DL in Practice: Practical Session 5 Report

Jérémy Rousseau, Alassane Watt

March, 2021

1 Introduction

In this assignment, the goal is to imitate the Rössler Attractor dynamic system using Deep Learning. Our approach to solving this problem is explained as follows:
We build a feed-foward network model that takes in as input $w_t = (x_t, y_t, z_t)$, the 3D coordinate of a point at time t , and outputs $w_{t+dt} = f(w_t)$, the 3D coordinate of the next-step point at time $t + dt$, where $f : R^3 -> R^3$. Once we find the optimal parameter through neural network training, we compare the trajectories obtained by the ground-truth Rössler attractor trajectory (RosslerMap.full_traj) with the ones generated by the network: $(w, f(w), f(f(w)), f(f(f(w))), ...)$.

2 Approach

2.1 Dataset

To find the appropriate parameter of the network, we sample trajectories using the function ROSSLER_MAP.full_traj. We used this function to sample 1,999,800 pairs of $(w, f(w)$ (we generated 200 trajectories each one with 10000 points and we picked $dt = 10e2$). We then split the dataset into a training set (90%) and a validation set(10%) to evaluate the model and keep the one with the smallest validation loss. Then we feed the training set to the network to train the parameters of the network. We set $Batch\_size = Niter1$ in order to train at each epoch on a whole trajectory.

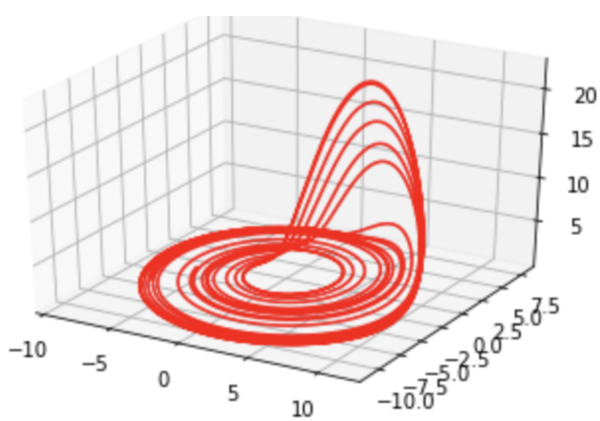


Figure 1: A Rössler-attractor trajectory.

2.2 Architecture

For the architecture of the model, we used a feed-forward network with 5 hidden layers with numbers of neurons (64,256,256,64,3) respectively 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         Param #
================================================================
            Linear-1                [-1, 1, 64]             256
            Linear-2               [-1, 1, 256]          16,640
            Linear-3               [-1, 1, 256]          65,792
            Linear-4                [-1, 1, 64]          16,448
            Linear-5                 [-1, 1, 3]             195
================================================================
Total params: 99,331
Trainable params: 99,331
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.38
Estimated Total Size (MB): 0.38
----------------------------------------------------------------
```

Figure 2: Model Summary

2.3 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 L2 norm between the predicted position by the network $f(w_t)$ and the real one w_{t+dt}. For the optimizer, we used the ADAM algorithm with a learning rate $lr = 103$. We trained the model for many epochs (around 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:

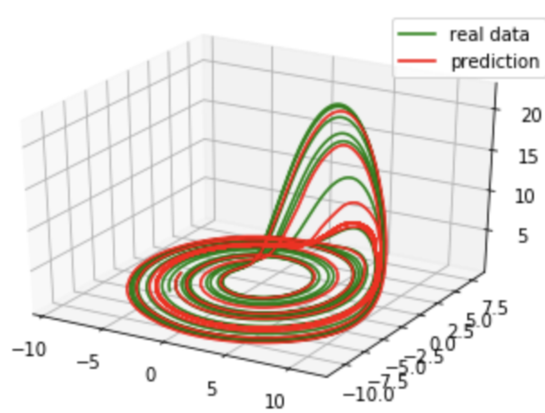


Figure 3: An example of predicted trajectory

As one 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 quite well.

3 Results Analysis

3.1 Statistical Analysis

3.1.1 Probability distribution

We plotted the estimated PDF over the data for 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.

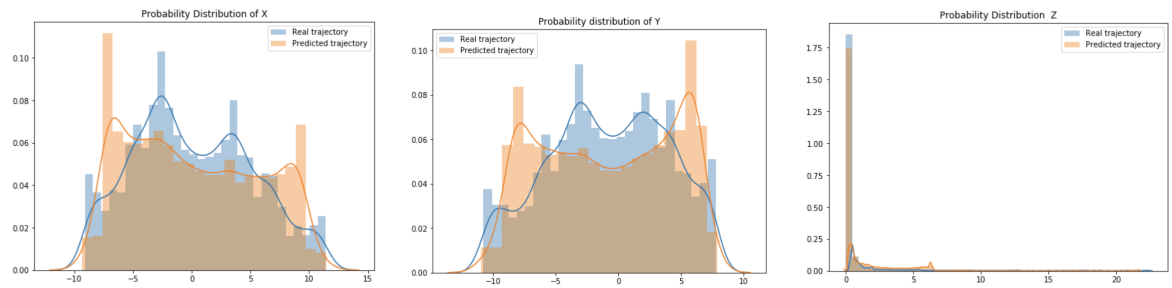


Figure 4: Probability distributions

3.1.2 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.

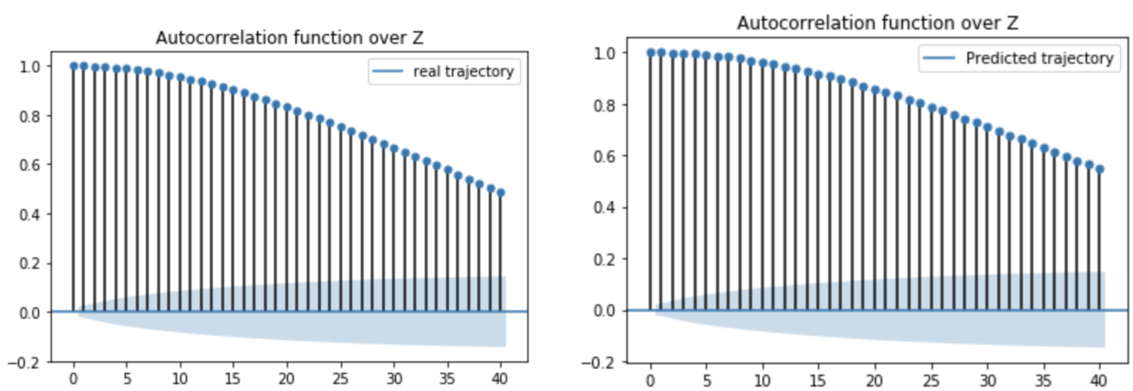
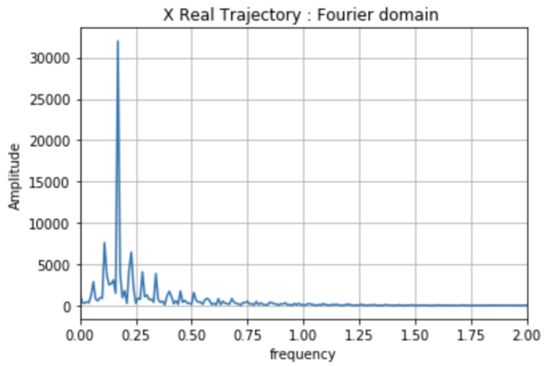


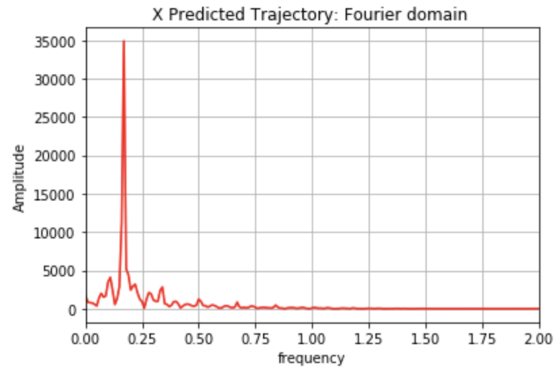
Figure 5: Time correlations

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

3.1.3 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.





4 Physical Analysis

Add your algorithm implementation and development here. See Algorithm **??** for how to include an algorithm in your document. This is how to make an *ordered* list:

5 Computational Results

5.1 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 points of the trajectories got close to the equilibrium, so it's reasonable that the Newton method didn't converge.

5.2 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 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 a similar shape, and the statistical and physical analysis confirms 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.