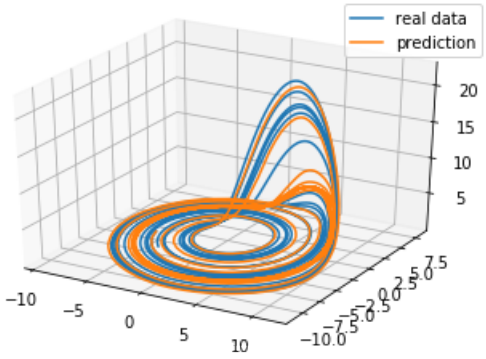


Figure 2: Plot of the predicted and real trajectories of the Rössler attractor

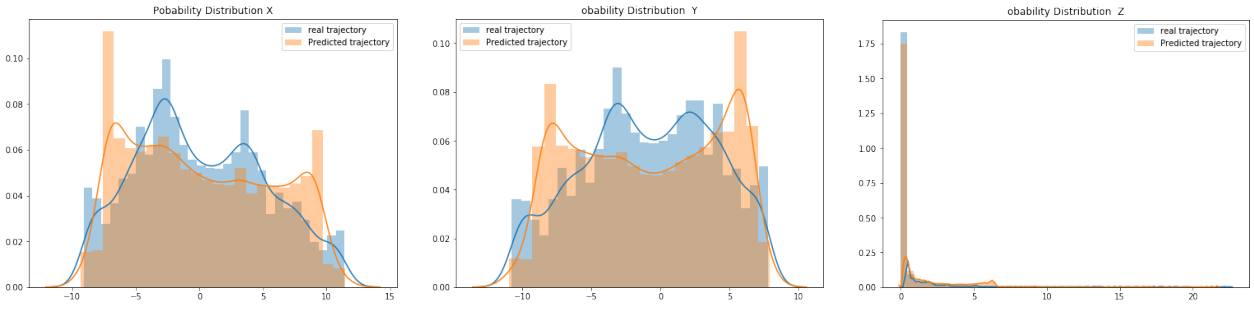
As you can see, the simulated trajectory with the NN is very similar to the real one. which is an indicator that the model did learn to generate trajectories of the Rössler attractor.

2 Analysis of the Results

2.1 Statistical Analysis

2.1.1 Probability distribution

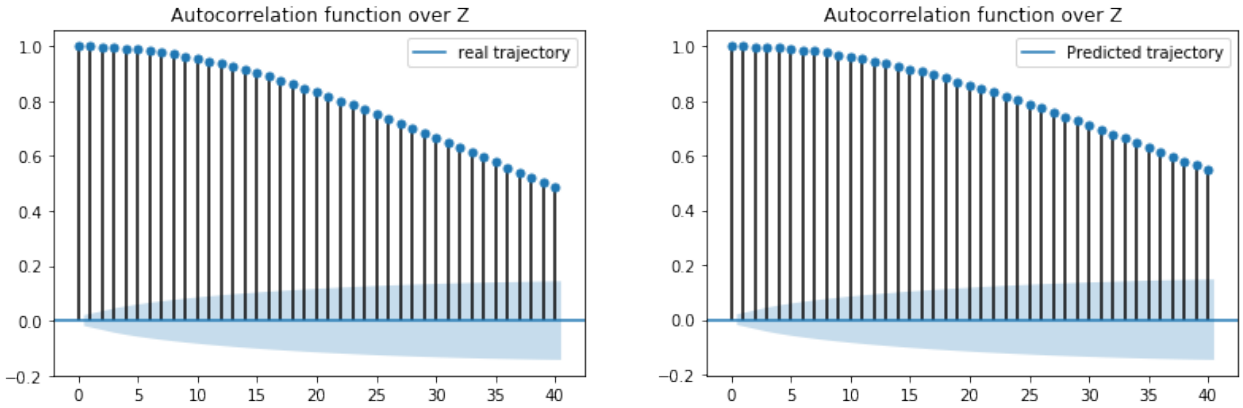
We ploted the estimated PDF over the data of two trajectories generated from the same starting point but one with the NN model, and the other trajectory with the function *ROSSLER_MAP.full_traj*. As you can see on the following plots, the estimated PDFs have slightly the same shape, for example the estimated PDF over the coordinate X has two peaks in both real and simulated trajectory, also the estimated PDFs over the coordinate Z are perfectly similar for the real and the simulated trajectory. Hence the trajectories generated by the model have very similar PDFs to the PDFs of the real trajectories.



2.1.2 Time correlations : the autocorrelation function

We compared the plots of the estimated autocorrelation function over the 3D coordinates of both trajectories (Real trajectory, and the generated one by the NN). For a matter of clarity

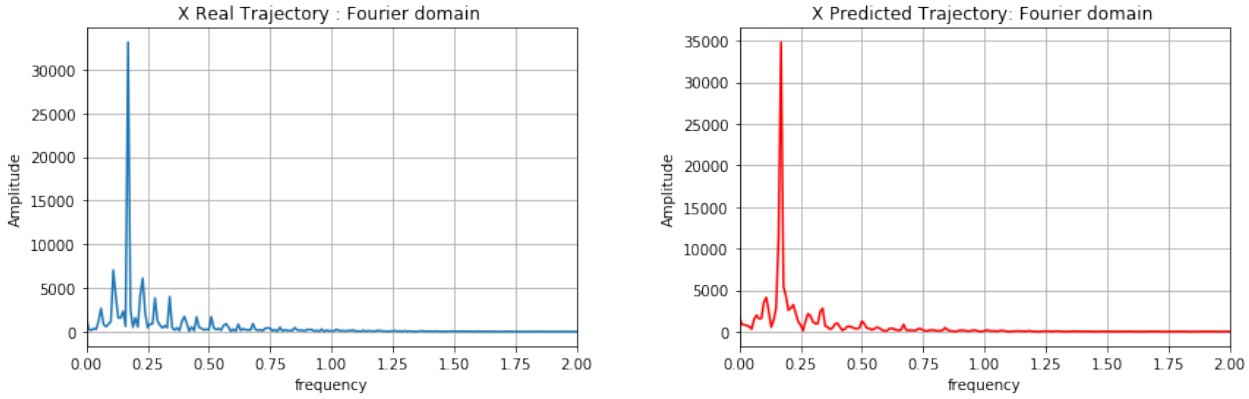
of the plots, we show only the autocorrelation function over Z.



As you can see on the plots, the autocorrelation functions are very similar, which confirms that the model generates trajectories very similar to reality.

2.1.3 Fourier Coefficients

We moved to the Fourier domain, to get information about frequencies . The plots below represent the coefficient of the Fourier transformation over the coordinate X for the real and the predicted trajectories, as you can see, the frequencies observed are similar for both trajectories.



2.2 Physical Analysis

2.2.1 Equilibrium point

We tried to use the newton method to find the equilibrium point, but this method didn't converge and we didn't get the value of the equilibrium point, which proves one limitation of the model, one way to justify this limitation is that all the trajectories used for the training

are generated from a starting point *INIT* very far from the equilibrium point, and none of the point of the trajectories got close to the equilibrium, so it's reasonable that the Newton method didn't converge.

2.2.2 Lyapunov exponent

We used the function **lyapunov_exponent** to find the Lyapunov exponents of the predicted trajectory, the values we got for these coefficients are $(0.102, -0.049, -5.371)$, this is a good result because we saw in the course that the biggest Lyapunov exponent has to be $\lambda \approx 0.07$.

To Sum up, The model we built generates trajectories very similar to real ones. As we saw on the plots, the trajectories had similar shape, and the statistical and physical analysis confirm the correctness of the generated trajectories, one limitation remains for the model is the **Equilibrium point**, one way to handle this limitation is to add in the training set trajectories that start close to the equilibrium point, so the model learns how to generate this kind of trajectories.