Reduced order modeling for computational fluid dynamics

Nicola Petkovic, Giovanni Alessandro Clini[a]

[a]*Politecnico di Milano,*

Abstract

The focus of this work will be the study of ML-based, non-intrusive reduced order modeling strategies for transport-dominated systems. In particular this work has to be interpreted as a description of the methodology and the structure of the associated code, this doesn't implies the usage of the several models as black boxes, but as a way to get insight on how much is possible to simplify the "compression" problem. Indeed what we aimed is to reproduce as an experiment the methodology provided by (1)

Keywords: Reduced Order Modeling, Autoencoder, Linear Advection Problem

Contents

1. Acronyms & Abbreviations

Acronym	Definition
PDE	Partial Differential Equations
ROM	Reduced Order Model
ML	Machine Learningl
DNN	Deep Neural Network
LSTM	Long-Short Term Memory
NODE	Neural Ordinary Differential Equation
TCN	Temporal Convolutional Network
SVD	Singular Value Decomposition
POD	Proper Orthogonal Decomposition
RIC	Relative Information Content
NMSE	Normalized Mean Square Error

Table 1: Acronyms & Abbreviations

2. Introduction

The discussion is developed according to the following questions: what's the problem?, Why we want to solve it?, How it was previously solved? and Which are the main steps? Nowadays modern scientific computing relies on efficient numerical simulation of complex physical systems, especially for applications that seek solutions at different times or parameter instances. For these types of applications, the relevant physical system is typically described by a set of parameterized nonlinear partial differential equations (PDEs).

Numerical discretizations of such systems using a high-fidelity (finite element, finite volume, or finite difference type) computational solver can be prohibitively expensive as they generate high dimensional representations of the solution in order to accurately resolve multiple time and space scales and underlying non-linearities.

However, there is compelling scientific evidence to suggest that the underlying dynamics often exhibit low-dimensional structure, here must be underlined that this is the insight that drives the paper.

A useful tool to exploit this feature are the Reduced order models (ROMs), which have the right characteristics to replace such expensive high-fidelity solvers by exploiting the intrinsic, low-rank structure (further it will be seen the importance of that point) of the simulation data in order to create more tractable models for the spatiotemporal evolution dynamics of the PDE system.

A variety of data-driven, ML-based approximation frameworks has been proposed to model the propagation of system dynamics in the latent space. Some of the highly successful examples involve the use of deep neural networks (DNNs), long-short-term memory (LSTM) networks, neural ordinary differential equations (NODE), and temporal convolutional networks (TCNs).

To go into more detail, one fundamental assumption of linear reduced basis methods like POD is that any element in the solution manifold M of the nonlinear PDE system can be accurately approximated using a linear combination of a small number of basis functions e.g. Gaussian functions. Here must be underlined that traditionally this concept is quantified by the Kolmogorov n-width, Dn, which measures the error introduced by approximating any element f of M with an element g of a linear space En. A common heuristic approach to get trough

estimate of Dn for a particular discretized solution manifold is to examine the rate of decay of the singular values obtained by a SVD of the system snapshots(the piece of graph or of the function we will consider in the analisys). However better results than the ones obtained with the POD technique are achievable with supervised learning technique, indeed the idea of autoencoder architecture is perfectly suitable to our problem. In the following section is described such architecture and how it is implemented.

3. Code structure

3.1. Generating the data

In order to train the autoencoder network, data that represents a flow of a fluid has to be generated. To do so, an analytic function that generates a two-dimensional Gaussian pulse characterized by its amplitude, center position, and width - which are passed as parameters to the function - is implemented. Furthermore, a function called *plot_mesh* that creates a scatter plot and overlays a grid on top of it to create a mesh is developed. Meshes are commonly used in scientific computing for solving partial differential equations that describe physical phenomena. The vertices and edges of a mesh can be used to represent the locations and connectivity of grid points in a numerical simulation, and the faces can be used to define the boundaries and shapes of objects or regions of interest. Then, these two functions are used in order to generate and visualize snapshots of a linearly advecting 2D pulse with different sizes and parametric variations. After creating a uniform mesh, snapshots of the pulse for different pulse sizes - controlled by a parameter called *sigma_list* - and for different parametric variations - controlled by the parameter *shift_param_list* - are generated, visualized, and then locally saved.

3.2. The model

This section is dedicated to an autoencoder model for unsupervised learning using TensorFlow. The autoencoder model has three layers: an encoder layer, a shift layer, and a decoder layer. The encoder layer maps the input data to a latent representation of smaller dimensionality, while the decoder layer reconstructs the input data from the latent representation. The shift layer performs an additional non-linear transformation on the latent representation (i.e. the special part since the common autoencoder doesn't have it).

The autoencoder model is implemented using the Keras API of TensorFlow, which allows easy construction of neural networks. The model is defined as a subclass of the tf.keras.Model class. The Encoder, Shift, and Decoder layers are defined as subclasses of the tf.keras.layers.Layer class. The Encoder and Shift layers use the Dense layer of Keras to define fully connected layers, while the Decoder layer reconstructs the input data using a series of fully connected layers.

The AAautoencoder class defines the complete autoencoder model by combining the Encoder, Shift, and Decoder layers. The call method of the AAautoencoder class defines the forward pass of the autoencoder model, which consists of encoding the input data using the Encoder layer, applying the Shift layer to the encoded data, and decoding the shifted data using the Decoder layer. The code also includes a save model function that can be used to save the trained model to disk using the TensorFlow Saved Model format. The function takes as input the trained model, the input data, a model number, and a results dictionary. The saved model function first saves the trained model to disk using the Saved Model format and then computes the reconstruction error of the trained model on the input data. The reconstruction error is added to the results dictionary using the model number as the key.

3.3. Training

This is a custom training loop function for an AAautoencoder model. The function takes as inputs the number of training epochs, the training and validation datasets, the latent dimension of the model, the activation function used in the layers, the size of the first layer of the encoder and the last layer of the decoder, the number of timepoints in the time-series data, and additional keyword arguments for the learning rate, batch size, regularization weight, loss weight, loss lower bound, decay factor, and learning rate decay.

The function first initializes the model, optimizer, and lists to store the training and validation losses, shift losses, reconstruction losses, and regularization losses. Then, for each epoch, the function iterates over the training dataset batches and computes the forward pass of the encoder, shift decoder, and true decoder for the minibatch. The function then computes the L2 regularization loss and the total loss value for the minibatch based on the combination loss function and loss weights specified in the keyword arguments. The function also computes the regularization loss based on the L2 norm of the model parameters and the regularization weight specified in the keyword arguments. The function then computes the gradients of the total loss with respect to the trainable variables and applies the gradients to update the variable values using the Adam optimizer. Finally, the function logs the training loss values for the minibatch and appends the average loss values for the epoch to the corresponding lists.

After completing the training loop for all epochs, the function iterates over the validation dataset batches and computes the forward pass of the encoder, shift decoder, and true decoder for the minibatch. The function then computes the validation loss values for the minibatch based on the combination loss function and loss weights specified in the keyword arguments. The function also appends the average validation loss values for the epoch to the corresponding lists. The function returns the trained model and the lists of loss values for each epoch.

4. Discussion and conclusions

We demonstrated the capability of the advection-aware autoencoder architecture to generate a compressed representation for high-fidelity snapshots of Linear Advection Problem. In this study, LSTM architectures are chosen to build dynamics models for the purposes of illustration. However, the methodology could be easily adapted to use any other approximation framework that might be more appropriate for a particular problem.

Consider the advection of a circular Gaussian pulse traveling in the positive y direction through a rectangular domain and at a constant speed c. These kind of signals are the basic ones exploited to build the dataset for the machine learning procedure.

The training dataset is constructed using 460 highfidelity snapshots. The remaining snapshots corresponding are used to create a test dataset. This creates a geometrically parameterized training and testing dataset.

The AA autoencoder network is trained on the parametric training set for 8000 epochs using the Adam optimizer with an initial learning rate of 0.0001 that decays step-wise.

The training snapshots are divided into two sets—starting from the initial time point every alternate snapshot is used for training the AA autoencoder model, while the rest are reserved for validation during training. In this study, the losses computed on the validation points are solely used to monitor the extent of overfitting during training, and later to evaluate the accuracy of prediction on unseen time steps associated with a training parameter value.

After exploring a large space of network design parameters, the results are obtained with two of the most optimal AA autoencoder designs. In the first model (AA1), only the input feature is augmented by the parameter value; while in the second model (AA2), both the input feature and the output labels are augmented by the parameter value. The encoder network XE of both the models is constructed with three hidden layers composed of 629, 251 and 62 units that connect an input feature (i.e., augmented snapshot) of dimension N = 100, 702 to an encoded latent vector representation of dimension k. For both the models, the decoder networks XD and PHIS are set up to be mirror images of the corresponding encoder network. The AA1 model uses the selu activation function for the hidden layers followed by a linear activation on the output layer, while the AA2 model uses the swish activation function for the hidden layers. The individual loss components L1 and L2 are defined as a weighted combination of the normalized mean square error (NMSE) loss and the pseudo-Huber loss.

The decoder predictions are evaluated for the two parameter values at the boundaries of the training range sigma equal to five and sigma equal to 16 as they present distinct challenges. Autoencoders are known to struggle with the extraction of discontinuities in the input feature space. The snapshot data for sigma five features a very steep gradient in the shape of the pulse profile which poses some of the same challenges as a discontinuous profile. Moreover, a single encoder network XE is being tasked, by design, to combine with two independent decoder networks XD and PHIS to map into both a stationary discontinuity as well as a moving discontinuity. Thus the spatial distribution of reconstruction error for the sigma five profile is more localized near the moving pulse, whereas that of the sigma sixteen profile is more uniformly spread out across the spatial domain. Despite all of these minor differences in prediction performance, there is a high degree of agreement between the full order decoder predictions and the high-dimensional snapshots, with less than 4 percentage parameter values.

Loss in accuracy with extrapolatory predictions for a geometrically parameterized dataset is one of the well-known challenges faced by both intrusive and non-intrusive reduced order modeling approaches, which requires particular attention to resolve representation issues posed by the topology of the parametric solution manifold. Thus, as expected, there is a noticeable drop in accuracy for the prediction of full order solutions, with the errors being especially localized near the moving pulse profile. This effect is also reflected in the relative error plots for the two unseen parameter values in test test.

However, the quality of predictions are still quite encouraging considering that these are purely extrapolatory predictions on an unseen parameter instance, without any special treatment of the solution manifold or modification of the training process. To asses this result instead of the accuracy metric it was decided to compare the generated data with the reconstructed ones by the decoder.

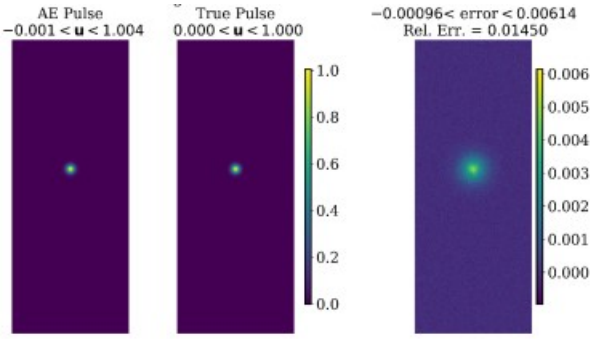

Figure 1: The figure shows the desired results, i.e. the one that allow to say that the reconstruction is good

References

[1] Dutta, S.; Rivera-Casillas, P.; Styles, B.; Farthing, M.W. Reduced Order Modeling Using Advection-Aware Autoencoders. Math. Comput. Appl. 2022, 27, 34. https://doi.org/10.3390/mca27030034

[2] Gonzalez, F.J.; Balajewicz, M. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. arXiv 2018, arXiv:1808.01346

[3] Bakarji, J.; Champion, K.; Kutz, J.N.; Brunton, S.L. Discovering Governing Equations from Partial Measurements with Deep Delay Autoencoders. arXiv 2022, arXiv:2201.05136.

[4] Maulik, R.; Lusch, B.; Balaprakash, P. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. Phys. Fluids 2021, 33, 037106.

[5] Hinton, G.E.; Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. Science 2006, 313, 504–507. Fluids 2021, 33, 037106.

[6] Mendible, A.; Brunton, S.L.; Aravkin, A.Y.; Lowrie, W.; Kutz, J.N. Dimensionality reduction and reduced-order modeling for traveling wave physics. Theor. Comput. Fluid Dyn. 2020, 34, 385–400.