Politecnico di Milano (2022)

PROJECT REPORT

Homework of the course "Modelling From Measurements"

Luca Radicioni*

Department of Mechanical engineering, Politecnico di Milano, Milano, Italy
*Corresponding author. Email: luca.radicioni@polimi.it

Abstract

This notes contain the homework's solutions related to the course of "Modelling From measurements", held by professor Nathan Kutz at Politecnico di Milano. The course focuses on data-driven techniques to derive the equations of dynamical systems, directly from measurements. Generally speaking, the information coming from sensors is not always organized in the best possible arrangement. Algorithms such as SVM or Autoencoders can be helpful to map the measurements into a reduced space, where less dimensions retain most of the knowledge. The equations of the dynamical system can be derived also in this embedding. To do so, some techniques as *Dynamic Mode Decomposition* (DMD) and *Sparse identification of non-linear dynamics* (SINDy) have been considered. The enitre code available open source at: https://github.com/lucarad96/ModellingFromMeasurement_homework.

Keywords: Singluar value decomposition; Dynamic mode decomposition; Dynamical models; Autoencoders; Model discovery

1. Introduction and overview

The importance of building models for dynamical systems, i.e systems characterized by interactions among quantities that *co-evolves* in time (Brunton and Kutz 2022), dwells in the possibility to increase the overall knowledge over a problem, with many benefits. Thanks to modelling is possible to achieve insightful *interpretations* on variables' dependencies, *predict* future states of the system, *classify* different conditions and ultimately *control* the real system itself.

Humans have always derived the governing equations for dynamical systems by "hand", through observation of some physical quantities, inferring assumptions and testing them by means of experiments. However, in recent times, the *automatic discovery of models* from datasets became attractive mainly due to a few factors. First of all, the availability of cheap and robust sensors that can provide an enormous amount of information in real time, which was unthinkable only few decades ago. Secondly, the incredible computational powers of modern machines, that allows to run complex and powerful algorithms also on relatively economical personal computers. Finally, the rising of *open-source softwares* made possible to have easy access to updated algorithms, that quickly evolve in performances and complexity thanks to the free contribution of programmers from all over the word.

Every engineer working with *machine learning* knows how easily has become the building of effective models directly from data. For instance, neural networks in Google's TensorFlow can

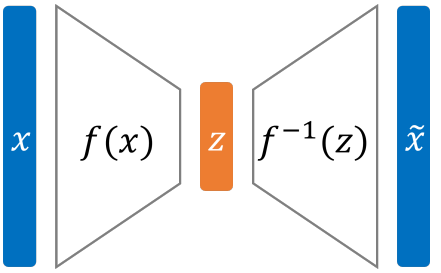

Figure 1. If there exists a function $f(x)$ able to map the original measurement space x into a reduced space z, it means that the function $\tilde{x} = f^{-1}(z)$ must be able to minimize the reconstruction error $e = \tilde{x} - x$.

be built in few lines of code with Keras API. Unfortunately, dynamic systems evolve in time and what has been collected in the training set might rapidly change in future observations. Quite often machine learning is inadequate to address engineering problems. After all, what machine learning does is a simple *interpolation* of datapoints. However, what is in most of the cases needed in practice, are models able to *extrapolate* for different and unseen conditions. Hence, the necessity for techniques able to derive from a dataset the *governing equations*.
Since sensors are rarely placed in an optimal arrangement, it results commonly convenient to operate a coordinate transformation before fitting whichever model to the data (figure 1). The main benefit in doing so is that generally the system features less but meaningful dimensions in the embedded space, and therefore the parameters required to be tuned might be tens (o even hundreds) times less than the ones required in the original space. It's also worth mentioning that in some cases the embedded space Z might have more dimensions than the sensor space X. In fact, it is always possible to increase the number of system's variable up to a point where the problem appears linear. This is what happens for example in SVM wtih the *kernel trick*.

2. Theoretical background

In this section, the mathematical background behind the main techniques adopted in the homework's solution are briefly introduced.

2.1 Singular Value Decomposition

Consider the data matrix X as shown in 1. It represents the values acquired from a measurement system at different snapshots of time. Evidently the arrangement of this data matrix depends on the choice of sensors and their position, which means that the measured values are likely to display a certain degree of correlation to each other.

$$X = \begin{bmatrix} x(t_1) & x(t_2) & \cdots & x(t_m) \end{bmatrix}. \tag{1}$$

A good idea could be to transform the data from the original space into a new set of coordinates orthogonal to each others. This is exactly what PCA (Principal Components Analysis) does. Eventually, it is possible to achieve dimensionality reduction by retaining only the leading r principal components.
An efficient algorithm able to retrieve those principal components takes the name of *Singular Value Decomposition* (SVD). SVD assures that a complex matrix $X \in \mathcal{C}^{n \times m}$ can be uniquely decomposed into three matrices, as shown in equation 2, where $U \in \mathcal{C}^{n \times n}$ and $V \in \mathcal{C}^{m \times m}$ are unitary matrices whose columns are orthonormal, whereas $\Sigma \in \mathcal{R}^{n \times m}$ features non-negative values on the diagonal and all zeros in the off-diagonal therms.

$$X = U \Sigma V^*. \tag{2}$$

Among other interesting properties, SVD is used to compute the eigenvalues (which corresponds to the square of the diagonal therms of σ) and the eigenvectors (Columns of V) of the matrix X. According Eckart-Young theorem SVD gives an optimal approximation of X by truncating the matrices at a rank r. The output of this operation is the approximated data matrix $\tilde{X}$. The equation $\tilde{X} = \tilde{U}\tilde{\Sigma}\tilde{V}^*$ is usually denoted as *truncated SVD*. In this way dimensionality reduction can be achieved with a minimal loss of information.

2.2 Dynamic Mode Decomposition

Once the system's dimension have been reduced, it could be convenient to build a model directly into this embedded system Z. Dynamical mode decomposition (DMD) provides a modal decomposition where each mode consists of spatially correlated structures that have the same linear behaviour in time (Brunton and Kutz 2022). Each DMD mode is associated with a particular eigenvalue $\lambda = a + ib$. The *exact DMD* algorithm can be obtained by combining SVD and the searching for the best linear operator A that allows to get the next snapshot of time x_{t+1} for a given snapshot x_t, as shown in equation 3.

$$x_{k+1} \approx Ax_k. \tag{3}$$

The exact DMD algorithm counts the following steps:

1. X is decomposed by means of truncated SVD:

$$X \approx \tilde{U}\tilde{\Sigma}\tilde{V}^*. \tag{4}$$

2. The *reduced-order matrix* $\tilde{A}$ through:

$$\tilde{A} = \tilde{U}^* X' \tilde{V}\tilde{\Sigma}^{-1}. \tag{5}$$

3. The *spectral decomposition* of $\tilde{A}$ is carried out. The columns of W are the eigenvector of $\tilde{A}$, the elements of the diagonal of Λ are the DMD eigenvalues:

$$\tilde{A}W = W\Lambda. \tag{6}$$

4. The *high-dimension DMD modes* Φ are reconstructed by:

$$\Phi = X'\tilde{V}\tilde{\Sigma}^{-1}W. \tag{7}$$

DMD allows to predict the future state of the system. The DMD *spectral decomposition* is:

$$x_k = \sum_{j=1}^{r} \Phi_j \, \lambda_j^{k-1} b_j \tag{8}$$

where $b = \Phi^{-1}x_1$ is the vector of mode amplitudes.The spectral expantion in equation 8 might also be written in continuous-time form by using the continuous eigenvalues $\omega = \log \frac{\lambda}{\Delta t}$:

$$x(t) = \sum_{j=1}^{r} \Phi_j \, e^{\omega_j t} b_j = \Phi \exp(\Omega t) b \tag{9}$$

Other versions of DMD do exists; for instance, in *optimal DMD*, the search for the set of DMD-parameters is approached as an optimization problem:

$$\arg\min_{b_j, \psi_j, \lambda_j} \|x - \sum_{j=1}^{r} b_j \, \psi_j e^{\lambda_j t}\| \tag{10}$$

2.3 Time-delay embedding

An attractive prospective for the estimation of data-driven models could rely on the intrinsic measurement coordinates for Koopman based on *time-delayed* measurements of the system. Instead of a direct advance (with linear or nonlinear measurements of the state of a system) as in DMD, it is possible to obtain the eigen-time-delay coordinates from a time-series of a single measurement $x(t)$ by taking the SVD of the Hankel matrix H (Brunton and Kutz 2022):

$$H = \begin{bmatrix} x(t_1) & x(t_2) & \cdots & x(t_{m_c}) \\ x(t_2) & x(t_3) & \cdots & x(t_{m_{c+1}}) \\ \vdots & \vdots & \ddots & \vdots \\ x(t_o) & x(t_{o+1}) & \cdots & x(t_m) \end{bmatrix} = U\Sigma V^*. \tag{11}$$

2.4 Neural Networks

Artificial Neural Networks (ANN) are deep leaning models that elaborate an input object, by transforming it through successive layers. ANN have been loosely inspired by the studies over the central nervous system of mammals. In their easiest representation, a feed-forward artificial neural network counts an input layer, a few hidden layers and an output layer. The neurons in the input layer propagate the input values to the neurons in the successive layer, by means of some connections characterized by weights. The weighted sums are propagated into the following layer, and so on until the values are ultimately sent to the output layer (Bono et al. 2022). The training of a neural network can be thought as an optimization over a compositional function:

$$\arg\min_{A_j} \left(f_M(A_M, \cdots, f_2(A_2, f_1(A_1, x)) \cdots) + \lambda g(A_j)\right) \tag{12}$$

An interesting structure which can be realized with an opportune neural network takes the name of "Autoencoder". An autoencoder has a structure similar to the one in figure 1; it is trained to reconstruct the input object and it displays many attractive properties, including dimensionality reduction by means of non-linear transformations.

2.5 Sparse identification of nonlinear dynamics

Typically, the form of a model that might be able to represent a non-linear dynamical system is established a priori, based on previous knowledge of the governing equations. The sparse identification of nonlinear dynamics (SINDy) algorithm tries to assess the form of the function f of the system 13 by approximating it with a generalized linear model, which has the fewest possible non-zero terms ("sparsity") in ξ.

$$\frac{d}{dt}x = f(x) \approx \sum_{k=1}^{p} \theta_k(x)\xi_x = \Theta(X)\Xi \tag{13}$$

In equation 13, $\Theta(X)$ is a *library* of nonlinear cadidate built from the data X:

$$\Theta(x) = \begin{bmatrix} 1 & X & X^2 & \cdots & sin(X) & cos(X) & \cdots \end{bmatrix} \tag{14}$$

The result of the SINDy regression is a parsimonious model that includes only the most important terms required to explain the observed behavior (Brunton and Kutz 2022).

2.6 Bagging

In machine learning, "Bagging" (bootstrap aggregating) is a procedure which allows to improve the performace of a classifier or a regressor thanks to a form of *ensemble* learning. It consists in dividing the set of data into an N number of subsets, which are used to train separately N models. Although it has been firstly proposed for the ensemble of decision trees, it can be applied to other system identification techniques, as DMD, SINDy or NN. It has been proved that, by using bagging, the overall robustness in prediction improves (Sashidhar and Kutz 2021).

3. Algorithm implementations and results

In this section, the methods adopted to solve the homework's exercises are reported. Exercises 1.1, 1.2, 1.3 are developed in a MATLAB environment, whereas all the others are carried out in Python. For the handling of neural networks the package used is TensorFlow.

3.1 Hare and Lynx

In the first four exercises of the homework, we deal with the modelling of a predator and prey dataset. From the collection of pelts of those years, it has been possible to estimate the population of *hare* and *lynx* between 1845 and 1903. The original dataset counts only 30 snapshots (a sample every 2 years, represented with dots in figure 2). To increase the number of points and consequently reduce the discontinuities among different snapshots, *cubic splines* are fitted onto data and the resolution lifted to 0.5 years (continuous lines in figure 2). We have now a total of 117 snapshots in the same time span.

3.1.1 DMD models

First of all, it must be noticed that DMD models may have only 2 eigenvalues, due to the fact that only 2 timeseries are available. If we apply exact DMD to the data matrix X, and we try to make predictions for the same initial conditions (by means of equation 9), we obtain as a result the asterisk-lines in figure 3. One should notice that this model results inadequate, since both the curves go below zero and then settle. With optimal DMD we get the dotted lines in figure 3. The situation has slightly improved (at least there are now positive values only), but the model seems not able to capture the complexity of the system.

The situation slightly improves with bagging. 200 subsets containing 30 sampling each ($\approx \frac{1}{4}$ of the complete dataset) are randomly generated, and used to generate 200 models separately. Eventually, predictions are made for every model, and then the mean and the standard deviation (single confidence

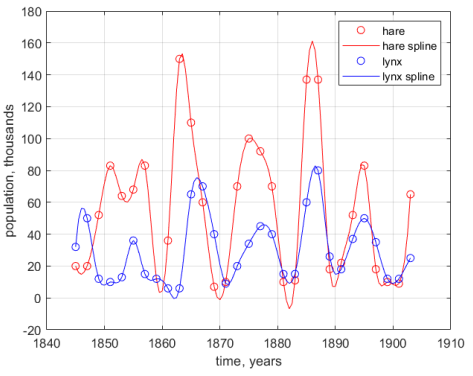

Figure 2. Estimated population of hares and lynxes from 1845 to 1903. Original dataset and splines-fitting

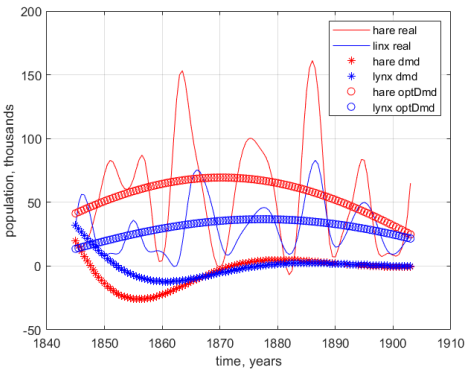

Figure 3. Real population of hare and lynx VS previsions from exact DMD VS previsions from optimal DMD

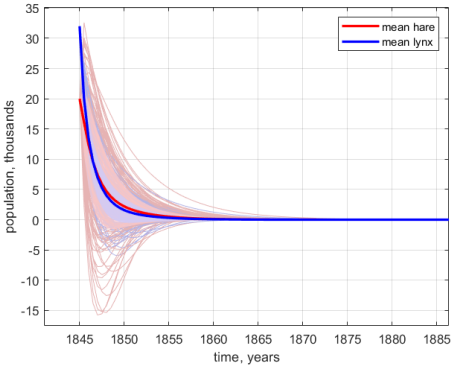

Figure 4. Predictions made by means of bagging and exact DMD. All the curves rapidly converge to zero.

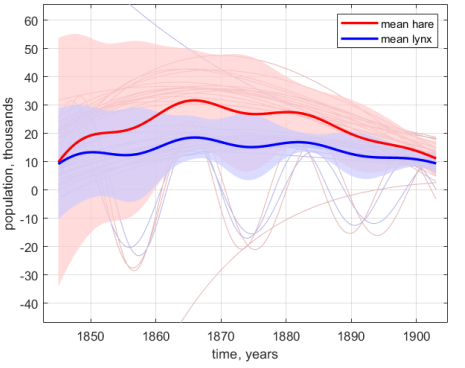

Figure 5. Predictions made by means of bagging and optimal DMD. Highlight on the overall mean and standard deviation area.

interval) of the results are plotted. The outcome are reported in figure 4 for the exact DMD, and in figure 5 for the optimal DMD. For both the methods the unstable models have been removed. The exact DMD has only slightly improved with bagging (at least now the values are non-negative), whereas they are definitely better now for the optimal DMD.

3.1.2 DMD with time-delay embedding

The Hankel matrix is built according the structure in equation 11, where $m_c = 90$. We can now increase the number of DMD modes r. In figure 6 and 7 we might see how the predictions behaved in exact DMD and optimal DMD, respectively, for different dimensions of the latent space. Evidently, a $r = 5$ is sufficient to build a model that captures the overall behaviour of the system for both optimal and exact DMD. We can affirm then that latent variables do exist for this problem.

When bagging is adopted, one may notice that the performances of the single models rapidly decrease with the reduction of the numer of samples per subset. Therefore, an high number of samples (80 per subset) are considered. 200 models are trained with $r = 8$, obtaining in prediction the values in figure 8 for the exact DMD and 9 for optimal DMD. However, in this problem, the result seems not justify the adoption of bagging. The simple time-delay DMD behaved better than DMD

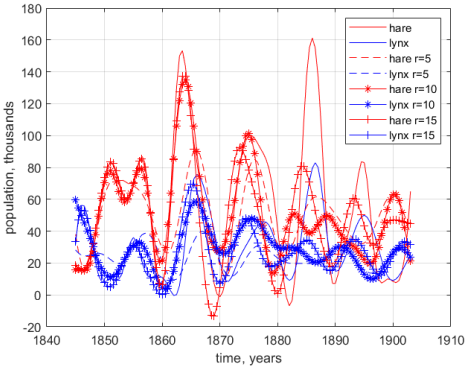

Figure 6. Time-delay exact DMD. Different dimensions of the latent space yielded fairly similar results.

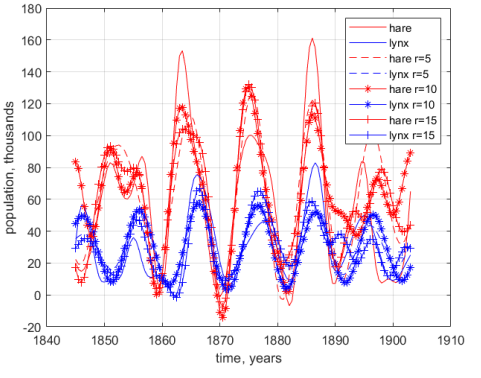

Figure 7. Time-delay optimal DMD. Results are quite close for different values of r.

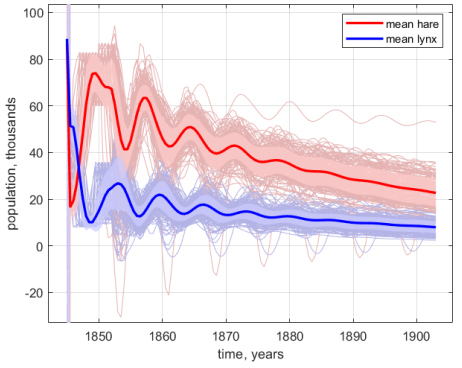

Figure 8. Time-delay, exact DMD with bagging. Dispersion extremely high in the first years. Then the curves converge.

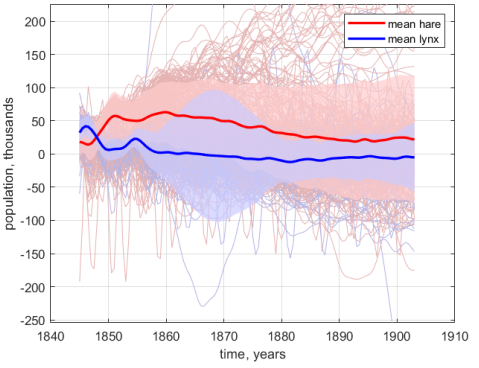

Figure 9. Time-delay, optimal DMD with bagging. Dispersion rises in time. Some unstable models might be present.

with bagging!

3.1.3 Lotka-Volterra empirical model

The Lotka-Volterra model consists of two first-order nonlinear differential equations: $\dot{x} = (b - py)x$; $\dot{y} = (rx - d)y$. It has been historically proposed for the modelling of the Predator-Prey problem (Berryman 1992). Having the two equations, we must define the 4 best parameters (b, p, r, d) that fit the hare and lynx dataset.

At a first glance, we can think at this situation as an optimization problem, where the objective function is the sum of the differences between the expected output and the current output for every time step (finite differences method). In MATLAB the function *fminunc* is used with this purpose, alongside a custom function which iteratively calculates and sum the absolute distance between the system output and the real difference at two successive time steps:

$$\arg\min_{b,p,r,d} \sum_{i=1}^{N-1} |F(b,p,r,d,X_i) - (X_{i+1} - X_i)| \tag{15}$$

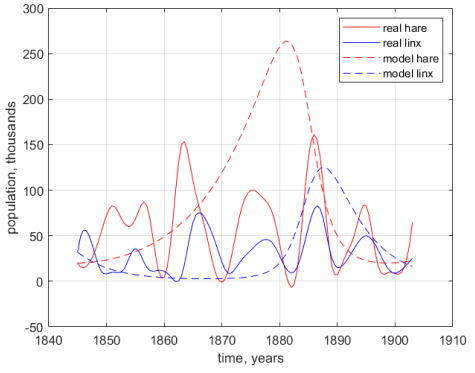

Figure 10. Predictions made by Lotka-Volterra equations, with parameters found by means of finite difference method (0.0986, 0.0031, 0.0023, 0.2137).

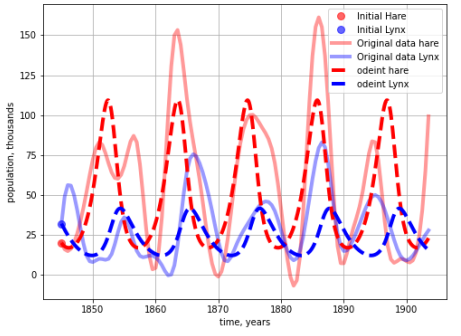

Figure 11. Lotka-Volterra parameters (0.8871, 0.0371, 0.008, 0.3977), obtained by optimizing the parameters and an ODE integrator to define the loss function.

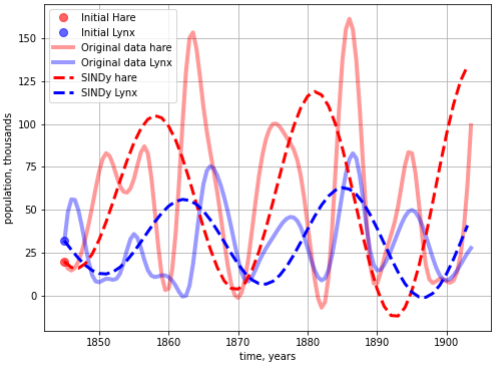

Figure 12. Real data and predictions made from a single SINDy model.

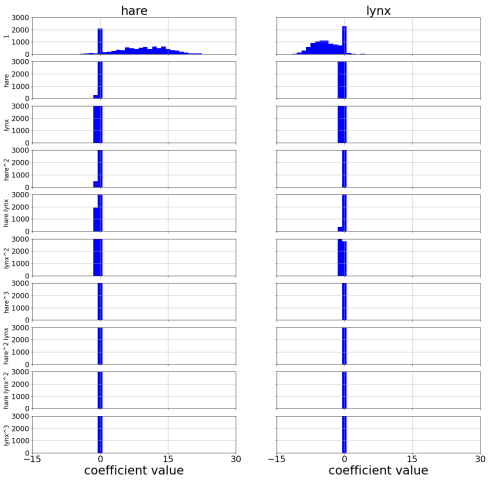

Figure 13. Histogram plots containing the distribution of coefficients obtained with bagging on both library and samples.

where F represents the Lotka-Volterra equations depending on the four parameters b, p, r, d to be optimized, X_i is the vector containing the values for hare and lynx at the i^{th} snapshot and N the total number of snapshots. One hundred random initial points are used to generate the same number of optimization sessions. Eventually, the best model produced the curve in figure 10. The model found with this procedure is obviously far from achieving a good understanding of the problem.

For this reasons, another optimization (scipy.optimize.fmin) is carried out and the error is computed by integrating the initial conditions (scipy.integrate.odeint) at every optimization iteration. The results are reported in figure 11. It must be said that the result is extremely sensible to the initial point.

3.1.4 Sparse Regression (SINDy)

In the previous section, we imposed as a boundary condition the use of a Lotka-Volterra model. But imagine if one wants to find the differential equations with no prior knowledge over a certain phenomenon. This is exactly what SINDy has been designed for.

In a first analysis, we train a single SINDy model. The polynomial library is set to fifth order, whereas the minimum value for a single parameter is set to 0.01. The model identified by the optimizer (Sequentially Thresholded Least Squares algorithm) is the one in equation 16, which produced the graph in figure 12.

$$\begin{cases} \dot{hare} = 13.162 + 0.148hare - 0.656lynx \\ \dot{lynx} = -4.07 + 0.143hare - 0.125lynx \end{cases} \tag{16}$$

Notice that the system achieved in this way is linear. Unfortunately, some values for hares are negative, while in a certain moment lynxes go to zero (which is quite unrealistic...).

SINDy Python package support also *bagging* for both dataset and library. So 100 models are trained, by using both bagging on samples (90) and library (2 therms randomly dropped for every model) which means a total of 10 thousand models. The threshold for every entry is set really low (0.001) to promote the building of models with the most possible number of parameters. The histograms containing the distribution of coefficients are plotted in figure 13.

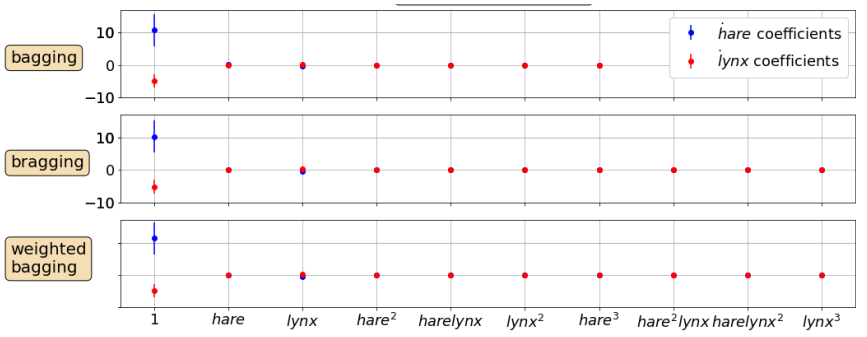

Figure 14. Distributions of parameters' values and highlight on means, medians and weighted means.

Since while trying to make predictions on the *double ensembling* we got a matrix multiplication error, we drop library bagging and focus on data bagging only. We trained 200 models with samples bragging; moreover, an integration for some time snapshots is carried out so as to cut off all the unstable models and the models which yield negative values. In figure 14, a comparison of bagging (mean of the parameters), bragging (median of the parameters) and weighted bragging (where the weights are defined according the mean squared error to the train set) is shown. Eventually, predictions are made for every model, and the results are plotted in figure 15. Compared to figure 12, the means (and medians) of the predictions assume reasonable values now.

3.2 Partial differential equations and neural networks

In this subsections, we are going to see how it is possible to train neural networks to make predictions in systems regulated by nonlinear partial differential equations, to evaluate their performances and limits.

3.2.1 Kuramoto-Sivashinsky equations

Kuramoto–Sivashinsky equation corresponds to a fourth-order nonlinear partial differential equation, originally proposed for the modelling of diffusive-thermal instabilities in a laminar front (Kudryashov 1990). In Python a random initial condition is integrated to compute the future states of the KS model, and then a neural network, whose structure is made of Dense layer(1024, 800, 800, 800, 1024) is trained on the dataset to predict the next state, given the previous one only. In figure 16, we see the original data generate from random initial conditions (left), the output of the neural network trained on that dataset for the same initial conditions (center) and the absolute error between the two (right). The predictions of the network resemble the original data, but from the error it is clear

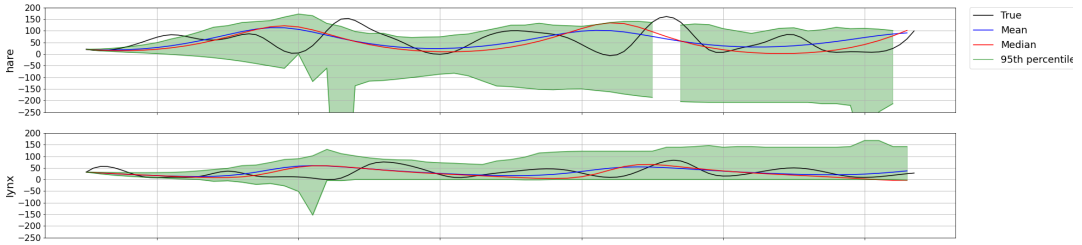

Figure 15. Overall predictions with the stable models, trained on 200 subsets of 80 samples each. Highlight on mean, median and 5-95% percentile range.

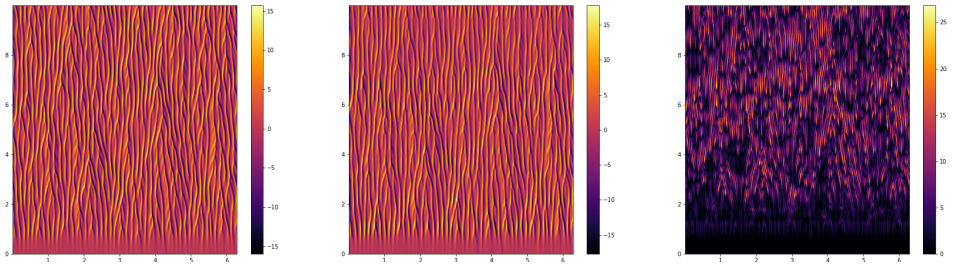

Figure 16. "Training data". Original data (left), predictions of the neural network with the same initial conditions (center) and absolute error plot (right).

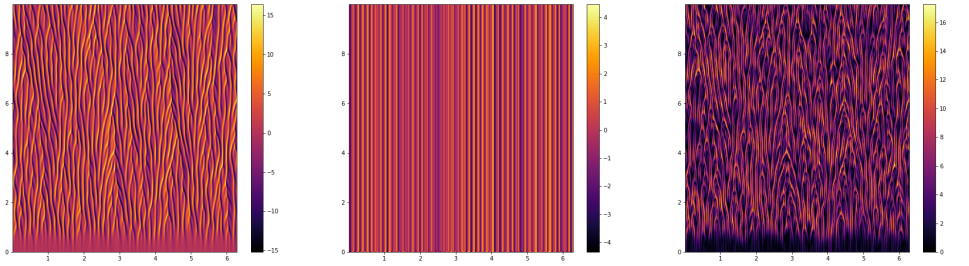

Figure 17. Exact results (left), predictions of the neural network (trained on another dataset, center) on new initial conditions and absolute error (right).

that differences exists. In figure 17, the same network is evaluated on different (but still random) initial conditions. The network results totally unable to manage this slightly different problem. The error is massive.

3.2.2 Reaction-diffusion system

The training of a neural network which simulates the behaviour of a system characterized by high dimensionality can be relatively troublesome. Consider for example the reaction-diffusion equations 17 applied to a system consisting of spiral periodic waves:

$$\begin{cases} u_t = 0.1\nabla^2 u + \lambda(A)u - \omega(A)v \\ v_t = 0.1\nabla^2 u + \lambda(A)u - \omega(A)v \\ A^2 = u^2 + v^2, \omega(A) = -\beta A^2, \lambda(A) = 1 - A^2 \end{cases} \tag{17}$$

If we want to train a neural network to predict the next state of u and v, being them large objects, we may end up in the training of an enormous network, characterized by million of parameters. But let's consider the following procedure:

1. We begin with a dataset composed by two main arrays: **u** (512, 512, 201 timesteps) and **v** (512, 512, 201 timesteps). We reshape them into two arrays of $(512^2, 201)$.
2. The two arrays are concatenated into a single array **uv** of shape: $(2 \cdot 512^2, 201)$. The SVD is applied. In this situation $r = 12$ is chosen, since the first twelve singular values account for more than 99.9% of the variance.
3. The array **uv** is projected into the reduced space. Then a neural network made of Dense layers (12, 10, 6, 10, 12) is trained to compute for every timestep the successive one.
4. Starting from the same initial timestep, the neural network generates recusively the successive 200 timesteps.

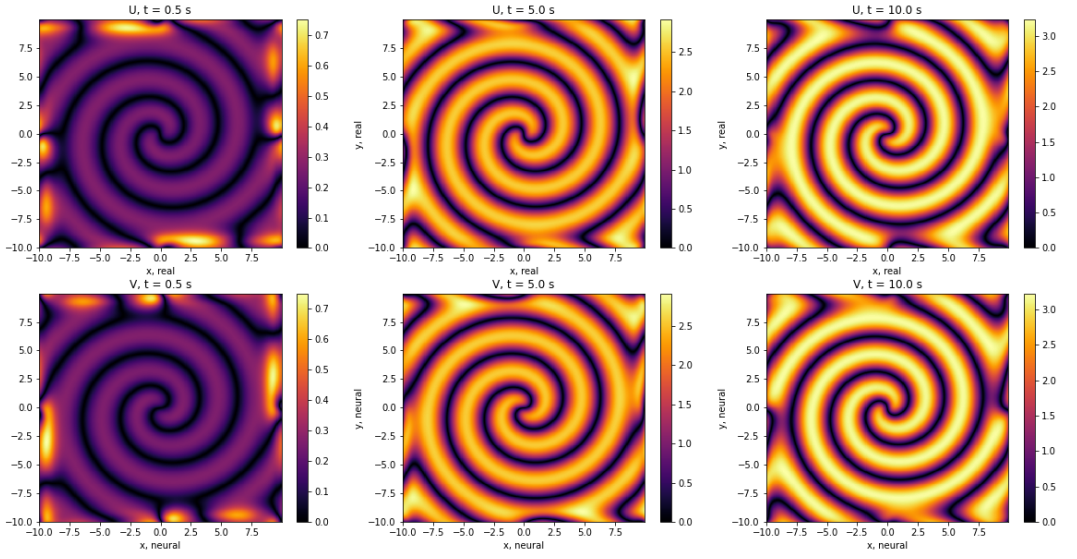

Figure 18. Absolute error in the reconstruction of u, v, for the same initial conditions. The more the time elapsed, the more the mistakes made by the neural network in prediction sum up.

5. The elements generated are scaled back into the original space, **u** and **v** are separated and reshaped to the original dimensions (512, 512).

Finally we can compare the outputs of the network with the original dataset, as in figure 18. The real and the neural u and v are compared at three different times, by taking the absolute error of the real arrays and the respective one computed by the network. Evidently, the error increases with the elapsing time.

We have just shown that, training a model in the reduced space, obtained by means of SVD, might extremely advantageous. The same element passes from being 2 objects of size 512×512 to be a single 12-dimensional vector, and the training of the model can be thousands of times faster!

3.3 Neural network for Lorenz equations

The "Lorentz equations" 18 are a set of ordinary differential equations system, famous for having chaotic solutions for certain sets of parameters and initial conditions.

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(\rho - z) - y \\ \dot{z} = xy - \beta z \end{cases} \tag{18}$$

Here we evaluate the performance of a neural network trained for certain values of rho with certain initial conditions, in making predictions on other values or rho with another set of initial conditions. In figure 19 trajectories generated by the integrator of Lorenz equations, for 3 values of rho and a set of 100 starting points. Those curves are then used to train a neural network (4, 10, 10, 10, 3), which takes as input the 3 coordinates of a point and the rho's value, and then tries to predict the successive point of the trajectory.

Finally, this neural network is tested onto a new set of random initial points, with a $\rho = 17$ (which falls within the range of the previous training set, making this problem an *interpolation*) and a *rho* = 40, which is evidently an *extrapolation* problem. The results are visually shown in figure 20.

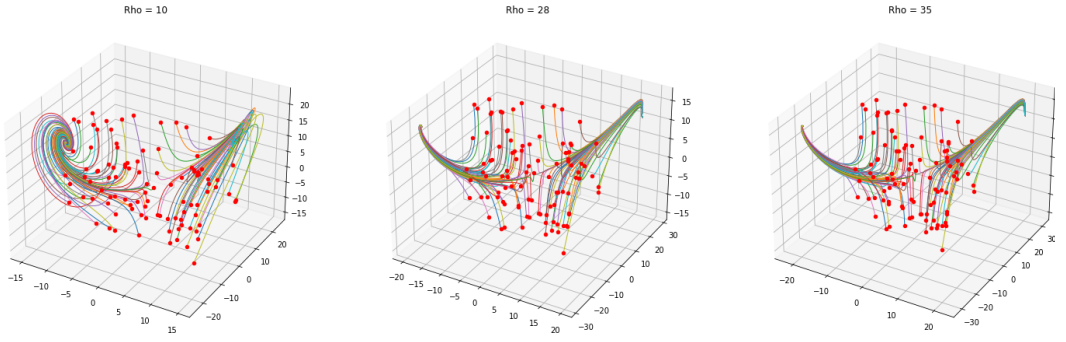

Figure 19. Lorenz equations. 3 different rhos (10, 28, 35) and 100 starting points.

When we evaluate the overall euclidean distance between the correct trajectories and the predicted trajectories, we end up noticing that the overall distance with *rho* = 40 is times greater than with *rho* = 17! This means that, for a neural network, interpolate among the training data is reasonably easier than extrapolate for different conditions.

4. Summary and conclusions

In this report, we have briefly proposed the theoretical background and discussed the solutions of some problems related to the data-driven estimation of physical laws. We have introduced the importance of extrapolating rather than carry out a simple interpolation (section 1), and then we have seen how it is possible to apply dynamic mode decomposition (with and without bagging,

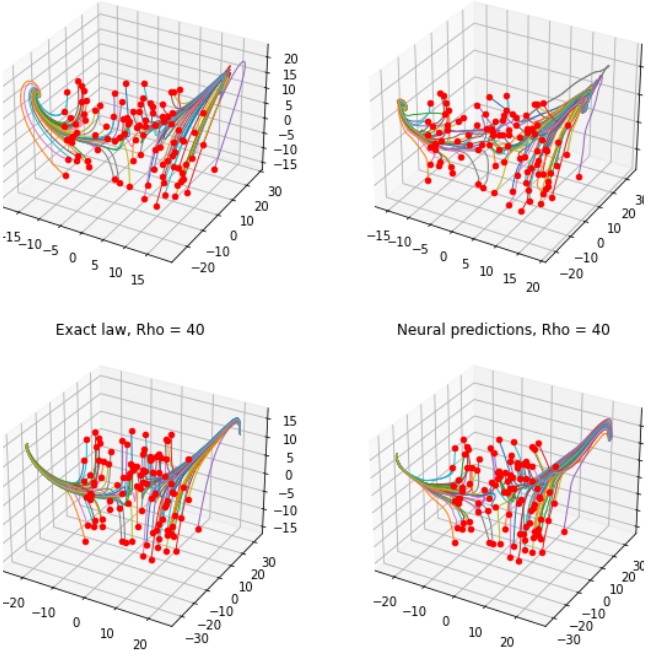

Figure 20. Exact laws and trajectories defined by the neural network.

section 3.1.1), also alongside time-delayed embedding (section 3.1.2). Then we have estimated the parameters of a system, in the case a model is given (section 3.1.3) and in the case there is no prior knowledge over the model itself (section 3.1.4). In sections 3.2.1 and 3.3 we have experience how neural networks fail in predict the state of systems when the conditions are just slightly different from the ones in the training set. We have also seen how advantageous could be the training of a machine learning model in the reduced space generated with SVD, instead of training it in the original space (section 3.2.2).

Acknowledgement

I would like to briefly thank professor Kutz for the lectures he has held for us in June 2022. The research topics of professor Kutz are fascinating indeed, and so is the way he teaches. I will always remember one of his thoughts related to the importance of extrapolation: "... eventually, we ended up reaching the moon, but we couldn't solve e the cancer problem, not even today. This happened because we knew who was the enemy when we wanted to reach the space, namely the gravity g, and we had then some models to work with. Besides, many mechanisms behind biological systems are still unknown. It's 2022. I think, as humanity, it is high-time to do something more for that...". Of course, I must thank also professor Berizzi for having arranged the course and for having hosted professor Kutz in Milano.

References

Berryman, Alan A. 1992. The orgins and evolution of predator-prey theory. *Ecology* 73 (5): 1530–1535.

Bono, Francesco Morgan, Luca Radicioni, Simone Cinquemani, Chiara Conese, and Marco Tarabini. 2022. Development of soft sensors based on neural networks for detection of anomaly working condition in automated machinery. In *Nde 4.0, predictive maintenance, and communication and energy systems in a globally networked world,* 12049:56–70. SPIE.

Brunton, Steven L, and J Nathan Kutz. 2022. *Data-driven science and engineering: machine learning, dynamical systems, and control.* Cambridge University Press.

Kudryashov, Nikolai A. 1990. Exact solutions of the generalized kuramoto-sivashinsky equation. *Physics Letters A* 147 (5-6): 287–291.

Sashidhar, Diya, and J Nathan Kutz. 2021. Bagging, optimized dynamic mode decomposition (bop-dmd) for robust, stable forecasting with spatial and temporal uncertainty-quantification. *arXiv preprint arXiv:2107.10878.*