

# A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs

Stefania Fresca<sup>a</sup>, Luca Dedé<sup>a</sup>, Andrea Manzoni<sup>a,\*</sup>

<sup>a</sup>MOX - Dipartimento di Matematica, Politecnico di Milano, P.zza Leonardo da Vinci 32, 20133 Milano, Italy

## Abstract

Traditional reduced order modeling techniques such as the reduced basis (RB) method (relying, e.g., on proper orthogonal decomposition (POD)) suffer from severe limitations when dealing with nonlinear time-dependent parametrized PDEs, because of the fundamental assumption of linear superimposition of modes they are based on. For this reason, in the case of problems featuring coherent structures that propagate over time such as transport, wave, or convection-dominated phenomena, the RB method usually yields inefficient reduced order models (ROMs) if one aims at obtaining reduced order approximations sufficiently accurate compared to the high-fidelity, full order model (FOM) solution. To overcome these limitations, in this work, we propose a new nonlinear approach to set reduced order models by exploiting deep learning (DL) algorithms. In the resulting nonlinear ROM, which we refer to as DL-ROM, both the nonlinear trial manifold (corresponding to the set of basis functions in a linear ROM) as well as the nonlinear reduced dynamics (corresponding to the projection stage in a linear ROM) are learned in a non-intrusive way by relying on DL algorithms; the latter are trained on a set of FOM solutions obtained for different parameter values. In this paper, we show how to construct a DL-ROM for both linear and nonlinear time-dependent parametrized PDEs; moreover, we assess its accuracy on test cases featuring different parametrized PDE problems. Numerical results indicate that DL-ROMs whose dimension is equal to the intrinsic dimensionality of the PDE solutions manifold are able to approximate the solution of parametrized PDEs in situations where a huge number of POD modes would be necessary to achieve the same degree of accuracy.

**Keywords:** parametrized PDEs, nonlinear time-dependent PDEs, reduced order modeling, deep learning, proper orthogonal decomposition

## 1. Introduction

The solution of a parametrized system of partial differential equations (PDEs) by means of a *full-order model* (FOM), whenever dealing with real-time or multi-query scenarios, entails prohibitive computational costs if the FOM is high-dimensional. In the former case, the FOM solution must be computed in a very limited amount of time; in the latter one, the FOM must be solved for a huge number of parameter instances sampled from the parameter space. Reduced order modeling techniques aim at replacing the FOM by a reduced order model (ROM), featuring a much lower dimension, still able to express the physical features of the problem described by the FOM. The basic assumption underlying the construction of such a ROM is that the solution of a parametrized PDE, belonging a priori to a high-dimensional (discrete) space, lies on a low-dimensional manifold embedded in this space. The goal of a ROM is then to approximate the *solution manifold* – that is, the set of all PDE solutions when the parameters vary in the parameter space – through a suitable, approximated *trial manifold*.

\*Corresponding author

Email addresses: [stefania.fresca@polimi.com](mailto:stefania.fresca@polimi.com) (Stefania Fresca), [luca.dede@polimi.com](mailto:luca.dede@polimi.com) (Luca Dedé), [andrea1.manzoni@polimi.com](mailto:andrea1.manzoni@polimi.com) (Andrea Manzoni)

A widespread family of reduced order modeling techniques relies on the assumption that the reduced-order approximation can be expressed by a linear combination of basis functions, built starting from a set of FOM solutions, called snapshots. Among these techniques, proper orthogonal decomposition (POD) – equivalent to principal component analysis in statistics [1], or Karhunen-Loëve expansion in stochastic applications – exploits the singular value decomposition of a suitable snapshot matrix (or the eigen-decomposition of the corresponding snapshot correlation matrix), thus yielding *linear* ROMs, in which the ROM approximation is given by the linear superimposition of POD modes. In this case, the solution manifold is approximated through a *linear* trial manifold, that is, the ROM approximation is sought in a low-dimensional linear trial subspace.

Projection-based methods are linear ROMs in which the ROM approximation of the PDE solution, for any new parameter value, results from the solution of a low-dimensional (nonlinear, dynamical) system, whose unknowns are the ROM degrees of freedom (or generalized coordinates). Despite the PDE (and thus the FOM) being linear or not, the operators appearing in the ROM are obtained by imposing that the projection of the FOM residual evaluated on the ROM trial solution is orthogonal to a low-dimensional, linear test subspace, which might coincide with the trial subspace. Hence, no matter whether the PDE is linear or not, the resulting ROM is *linear* since the reduced dynamics is obtained through a projection onto a linear subspace [2, 3, 4]. However, linear ROMs show severe computational bottlenecks when dealing with problems featuring coherent structures (possibly dependent on parameters) that propagate over time, namely in transport and wave-type phenomena, or convection-dominated flows. In these cases, the dimension of the linear trial manifold can easily become extremely large if compared to the intrinsic dimension of the solution manifold for the sake of accuracy, thus compromising the ROM efficiency. To overcome this bottleneck, *ad-hoc* extensions of the POD strategy have been considered, towards nonlinear approaches to build a ROM [5, 6].

In this paper we propose a computational, non-intrusive approach based on deep learning (DL) algorithms to deal with the construction of efficient ROMs (which we refer to as DL-ROMs) in order to tackle parameter-dependent PDEs; in particular, we consider PDEs that feature wave-type phenomena. A comprehensive framework is presented for the global approximation of the map  $(t, \boldsymbol{\mu}) \mapsto \mathbf{u}_h(t, \boldsymbol{\mu})$ , where  $t \in (0, T)$  denotes time,  $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}$  a vector of input parameters and  $\mathbf{u}_h(t, \boldsymbol{\mu}) \in \mathbb{R}^{N_h}$  the solution of a large-scale dynamical system arising from the space discretization of a parametrized, time-dependent (non)linear PDE. Several recent works have shown possible applications of DL techniques to parametrized PDEs – thanks to their approximation capabilities, their extremely favorable computational performances during online testing phases, and their relative easiness of implementation – both from a theoretical [7] and a computational standpoint. Regarding this latter aspect, artificial neural networks (ANN), such as feedforward neural networks, have been employed to model the reduced dynamics in a data-driven [8], and less intrusive way (avoiding, e.g., the costs entailed by projection-based ROMs), but still relying on a linear trial manifold built, e.g., through POD. For instance, in [9, 10, 11, 12] the solution of a (nonlinear, time-dependent) ROM for any new parameter value has been replaced by the evaluation of ANN-based regression models; similar ideas can be found, e.g., in [13, 14, 15]. Few attempts have been made in order to describe the reduced trial manifold where the approximation is sought (avoiding, e.g., the linear superimposition of POD modes) through ANNs, see, e.g., [16, 17].

For instance, a projection-based ROM technique has been introduced in [17], in which the FOM system is projected onto a nonlinear trial manifold identified by means of the decoder function of a convolutional autoencoder neural network. However, the ROM is derived by minimizing a residual formulation, for which the quasi-Newton method herein employed requires the computation of an approximated Jacobian of the residual at each time step. A ROM technique based on a deep convolutional recurrent autoencoder has been proposed in [16], where a reduced trial manifold is obtained by means of a convolutional autoencoder; the latter is then used to train a Long Short-Term Memory (LSTM) neural network modeling the reduced dynamics. However, no explicit parameter dependence in the PDE problem is considered, apart from  $\boldsymbol{\mu}$ -dependent initial data, and the LSTM is trained on reduced approximations obtained through the encoder function of the autoencoder. Another promising application of machine and deep learning techniques within a ROM framework deals with the efficient evaluation of reduced error models, see, e.g., [18, 19, 20, 21].

Our goal is to set up nonlinear ROMs whose dimension is nearly equal (if not equal) to the intrinsic dimension of the solution manifold that we aim at approximating. Our DL-ROM approach combines and improves the techniques introduced in [16, 17] by shaping an all-inclusive DL-based ROM technique, where we both (*i*) construct the reduced trial manifold and (*ii*) model the reduced dynamics on it employing ANNs. The former task is achieved by using the decoder function of a convolutional autoencoder; the latter task is instead carried out by considering a feedforward neural network and the encoder function of a convolutional autoencoder. Moreover, we set up a computational procedure performing the training of both network architectures simultaneously, by minimizing a loss function that weights two terms, one dedicated to each single task. In this respect, we are able to design a flexible framework capable to handle parameters affecting both PDE operators and data, which avoids both the expensive projection stage of [17] and the training of a more expensive LSTM network. In our technique, the intrusive construction of a ROM is replaced by the evaluation of the ROM generalized coordinates through a deep feedforward neural network taking only  $(t, \boldsymbol{\mu})$  as inputs. The proposed technique is purely data-driven, that is, it only relies on the computation of a set of FOM snapshots – in this respect, DL does not replace the high-fidelity FOM as, e.g., in the works by Karniadakis and coauthors [22, 23, 24, 25, 26]; rather, DL techniques are built upon it, to enhance the repeated evaluation of the FOM for different values of the parameters.

The structure of the paper is as follows. In section 2 we show how to generate nonlinear ROMs by reinterpreting the classical ideas behind linear ROMs for parametrized PDEs. In section 3 we detail the construction of the proposed DL-ROM, whose accuracy is numerically assessed in section 4 by considering three different test cases of increasing complexity (both with respect to the parametric dependence and the nature of the PDE). Finally, some conclusions are drawn in section 5. A quick overview of useful facts about deep feedforward, convolutional and autoencoders neural networks is reported in Appendix A to make the paper self-contained.

## 2. From linear to nonlinear dimensionality reduction

Starting from the well-known setting of linear (projection-based) ROMs, in this section we generalize this task to the case of nonlinear ROMs.

### 2.1. Problem formulation

We formulate the construction of ROMs in algebraic terms, starting from the high-fidelity (spatial) approximation of nonlinear, time-dependent, parametrized PDEs. By introducing suitable space discretizations techniques (such as, e.g., the Finite Element Method, Isogeometric Analysis or the Spectral Element Method) the high-fidelity, full order model (FOM) can be expressed as a nonlinear parametrized dynamical system. Given  $\boldsymbol{\mu} \in \mathcal{P}$ , we aim at solving the initial value problem

$$\begin{cases} \dot{\mathbf{u}}_h(t; \boldsymbol{\mu}) = \mathbf{f}(t, \mathbf{u}_h(t; \boldsymbol{\mu}); \boldsymbol{\mu}) & t \in (0, T) \\ \mathbf{u}_h(0; \boldsymbol{\mu}) = \mathbf{u}_0(\boldsymbol{\mu}), \end{cases} \quad (1)$$

where the parameter space  $\mathcal{P} \subset \mathbb{R}^{n_\mu}$  is a bounded and closed set,  $\mathbf{u}_h : [0, T] \times \mathcal{P} \rightarrow \mathbb{R}^{N_h}$  is the parametrized solution of (1),  $\mathbf{u}_0 : \mathcal{P} \rightarrow \mathbb{R}^{N_h}$  is the initial datum and  $\mathbf{f} : (0, T) \times \mathbb{R}^{N_h} \times \mathcal{P} \rightarrow \mathbb{R}^{N_h}$  is a (nonlinear) function, encoding the system dynamics. The FOM dimension  $N_h$  is related with the finite dimensional subspaces introduced for the space discretization of the PDE – here  $h > 0$  usually denotes a discretization parameter, such as the maximum diameter of elements in a computational mesh – and can be extremely small whenever the PDE problem shows complex physical behaviors and/or high degrees of accuracy are required to its solution. The parameter  $\boldsymbol{\mu} \in \mathcal{P}$  may represent physical or geometrical properties of the system, like, e.g., material properties, initial and boundary conditions, or the shape of the domain. In order to solve problem (1), suitable time discretizations are employed, such as backward differentiation formulas [27].

Our goal is the efficient numerical approximation of the whole set

$$\mathcal{S}_h = \{\mathbf{u}_h(t; \boldsymbol{\mu}) \mid t \in [0, T] \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h}, \quad (2)$$

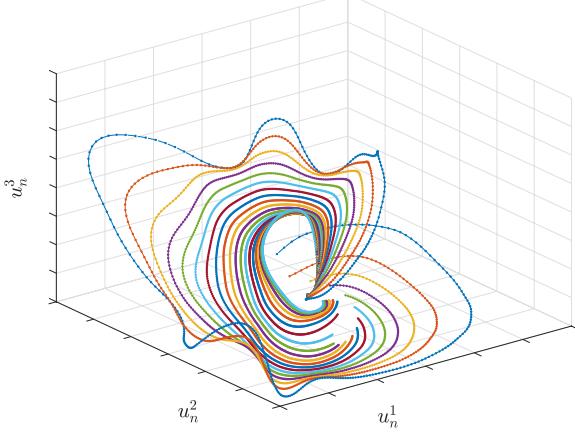


Figure 1: Example of a two-dimensional manifold embedded in  $\mathbb{R}^3$ . Each curve represents the time-evolution of the first three components of the solution of a (nonlinear) parametrized PDE for a fixed parameter value  $\mu$ .

of solutions to problem (1) when  $(t; \mu)$  varies in  $[0, T] \times \mathcal{P}$ , also referred to as *solution manifold* (a sketch is provided in Figure 1). Assuming that, for any given parameter  $\mu \in \mathcal{P}$ , problem (1) admits a unique solution, for each  $t \in (0, T)$ , the intrinsic dimension of the solution manifold is at most  $n_\mu + 1 \ll N_h$ , where  $n_\mu$  is the number of parameters (time plays the role of an additional coordinate). This means that each point  $\mathbf{u}_h(t; \mu)$  belonging to  $\mathcal{S}_h$  is completely defined in terms of at most  $n_\mu + 1$  intrinsic coordinates, or equivalently, the tangent space to the manifold at any given  $\mathbf{u}_h(t; \mu)$  is spanned by  $n_\mu + 1$  basis vectors.

## 2.2. Linear dimensionality reduction: projection-based ROMs

The most common way to build a ROM for the efficient approximation of problem (1) relies on the introduction of a *reduced linear trial manifold*, that is of a subspace  $\tilde{\mathcal{S}}_n = \text{Col}(V)$  of dimension  $n \ll N_h$ , spanned by the  $n$  columns of a matrix  $V \in \mathbb{R}^{N_h \times n}$ . Hence, a linear ROM looks for an approximation  $\tilde{\mathbf{u}}_h(t; \mu) \approx \mathbf{u}_h(t; \mu)$  in the form

$$\tilde{\mathbf{u}}_h(t; \mu) = V \mathbf{u}_n(t; \mu), \quad (3)$$

where  $\tilde{\mathbf{u}}_h : [0, T] \times \mathcal{P} \rightarrow \tilde{\mathcal{S}}_n$ . Here  $\mathbf{u}_n(t; \mu) \in \mathbb{R}^n$  for each  $t \in [0, T]$ ,  $\mu \in \mathcal{P}$  denotes the vector of intrinsic coordinates (or degrees of freedom) of the ROM approximation; note that the map

$$\Psi_h : \mathbb{R}^n \rightarrow \mathbb{R}^{N_h}, \quad \mathbf{s}_n \mapsto \tilde{\mathbf{s}}_h = V \mathbf{s}_n$$

that, given the (low-dimensional) intrinsic coordinates, returns the (high-dimensional) approximation of the FOM solution  $\mathbf{u}_h(t; \mu)$ , is linear.

Proper Orthogonal Decomposition (POD) is one of the most widely employed techniques to generate the linear trial manifold [4]. Considering a set of  $N_{train}$  instances of the parameter  $\mu \in \mathcal{P}$ , we introduce the snapshot matrix  $S \in \mathbb{R}^{N_h \times N_s}$  defined as

$$S = [\mathbf{u}(t^1; \mu_1) \mid \dots \mid \mathbf{u}(t^{N_t}; \mu_1) \mid \dots \mid \mathbf{u}(t^1; \mu_{N_{train}}) \mid \dots \mid \mathbf{u}(t^{N_t}; \mu_{N_{train}})], \quad (4)$$

where we have introduced a partition of the time interval  $[0, T]$  in  $N_t$  time steps  $\{t^k\}_{k=1}^{N_t}$ ,  $t^k = k\Delta t$ , of size  $\Delta t = T/N_t$  and  $N_s = N_{train}N_t$ . Moreover, let us introduce a symmetric and positive definite matrix  $X_h \in \mathbb{R}^{N_h \times N_h}$  encoding a suitable norm (e.g., the energy norm) on the high-dimensional space and admitting a Cholesky factorization  $X_h = H^T H$ . POD computes the Singular Value Decomposition (SVD) of  $HS$ ,

$$HS = U \Sigma Z^T,$$

where  $U = [\zeta_1 \mid \dots \mid \zeta_{N_h}] \in \mathbb{R}^{N_h \times N_h}$ ,  $Z = [\psi_1 \mid \dots \mid \psi_{N_s}] \in \mathbb{R}^{N_s \times N_s}$  and  $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r) \in \mathbb{R}^{N_h \times N_s}$  with  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$ , and  $r \leq \min(N_h, N_s)$ , and sets the columns of  $V$  in terms of the first  $n$  left singular

vectors of  $S$  that is,  $V = [H^{-1}\zeta_1 | \dots | H^{-1}\zeta_n]$ . By construction, the columns of  $V$  are orthonormal (with respect to the scalar product  $(\cdot, \cdot)_{X_h}$ ) and among all possible  $n$ -dimensional subspaces spanned by the column of a matrix  $W \in \mathbb{R}^{N_h \times n}$ ,  $V$  provides the best reconstruction of the snapshots, that is,

$$\sum_{i=1}^{N_{train}} \sum_{k=1}^{N_t} \|\mathbf{u}(t^k; \boldsymbol{\mu}_i) - VV^T X_h \mathbf{u}(t^k; \boldsymbol{\mu}_i)\|_{X_h}^2 = \min_{W \in \mathcal{V}_n} \sum_{i=1}^{N_{train}} \sum_{k=1}^{N_t} \|\mathbf{u}(t^k; \boldsymbol{\mu}_i) - WW^T X_h \mathbf{u}(t^k; \boldsymbol{\mu}_i)\|_{X_h}^2, \quad (5)$$

where  $\mathcal{V}_n = \{W \in \mathbb{R}^{N_h \times n} : W^T X_h W = I\}$ . For this reason, we refer to  $VV^T X_h \mathbf{u}_h(t; \boldsymbol{\mu})$  as to the optimal-POD reconstruction of  $\mathbf{u}_h(t; \boldsymbol{\mu})$  onto a reduced subspace of dimension  $n < N_h$ .

In order to model the reduced dynamics of the system, that is, the time-evolution of the generalized coordinates  $\mathbf{u}_n(t; \boldsymbol{\mu})$ , we can replace  $\mathbf{u}_h(t; \boldsymbol{\mu})$  by (3) in system (1),

$$\begin{cases} V\dot{\mathbf{u}}_n(t; \boldsymbol{\mu}) = \mathbf{f}(t, V\mathbf{u}_n(t; \boldsymbol{\mu}); \boldsymbol{\mu}) & t \in (0, T) \\ V\mathbf{u}_n(0; \boldsymbol{\mu}) = \mathbf{u}_0(\boldsymbol{\mu}), \end{cases} \quad (6)$$

and impose that the residual

$$\mathbf{r}_h(V\mathbf{u}_n(t; \boldsymbol{\mu})) = V\dot{\mathbf{u}}_n(t; \boldsymbol{\mu}) - \mathbf{f}(t, V\mathbf{u}_n(t; \boldsymbol{\mu}); \boldsymbol{\mu}) \quad (7)$$

associated to the first equation of (6) is orthogonal to a  $n$ -dimensional subspace spanned by the column of a matrix  $Y \in \mathbb{R}^{N_h \times n}$ , that is,  $Y^T \mathbf{r}_h(V\mathbf{u}_n) = \mathbf{0}$ . This condition yields the following ROM

$$\begin{cases} Y^T V\dot{\mathbf{u}}_n(t; \boldsymbol{\mu}) = Y^T \mathbf{f}(t, V\mathbf{u}_n(t; \boldsymbol{\mu}); \boldsymbol{\mu}) & t \in (0, T) \\ \mathbf{u}_n(0; \boldsymbol{\mu}) = (Y^T V)^{-1} Y^T \mathbf{u}_0(\boldsymbol{\mu}). \end{cases} \quad (8)$$

In the case  $Y = V$  a Galerkin projection is performed, while the case  $Y \neq V$  yields a more general Petrov-Galerkin projection. Note that choosing  $Y$  such that  $Y^T V = I \in \mathbb{R}^{N_h \times N_h}$  does not automatically ensure ROM stability on long time intervals.

The RB method under the form of either Galerkin-POD or Petrov-Galerkin-POD methods has been successfully applied to a broad range of parametrized time-dependent (non)linear problems (see, e.g., [6, 28]) however it provides low-dimensional subspaces of dimension  $n \gg n_\mu + 1$  much larger than the intrinsic dimension of the solution manifold – relying on a linear, global trial manifold thus represent a major bottleneck to computational efficiency [5, 6]. This is the case, for instance, of hyperbolic problems, for which the RB method is not able in practice to significantly decrease the dimensionality of the problem. The same difficulty might also affect the use of hyper-reduction techniques, such as the (discrete) empirical interpolation [29, 30], mandatory in order to assemble the operators appearing in the ROM (8) without relying on expensive  $N_h$ -dimensional arrays. See, e.g., [31] for further details.

### 2.3. Nonlinear dimensionality reduction

A first attempt to overcome the computational bottleneck entailed by the use of a linear, global trial manifold is to build a *piecewise* linear trial manifold, using local reduced bases whose dimension is smaller than the one of the global linear trial manifold. Clustering algorithms applied on a set of snapshots can be employed to partition them into  $N_c$  clusters from which POD can extract a subspace of reduced dimension; the ROM is then obtained by following the strategy described above on each cluster separately, see, e.g. [32, 33]. An alternative approach based on classification binary trees has been introduced in [34]. These strategies have been employed (and compared) in [6] in order to solve parametrized problems in cardiac electrophysiology. Using a piecewise linear trial manifold partially overcomes the limitation of a linear dimensionality reduction technique as POD, yet employing local bases of dimension much higher than the intrinsic dimension of the solution manifold  $\mathcal{S}_h$ . An approach based on a dictionary of solutions, computed offline, has been developed in [35] as an alternative to using a truncated reduced basis based on POD, together with an online  $L^1$ -norm minimization of the residual.

Other possible options involving nonlinear transformations of modes might rely on a reconstruction of the POD modes at each time step using Lax pairs [36], on the solution of Monge-Kantorovich optimal transport problems [37], on a problem-dependent change of coordinates requiring the solution of an optimization problem repeatedly [38], on shifted POD modes [39] after multiple transport velocities have been identified and separated, or again basis updates are derived from querying the full model at a few selected spatial coordinates [40]. Despite providing remarkable improvements compared to the *classic* (Petrov-)Galerkin-POD approach, all these strategies exhibit some drawbacks, such as: *(i)* the high computational costs entailed during the online testing evaluation stage of the ROM – which is not restricted to the intensive offline training stage; *(ii)* performances and settings are highly dependent on the problem at hand; *(iii)* the need to deal only with a linear superimposition of modes (which characterizes linear ROMs), yielding low-dimensional spaces whose dimension is still (much) higher than the intrinsic dimension of the solution manifold.

Motivated by the need of avoiding the drawbacks of linear ROMs and setting a general paradigm for the construction of efficient, extremely low-dimensional ROMs, we resort to nonlinear dimensionality reduction techniques. Similarly to [16, 17], we build a nonlinear ROM to approximate  $\mathbf{u}_h(t; \boldsymbol{\mu}) \approx \tilde{\mathbf{u}}_h(t; \boldsymbol{\mu})$  by

$$\tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}) = \Psi_h(\mathbf{u}_n(t; \boldsymbol{\mu})), \quad (9)$$

where  $\Psi_h : \mathbb{R}^n \rightarrow \mathbb{R}^{N_h}$ ,  $\Psi_h : \mathbf{s}_n \mapsto \Psi_h(\mathbf{s}_n)$ ,  $n \ll N_h$ , is a nonlinear, differentiable function. As a matter of fact, the solution manifold  $\mathcal{S}_h$  is approximated by a *reduced nonlinear trial manifold*

$$\tilde{\mathcal{S}}_n = \{\Psi_h(\mathbf{u}_n(t; \boldsymbol{\mu})) \mid \mathbf{u}_n(t; \boldsymbol{\mu}) \in \mathbb{R}^n, t \in [0, T] \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h} \quad (10)$$

so that  $\tilde{\mathbf{u}}_h : [0, T] \times \mathcal{P} \rightarrow \tilde{\mathcal{S}}_n$ . As before,  $\mathbf{u}_n : [0, T] \times \mathcal{P} \rightarrow \mathbb{R}^n$  denotes the vector-valued function of two arguments representing the intrinsic coordinates of the ROM approximation. Our goal is to set a ROM whose dimension  $n$  is as close as possible to the intrinsic dimension  $n_\mu + 1$  of the solution manifold  $\mathcal{S}_h$ , i.e.  $n \geq n_\mu + 1$ , in order to correctly capture the solution of the dynamical system by containing the size of the approximation spaces [17].

To model the relationship between each couple  $(t, \boldsymbol{\mu}) \mapsto \mathbf{u}_n(t, \boldsymbol{\mu})$ , and to describe the system dynamics on the reduced nonlinear trial manifold  $\tilde{\mathcal{S}}_n$  in terms of the intrinsic coordinates, we consider a nonlinear map under the form

$$\mathbf{u}_n(t; \boldsymbol{\mu}) = \Phi_n(t, \boldsymbol{\mu}), \quad (11)$$

where  $\Phi_n : [0, T] \times \mathbb{R}^{n_\mu+1} \rightarrow \mathbb{R}^n$  is a differentiable nonlinear function. No additional assumptions such as, e.g., the (exact, or approximate) affine  $\boldsymbol{\mu}$ -dependence as in the RB method, are needed.

### 3. A deep learning-based reduced order model (DL-ROM)

We now detail the construction of the proposed nonlinear ROM. In this respect, we define the functions  $\Psi_h$  and  $\Phi_n$  in (9) and (11) by means of deep learning (DL) algorithms, exploiting neural network architectures. This choice is motivated by their ability of effectively approximating nonlinear maps, and by their ability to learn from data and generalize to unseen data. On the other hand, DL models enable us to build non-intrusive, completely data-driven, ROMs, since their construction only requires to access the dataset, the parameter values and the snapshot matrix, but not the FOM arrays appearing in (1).

The DL-ROM technique that we develop in this paper is composed by two main blocks responsible, respectively, for the *reduced dynamics learning* and the *reduced trial manifold learning* (see Figure 2). Hereon, we denote by  $N_{train}$ ,  $N_{test}$  and  $N_t$  the number of training-parameter instances, of testing-parameter instances and time instances, respectively, and we set  $N_s = N_{train}N_t$ . The dimension of both the FOM solution and the ROM approximation is  $N_h$ , while  $n$  denotes the number of intrinsic coordinates, with  $n \ll N_h$ .

For the description of the system dynamics on the reduced nonlinear trial manifold (which we refer to as *reduced dynamics learning*), we employ a *deep feedforward neural network* (DFNN) with  $L$  layers, that is, we define the function  $\Phi_n$  in definition (11) as

$$\Phi_n(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}) = \phi_n^{DF}(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}), \quad (12)$$

thus yielding the map

$$(t, \mu) \mapsto \mathbf{u}_n(t; \mu, \theta_{DF}) = \phi_n^{DF}(t; \mu, \theta_{DF}),$$

where  $\phi_n^{DF}$  takes the form (A.1), with  $t \in [0, T]$ , and results from the subsequent composition of a nonlinear activation function, applied to a linear transformation of the input,  $L$  times. Here  $\mu \in \mathcal{P} \subset \mathbb{R}^{n_\mu}$  and  $\theta_{DF}$  denotes the vector of parameters of the DFNN.

Regarding instead the description of the reduced nonlinear trial manifold  $\tilde{\mathcal{S}}_n$  defined in (10) (which we refer to as *reduced trial manifold learning*), we employ the *decoder function of a convolutional autoencoder* (AE), that is, we define the function  $\Psi_h$  appearing in (9) and (10) as

$$\Psi_h(\mathbf{u}_n(t; \mu, \theta_{DF}); \theta_D) = \mathbf{f}_h^D(\mathbf{u}_n(t; \mu, \theta_{DF}); \theta_D), \quad (13)$$

thus yielding the map

$$\mathbf{u}_n(t; \mu, \theta_{DF}) \mapsto \tilde{\mathbf{u}}_h(t; \mu, \theta) = \mathbf{f}_h^D(\mathbf{u}_n(t; \mu, \theta_{DF}); \theta_D),$$

where  $\mathbf{f}_h^D$  results from the composition of several layers, some of which of convolutional type, overall depending on the vector  $\theta_D$  of parameters of the decoder function.

Combining the two former stages, the DL-ROM approximation is then given by

$$\tilde{\mathbf{u}}_h(t; \mu, \theta) = \mathbf{f}_h^D(\phi_n^{DF}(t; \mu, \theta_{DF}); \theta_D), \quad (14)$$

where  $\phi_n^{DF}(\cdot; \cdot, \theta_{DF}) : \mathbb{R}^{(n_\mu+1)} \rightarrow \mathbb{R}^n$  and  $\mathbf{f}_h^D(\cdot; \theta_D) : \mathbb{R}^n \rightarrow \mathbb{R}^{N_h}$  are defined as in (12) and (13), respectively, and  $\theta = (\theta_{DF}, \theta_D)$  are the parameters defining the neural network. The architecture of DL-ROM is shown in Figure 2.

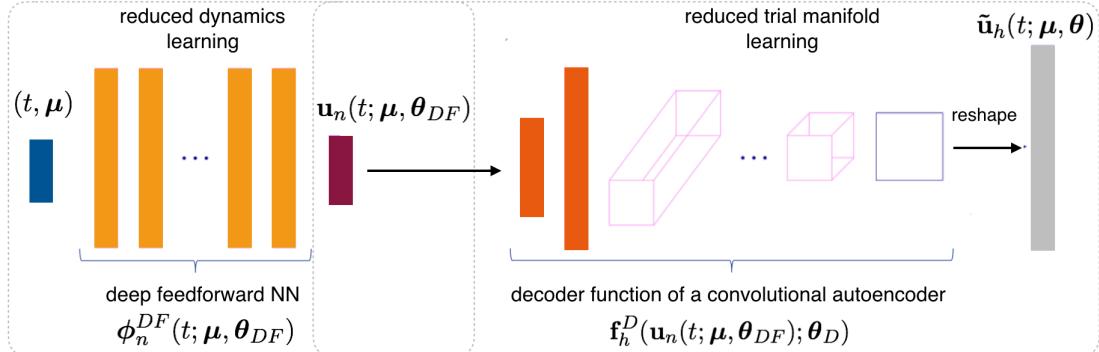


Figure 2: DL-ROM architecture (online stage, testing).

Computing the ROM approximation (14) for any new value of  $\mu \in \mathcal{P}$ , at any given time, requires to evaluate the map  $(t, \mu) \rightarrow \tilde{\mathbf{u}}_h(t; \mu, \theta)$  at the testing stage, once the parameters  $\theta = (\theta_{DF}, \theta_D)$  have been determined, once and for all, during the training stage. The training stage consists in solving an optimization problem (in the variable  $\theta$ ) after a set of snapshots of the FOM have been computed. More precisely, provided the parameter matrix  $M \in \mathbb{R}^{(n_\mu+1) \times N_s}$  defined as

$$M = [(t^1, \mu_1) | \dots | (t^{N_t}, \mu_1) | \dots | (t^1, \mu_{N_{train}}) | \dots | (t^{N_t}, \mu_{N_{train}})], \quad (15)$$

and the snapshot matrix  $S$ , defined in (4), we solve the problem: find the optimal parameters  $\theta^*$  solution of

$$\mathcal{J}(\theta) = \frac{1}{N_s} \sum_{i=1}^{N_{train}} \sum_{k=1}^{N_t} \mathcal{L}(t^k, \mu_i; \theta) \rightarrow \min_{\theta} \quad (16)$$

where

$$\mathcal{L}(t^k, \mu_i; \theta) = \frac{1}{2} \|\mathbf{u}_h(t^k; \mu_i) - \tilde{\mathbf{u}}_h(t^k; \mu_i, \theta)\|^2 = \frac{1}{2} \|\mathbf{u}_h(t^k; \mu_i) - \mathbf{f}_h^D(\phi_n^{DF}(t^k; \mu_i, \theta_{DF}); \theta_D)\|^2. \quad (17)$$

To solve the optimization problem (16)-(17) we use the ADAM algorithm [41] which is a stochastic gradient descent method [42] computing an adaptive approximation of the first and second momentum of the gradients of the loss function. In particular, it computes exponentially weighted moving averages of the gradients and of the squared gradients. We set the starting learning rate to  $\eta = 10^{-4}$ , the batch size to  $N_b = 20$  and the maximum number of epochs to  $N_{\text{epochs}} = 10000$ . We perform cross-validation, in order to tune the hyper-parameters of the DL-ROM, by splitting the data in training and validation and following a proportion 8:2. Moreover, we implement an early-stopping regularization technique to reduce overfitting [43]. In particular, we stop the training if the loss does not decrease over 500 epochs. As nonlinear activation function we employ the ELU function [44] defined as

$$\sigma(z) = \begin{cases} z & z \geq 0 \\ \exp(z) - 1 & z < 0. \end{cases}$$

No activation function is applied at the last convolutional layer of the decoder neural network, as usually done when dealing with autoencoders. The parameters, weights and biases, are initialized through the He uniform initialization [45].

As we rely on a convolutional autoencoder to define the function  $\Psi_h$ , we also exploit the encoder function

$$\tilde{\mathbf{u}}_n(t; \mu, \theta_E) = \mathbf{f}_n^E(\mathbf{u}(t; \mu); \theta_E), \quad (18)$$

which maps each FOM solution associated to the pairs  $(t; \mu) \in \text{Col}(M)$  provided as inputs to the feed-forward neural network (12), onto a low-dimensional representation  $\tilde{\mathbf{u}}_n(t; \mu, \theta_E)$  depending on the parameters vector  $\theta_E$  defining the encoder function.

Indeed, the actual architecture of DL-ROM that is used only during the training and the validation phases, but not during testing, is the one shown in Figure 3. In practice, we add to the architecture of

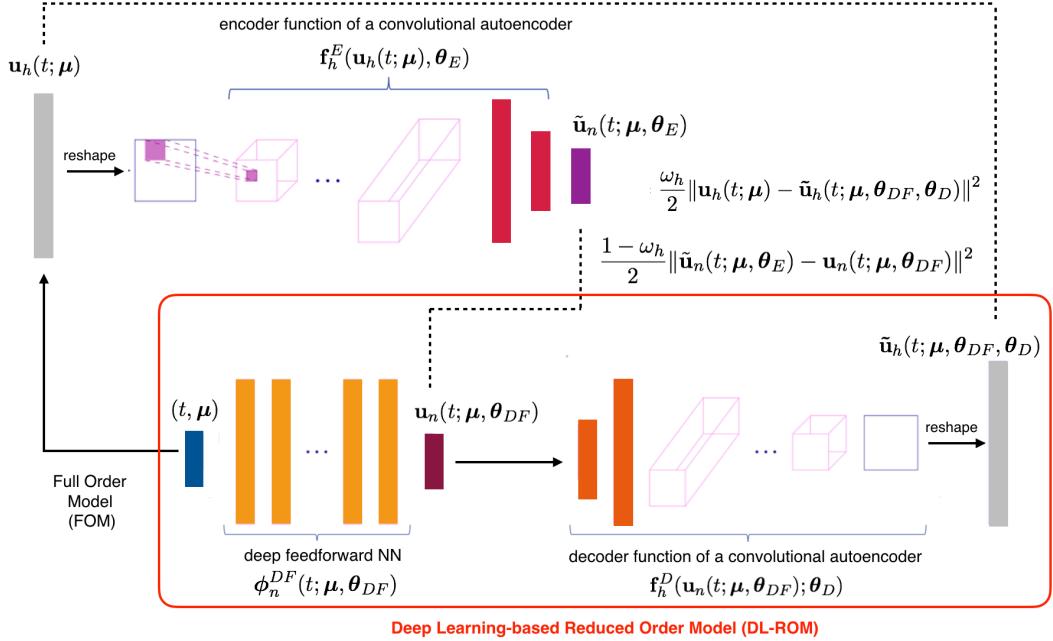


Figure 3: DL-ROM architecture (offline stage, training and validation).

the DL-ROM introduced above the encoder function of the convolutional autoencoder. This produces an additional term in the *per-example* loss function (17), thus calling the following optimization problem to be

solved:

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{N_s} \sum_{i=1}^{N_{train}} \sum_{k=1}^{N_t} \mathcal{L}(t^k, \boldsymbol{\mu}_i; \boldsymbol{\theta}), \quad (19)$$

where

$$\mathcal{L}(t^k, \boldsymbol{\mu}_i; \boldsymbol{\theta}) = \frac{\omega_h}{2} \|\mathbf{u}_h(t^k; \boldsymbol{\mu}_i) - \tilde{\mathbf{u}}_h(t^k; \boldsymbol{\mu}_i, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)\|^2 + \frac{1-\omega_h}{2} \|\tilde{\mathbf{u}}_n(t^k; \boldsymbol{\mu}_i, \boldsymbol{\theta}_E) - \mathbf{u}_n(t^k; \boldsymbol{\mu}_i, \boldsymbol{\theta}_{DF})\|^2 \quad (20)$$

and  $\boldsymbol{\theta} = (\boldsymbol{\theta}_E, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$ , with  $\omega_h \in [0, 1]$ . The *per-example* loss function (20) combines the reconstruction error (that is, the error between the FOM solution and the DL-ROM approximation) and the error between the intrinsic coordinates and the output of the encoder. This further term allows to enhance the performance of the DL-ROM, as shown in Test 3 of section 4.

### 3.1. Training and Testing Algorithms

Let us now detail the algorithms through which the training and testing phases of the networks are performed.

First of all, data normalization and standardization enhance the training phase of the network by rescaling all the values contained in the dataset to a common frame. For this reason, the inputs and the output of DL-ROM are normalized by applying an affine transformation in order to rescale them in the range  $[0, 1]$ . In particular, provided a training dataset  $X = [\mathbf{x}^1 | \mathbf{x}^2 | \dots | \mathbf{x}^{N_s}]^T \in \mathbb{R}^{N_s \times M}$ , we define

$$X_{max} = \max_{i \in \{1, \dots, N_s\}} \max_{j \in \{1, \dots, M\}} X_{ij} \quad \text{and} \quad X_{min} = \min_{i \in \{1, \dots, N_s\}} \min_{j \in \{1, \dots, M\}} X_{ij} \quad (21)$$

so that data are normalized by applying the following transformation

$$X \mapsto \frac{X - X_{min}}{X_{max} - X_{min}}. \quad (22)$$

Transformation (22) is applied also to the validation and testing sets, but considering as  $X_{max}$  and  $X_{min}$  the values computed over the training set. We point out that the input of the encoder function, the FOM solution  $\mathbf{u}_h = \mathbf{u}_h(t^k; \boldsymbol{\mu}_i)$  for a given (time, parameter) instance  $(t^k, \boldsymbol{\mu}_i)$ , is reshaped in a matrix. In particular, starting from  $\mathbf{u}_h \in \mathbb{R}^{N_h}$  we apply the transformation  $\mathbf{u}_h^R = \text{reshape}(\mathbf{u}_h)$  where  $\mathbf{u}_h^R \in \mathbb{R}^{N_h^{1/2} \times N_h^{1/2}}$ . If  $N_h$  is not a square, the input  $\mathbf{u}_h$  is zero-padded [43]. For the sake of simplicity, we continue to refer to the reshaped FOM solution to as  $\mathbf{u}_h$ . The inverse reshaping transformation is applied to the output of the last convolutional layer in the decoder function, the ROM approximation. Moreover, we highlight that applying one of the functions (12)-(13)-(18) to the matrix  $X$  means applying it row-wise.

The training algorithm referring to the architecture of DL-ROM depicted in Figure 3 is reported in Algorithm 1. During the training phase, the optimal parameters of the DL-ROM neural network are found by solving the optimization problem (19)-(20) through the back-propagation and ADAM algorithms.

At testing time, the encoder function is instead discarded, that is the DL-ROM architecture is the one shown in Figure 2 and the testing algorithm is the one pointed out in Algorithm 2. The testing phase corresponds to a forward step of the DL-ROM neural network in Figure 2.

## 4. Numerical results

In this section, we report the numerical results obtained by applying the proposed DL-ROM technique to three parametrized, time-dependent PDE problems, namely *(i)* Burgers equation, *(ii)* a linear transport equation, and *(iii)* a coupled PDE-ODE system arising from cardiac electrophysiology, namely the monodomain equation; this latter is a system of time dependent, nonlinear equations, whose solutions feature a traveling wave behavior. For the time being, we deal with problems set in  $d = 1$  (spatial) dimension featuring up to  $n_\mu = 2$  parameters; we will consider the extension to differential problems in  $d = 2$  and  $d = 3$

---

**Algorithm 1** DL-ROM training algorithm

---

**Input:** Parameter matrix  $M \in \mathbb{R}^{(n_\mu+1) \times N_s}$ , snapshot matrix  $S \in \mathbb{R}^{N_h \times N_s}$ , training-validation splitting fraction  $\alpha$ , starting learning rate  $\eta$ , batch size  $N_b$ , maximum number of epochs  $N_{\text{epochs}}$ , early stopping criterion, number of minibatches  $N_{\text{batch}} = (1 - \alpha)N_s/N_b$ .

**Output:** Optimal model parameters  $\boldsymbol{\theta}^* = (\boldsymbol{\theta}_E^*, \boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*)$ .

- 1: Randomly shuffle  $M$  and  $S$
- 2: Split data in  $M = [M^{\text{train}}, M^{\text{val}}]$  and  $S = [S^{\text{train}}, S^{\text{val}}]$  ( $M^{\text{val}}, S^{\text{val}} \in \mathbb{R}^{N_h \times \alpha N_s}$ )
- 3: Normalize data in  $M$  and  $S$  according to (22)
- 4: Randomly initialize  $\boldsymbol{\theta}^0 = (\boldsymbol{\theta}_E^0, \boldsymbol{\theta}_{DF}^0, \boldsymbol{\theta}_D^0)$
- 5:  $n_{\text{epochs}} = 0$
- 6: **while** ( $\neg$ early-stopping **and**  $n_{\text{epochs}} \leq N_{\text{epochs}}$ ) **do**
- 7:   **for**  $k = 1 : N_{\text{batch}}$  **do**
- 8:     Sample a minibatch  $(M^{\text{batch}}, S^{\text{batch}}) \subseteq (M^{\text{train}}, S^{\text{train}})$
- 9:      $S^{\text{batch}} = \text{reshape}(S^{\text{batch}})$
- 10:      $\tilde{S}_n^{\text{batch}}(\boldsymbol{\theta}_E^{N_{\text{batch}} n_{\text{epochs}} + k}) = \mathbf{f}_n^E(S^{\text{batch}}, \boldsymbol{\theta}_E^{N_{\text{batch}} n_{\text{epochs}} + k})$
- 11:      $S_n^{\text{batch}}(\boldsymbol{\theta}_{DF}^{N_{\text{batch}} n_{\text{epochs}} + k}) = \phi_n^{\text{DF}}(M^{\text{batch}}; \boldsymbol{\theta}_{DF}^{N_{\text{batch}} n_{\text{epochs}} + k})$
- 12:      $\tilde{S}_h^{\text{batch}}(\boldsymbol{\theta}_{DF}^{N_{\text{batch}} n_{\text{epochs}} + k}, \boldsymbol{\theta}_D^{N_{\text{batch}} n_{\text{epochs}} + k}) = \mathbf{f}_h^D(S_n^{\text{batch}}(\boldsymbol{\theta}_{DF}^{N_{\text{batch}} n_{\text{epochs}} + k}); \boldsymbol{\theta}_D^{N_{\text{batch}} n_{\text{epochs}} + k})$
- 13:      $\tilde{S}_h^{\text{batch}} = \text{reshape}(\tilde{S}_h^{\text{batch}})$
- 14:     Accumulate loss (20) on  $(M^{\text{batch}}, S^{\text{batch}})$  and compute  $\hat{\nabla}_{\boldsymbol{\theta}} \mathcal{J}$
- 15:      $\boldsymbol{\theta}^{N_{\text{batch}} n_{\text{epochs}} + k + 1} = \text{ADAM}(\eta, \hat{\nabla}_{\boldsymbol{\theta}} \mathcal{J}, \boldsymbol{\theta}^{N_{\text{batch}} n_{\text{epochs}} + k})$
- 16:   **end for**
- 17:   Repeat instructions 9-13 on  $(M^{\text{val}}, S^{\text{val}})$  with the updated weights  $\boldsymbol{\theta}^{N_{\text{batch}} n_{\text{epochs}} + k + 1}$
- 18:   Accumulate loss (20) on  $(M^{\text{val}}, S^{\text{val}})$  to evaluate early-stopping criterion
- 19:    $n_{\text{epochs}} = n_{\text{epochs}} + 1$
- 20: **end while**

---

**Algorithm 2** DL-ROM testing algorithm

---

**Input:** Testing parameter matrix  $M^{\text{test}} \in \mathbb{R}^{(n_\mu+1) \times (N_{\text{test}} N_t)}$  and optimal model parameters  $(\boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*)$ .

**Output:** ROM approximation matrix  $\tilde{S}_h \in \mathbb{R}^{N_h \times (N_{\text{test}} N_t)}$ .

- 1: Load  $\boldsymbol{\theta}_{DF}^*$  and  $\boldsymbol{\theta}_D^*$
  - 2:  $S_n(\boldsymbol{\theta}_{DF}^*) = \phi_n^{\text{DF}}(M^{\text{test}}; \boldsymbol{\theta}_{DF}^*)$
  - 3:  $\tilde{S}_h(\boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*) = \mathbf{f}_h^D(S_n(\boldsymbol{\theta}_{DF}^*); \boldsymbol{\theta}_D^*)$
  - 4:  $\tilde{S}_h = \text{reshape}(\tilde{S}_h)$
- 

in a forthcoming publication. For this reason, our focus is now on the numerical accuracy of our DL-ROM technique rather than on its computational efficiency and, therefore, on its comparison with linear ROMs such as the RB method featuring linear (possibly, piecewise linear) trial manifolds built through POD.

To evaluate the performance of DL-ROM we rely on the loss function (20) and on the following error indicator

$$\epsilon_{\text{rel}} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left( \frac{\sqrt{\sum_{k=1}^{N_t} \|\mathbf{u}_h^k(\boldsymbol{\mu}_{\text{test},i}) - \tilde{\mathbf{u}}_h^k(\boldsymbol{\mu}_{\text{test},i})\|^2}}{\sqrt{\sum_{k=1}^{N_t} \|\mathbf{u}_h^k(\boldsymbol{\mu}_{\text{test},i})\|^2}} \right). \quad (23)$$

We implement the neural network required by our DL-ROM technique by means of the Tensorflow deep learning framework [46] and the numerical simulations are performed on a workstation equipped with an Nvidia GeForce GTX 1070 8 GB GPU.

#### 4.1. Test 1: Burgers Equation

Let us consider the parametrized one-dimensional nonlinear Burgers equation

$$\begin{cases} \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - \frac{1}{\mu} \frac{\partial^2 u}{\partial x^2} = 0, & (x, t) \in (0, L) \times (0, T) \\ u(0, t) = 0, & t \in (0, T) \\ u(L, t) = 0, & t \in (0, T) \\ u(x, 0) = u_0(x), & x \in (0, L), \end{cases} \quad (24)$$

where

$$u_0(x) = \frac{x}{1 + \sqrt{1/A_0} \exp(\mu x^2/4)},$$

with  $A_0 = \exp(\mu/8)$ ,  $L = 1$  and  $T = 2$ . System (24) has been discretized in space by means of linear finite elements, with  $N_h = 256$  grid points, and in time by means of the Backward Euler scheme, with  $N_t = 100$  time instances. The parameter space, to which belongs the single ( $n_\mu = 1$ ) parameter, is given by  $\mathcal{P} = [100, 1000]$ . We consider  $N_{train} = 20$  training-parameter instances uniformly distributed over  $\mathcal{P}$  and  $N_{test} = 19$  testing-parameter instances, each of them corresponding to the midpoint between two consecutive training-parameter instances.

The configuration of the DL-ROM neural network used for this test case is the following. We choose a 12-layers DFNN equipped with 50 neurons per hidden layer and  $n$  neurons in the output layer, where  $n$  corresponds to the dimension of the reduced trial manifold. The architectures of the encoder and decoder functions are instead reported in Tables 1 and 2, and are similar to the ones used in [17].

Layer	Input Dimension	Output Dimension	Kernel Size	# of Filters	Stride	Padding
1			[5, 5]	8	1	SAME
2			[5, 5]	16	2	SAME
3			[5, 5]	32	2	SAME
4			[5, 5]	64	2	SAME
5	$N_h$	256				
6	256	$n$				

Table 1: *Test 1*: Attributes of convolutional layers and dense layers in the encoder  $\mathbf{f}_n^E$ .

Layer	Input dimension	Output dimension	Kernel size	# of filters	Stride	Padding
1	$n$	256				
2	256	$N_h$				
3			[5, 5]	64	2	SAME
4			[5, 5]	32	2	SAME
5			[5, 5]	16	2	SAME
6			[5, 5]	1	1	SAME

Table 2: *Test 1*: Attributes of dense layers and transposed convolutional layers in the decoder  $\mathbf{f}_h^D$ .

Problem (24) does not represent a remarkably challenging task for linear ROM, indeed by considering for example POD and by applying it to the snapshot matrix (the latter built by collecting the solution of (24) for  $N_s = N_{train}N_t$  training-parameter instances) it is sufficient to assemble a linear trial manifold of dimension 20 in order to capture more than the 99.99% of the energy of the system [12, 4]. In order to assess the performance of our DL-ROM technique, we compute the DL-ROM solution by fixing the dimension of the nonlinear trial manifold to  $n = 20$ . In Figure 4 we show the DL-ROM and the optimal-POD reconstructions, along with the FOM solution, for the time instance  $t = 0.02$  and for the testing-parameter instance  $\mu_{test} = 976.32$ , the testing value of  $\mu$  for which the reconstruction task results to be the most

difficult both for POD and DL-ROM, being the diffusion term in (24) smaller and the solution closer to the one of a purely hyperbolic system. In particular, for  $\mu_{test} = 976.32$ , employing the DL-based ROM technique presented in this work allows us to halve the error indicator  $\epsilon_{rel}$  associated to the optimal-POD approximation of the FOM solution. Referring to Figure 4, the DL-ROM reconstruction is more accurate than the optimal-POD one, indeed it mostly fits the FOM solution, even in correspondence of its maximum, as shown in the zooms of Figure 4. Moreover, it does not introduce oscillations where a large gradient of the FOM solution is observed, as it happens instead by employing POD.

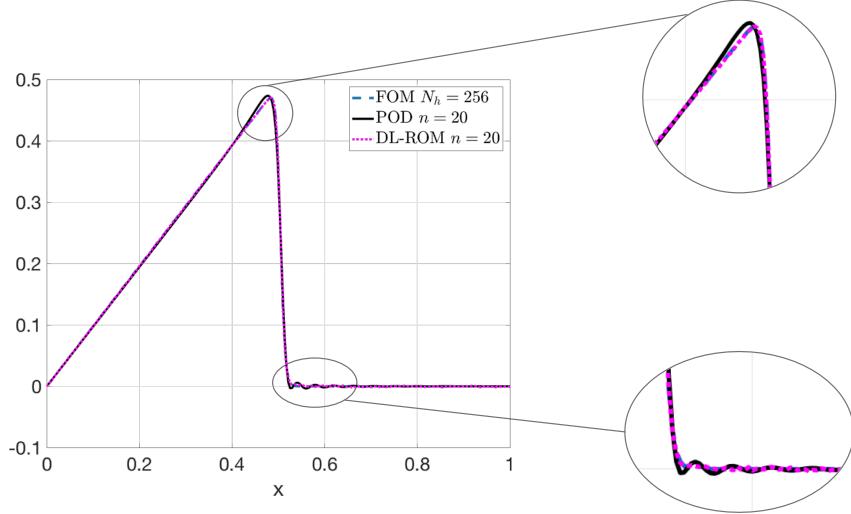


Figure 4: *Test 1*: FOM, optimal-POD and DL-ROM solutions for the testing-parameter instance  $\mu_{test} = 976.32$  at  $t = 0.02$ , with  $n = 20$ .

In Figure 5 we show the same comparison of Figure 4 but this time considering both for POD and DL-ROM a reduced dimension  $n = 10$ . The difference in terms of accuracy provided by the two approaches is even more striking in this case.

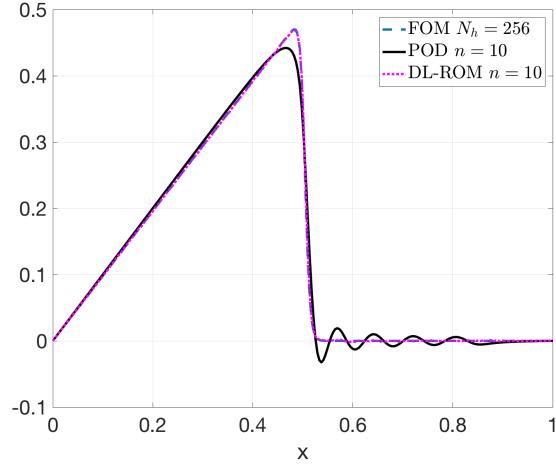


Figure 5: *Test 1*: FOM, optimal-POD and DL-ROM solutions for the testing-parameter instance  $\mu_{test} = 976.32$  at  $t = 0.02$ , with  $n = 10$ .

Finally, in Figure 6 we highlight the accuracy properties of both the DL-ROM and POD techniques by displaying the behavior of the error indicator  $\epsilon_{rel}$ , defined in (23), with respect to the dimension  $n$  of the corresponding reduced trial manifold. For  $n < 20$  the DL-ROM approximation is more accurate than the

one provided by POD, and only for  $n = 20$  the two techniques provide almost the same accuracy.

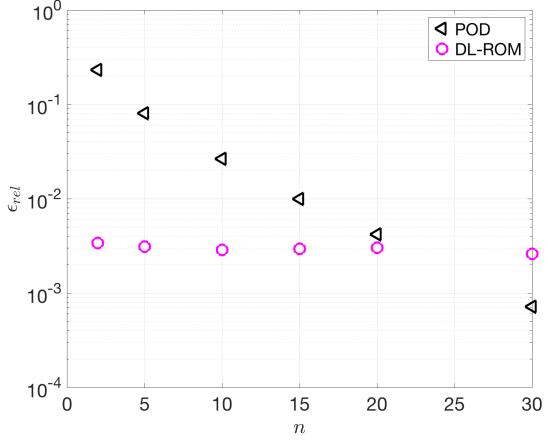


Figure 6: *Test 1*: Error indicator  $\epsilon_{rel}$  vs.  $n$  on the testing set.

#### 4.2. Test 2: Linear Transport Equation

We consider two tests for this set of parametrized differential models.

##### Test 2.1: $n_\mu = 1$

First, we consider the parametrized one-dimensional linear transport equation

$$\begin{cases} \frac{\partial u}{\partial t} + \mu \frac{\partial u}{\partial x} = 0, & (x, t) \in \mathbb{R} \times (0, T) \\ u(x, 0) = u_0(x), & x \in \mathbb{R}, \end{cases} \quad (25)$$

whose exact solution is  $u(x, t) = u_0(x - \mu t)$ . We set  $u_0(x) = (1/\sqrt{2\pi\sigma})e^{-x^2/2\sigma}$  and  $T = 1$ .

The parameter (here  $n_\mu = 1$ ) represents the velocity of the travelling wave and the parameter space is given by  $\mathcal{P} = [0.775, 1.25]$ . The dataset is built by uniformly sampling the exact solution in the domain  $(0, L) \times (0, T)$ , with  $L = 1$ , and by considering  $N_h = 256$  degrees of freedom in the space discretization and  $N_t = 200$  time instances in the time one. We consider  $N_{train} = 20$  training-parameter instances uniformly distributed in the parameter space  $\mathcal{P}$  and  $N_{test} = 19$  testing-parameter instances such that  $\mu_{test,i} = (\mu_{train,i} + \mu_{train,i+1})/2$ , for  $i = 1, \dots, N_{test}$ . This test case, and more in general hyperbolic problems, are examples in which the use of a linear approach to ROM generally yields poor performance in terms of accuracy. Indeed, the dimension of the linear trial manifold must be very large, if compared to the dimension of the solution manifold, in order to capture the variability of the FOM solution over the parameter space  $\mathcal{P}$ . We set  $\sigma = 10^{-4}$  in order to assess the performance of DL-ROM in a scenario which is still remarkably challenging for ROM on linear trial manifolds.

Figure 7 shows the exact solution, which here plays the role of the FOM solution, and the DL-ROM one for the testing-parameter instance  $\mu_{test} = 0.8625$ ; here, we set the dimension of the nonlinear trial manifold to  $n = 2$ , equal to the dimension of the solution manifold  $n_\mu + 1$ . Moreover, in Figure 7 we highlight the relative error  $\epsilon_k \in \mathbb{R}^{N_h}$ , for  $k = 1, \dots, N_t$ , associated to a given  $\boldsymbol{\mu}_{test} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}$  (in this case  $n_\mu = 1$ ), defined as

$$\epsilon_k = \frac{|\mathbf{u}_h^k(\boldsymbol{\mu}_{test}) - \tilde{\mathbf{u}}_h^k(\boldsymbol{\mu}_{test})|}{\sqrt{\sum_{k=1}^{N_t} \|\mathbf{u}_h^k(\boldsymbol{\mu}_{test})\|^2}}, \quad (26)$$

which widens in proximity of the spike of the exact solution.

In Figure 8 we report the exact solution and the DL-ROM one, obtained by setting  $n = 2$ , for three particular time instances. In order to compare the performance of the proposed nonlinear ROM with a linear

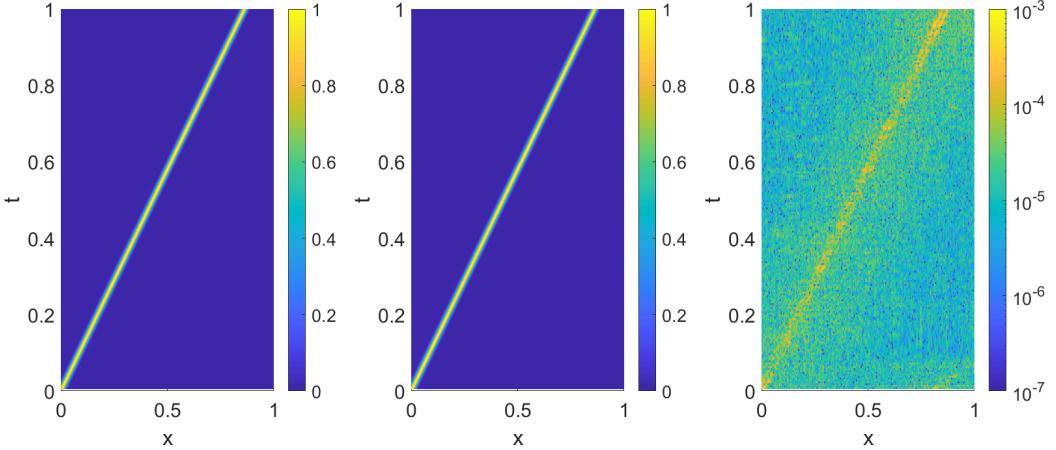


Figure 7: *Test 2.1*: Exact solution (left), DL-ROM solution with  $n = 2$  (center) and relative error  $\epsilon_k$  (right) for the testing-parameter instance  $\mu_{test} = 0.8625$  in the space-time domain.

approach, we perform the POD on the snapshot matrix and show, for the same testing-parameter instance, the optimal POD-reconstruction, i.e. the projection of the FOM (exact) solution onto the POD basis, in Figure 8. For example, by considering  $n = 2$ , the error indicator, defined in (23), is  $\epsilon_{rel} = 8.74 \cdot 10^{-3}$ . By considering a linear ROM technique instead, even by considering a reduced trial manifold of dimension  $n = 50$ , built by means of the POD, the reconstructed solution presents spurious oscillations which result in a poor approximation of the FOM solution (see Figure 8). Indeed, in order to achieve the same accuracy obtained through DL-ROM over the testing set one has to select 90 basis functions, i.e. a linear trial manifold of dimension  $n = 90$ .

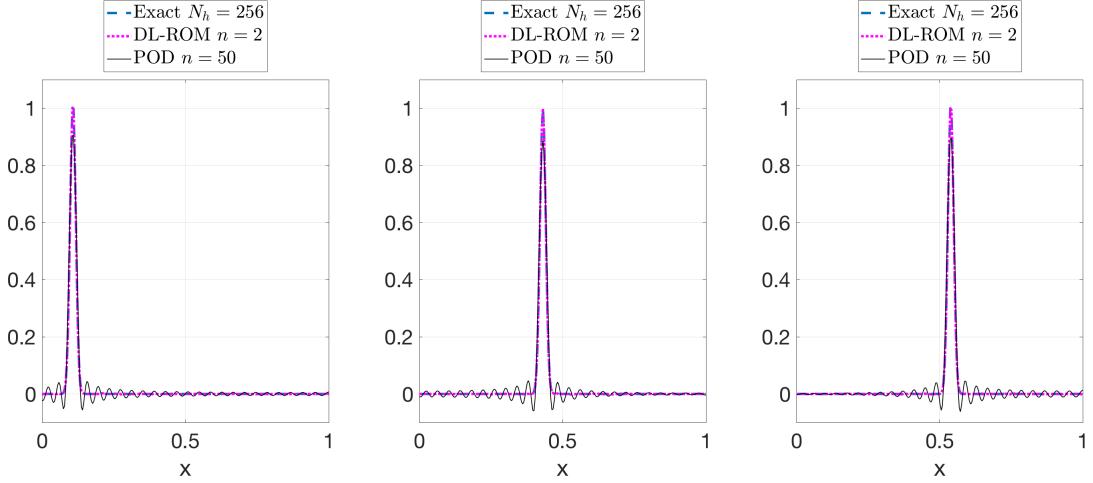


Figure 8: *Test 2.1*: Exact, DL-ROM and POD solutions for the testing-parameter instance  $\mu_{test} = 0.8625$  at  $t = 0.125, 0.5$  and  $0.625$ .

Figure 9 shows the behavior of the error indicator (23) with respect to the reduced dimension  $n$ . By increasing the dimension of the nonlinear trial manifold there is a slight improvement of the performance of the DL-ROM neural network, i.e. the error indicator decreases. This improvement is not particularly relevant because by increasing  $n$ , the number of parameters of the DL-ROM neural network, i.e. weights and biases, is increased by a limited quantity. In this way the approximation capability of the neural network remains almost the same and so does the error indicator (23).

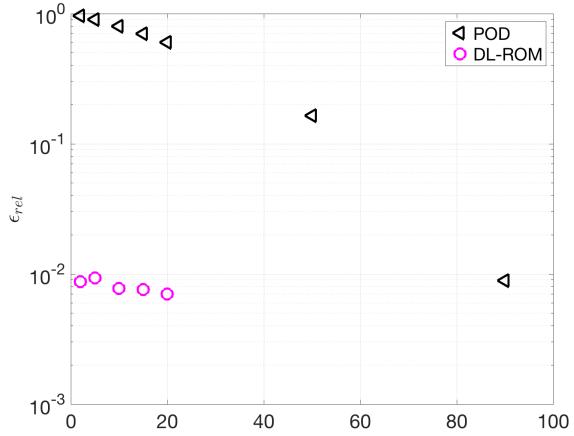


Figure 9: *Test 2.1:* Error indicator  $\epsilon_{rel}$  vs.  $n$  on the testing set.

**Remark 1.** (*Hyperparameters Tuning*). The hyperparameters of the DL-ROM neural network are tuned by evaluating the loss function over the validation set and by setting each of them equal to the value minimizing the generalization error on the validation set. In particular, we show the tests performed to choose the size of the (transposed) convolutional kernels in the (decoder) encoder function, the number of hidden layers in the feedforward neural network and the number of neurons for each hidden layer. The hyperparameters evaluation starts from the default configuration in Table 3.

Kernel Size	# Hidden Layers	# Neurons
[3, 3]	1	50

Table 3: *Test 2.1:* Starting configuration of DL-ROM.

Then, the best values are found iteratively by studying the impact of the variation of a single hyperparameter at a time on the validation loss. Once the best value of a hyperparameter is found, this value replaces the default value from that point on. For each hyperparameter the tuning is performed in a range of values for which the training of the network is affordable regarding computational costs. In Figure 10, we show the

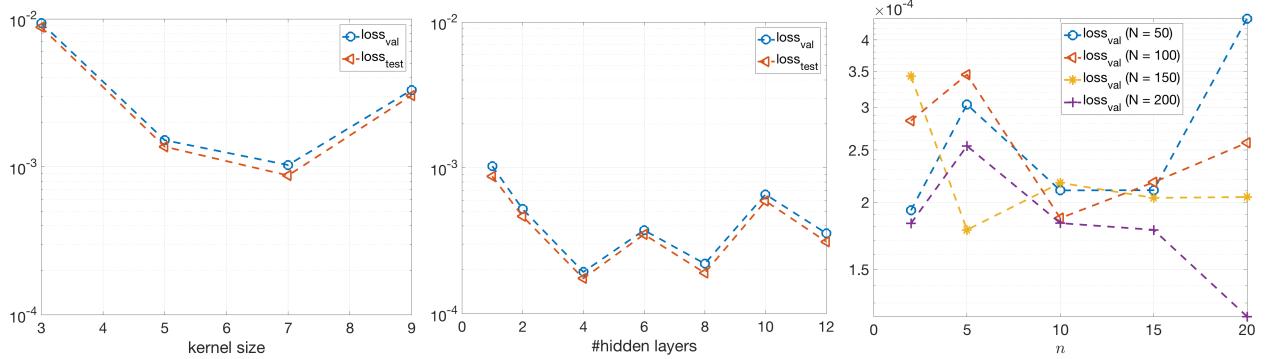


Figure 10: *Test 2.1:* Impact of the kernel size (left), the number of hidden layers (center) and the number of neurons (right) on the validation and testing loss.

impact of the size of the convolutional kernels on the loss over the validation and testing sets, the number of hidden layers in the feedforward forward neural network and the number of neurons in each hidden layer by varying the reduced dimension in order to find the best value of such hyperparameter over  $n$ . The final configuration of the DL-ROM neural network is the one provided in Table 4.

Kernel Size	# Hidden Layers	# Neurons
[7, 7]	4	200

Table 4: *Test 2.1*: Final configuration of DL-ROM.

*Test 2.2:*  $n_\mu = 2$

Here we consider again the parametrized one-dimensional transport equation

$$\begin{cases} \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, & (x, t) \in \mathbb{R} \times (0, T) \\ u(x, 0) = u_0(x), & x \in \mathbb{R}. \end{cases} \quad (27)$$

The exact solution of (27) is  $u(x, t) = u_0(x - t; \boldsymbol{\mu})$  but this time we set the initial datum equal to

$$u_0(x; \boldsymbol{\mu}) = \begin{cases} 0, & \text{if } x < \mu_1 \\ \mu_2, & \text{if } x \geq \mu_1, \end{cases} \quad (28)$$

where  $\boldsymbol{\mu} = [\mu_1, \mu_2]^T$ . The  $n_\mu = 2$  parameters belong to the parameter space  $\mathcal{P} = \mathcal{P}_{\mu_1} \times \mathcal{P}_{\mu_2} = [0.025, 0.25] \times [0.5, 1]$ . We build the dataset by uniformly sampling the exact solution in the domain  $(0, L) \times (0, T)$ , with  $L = 1$  and  $T = 1$ , and by considering  $N_h = 256$  grid points for the space discretization and  $N_t = 100$  time instances for the time one. We collect, both for  $\mu_1$  and  $\mu_2$ ,  $N_{train} = 21$  training-parameter instances uniformly distributed in the parameter space  $\mathcal{P}$  and  $N_{test} = 20$  testing-parameter instances, selected as in the other test cases. Equation (27), completed with the initial datum (28), stands as one of the most challenging problems for linear ROM techniques because of the difficulty to accurately reconstruct the jump discontinuity of the exact solution as a linear combination of basis functions computed from the snapshots, for a testing-parameter instance. The architecture of the DL-ROM neural network used here is the one presented in the Test 2.1.

In Figure 11 we show the exact solution, which here again plays the role of the FOM solution, and the DL-ROM one, obtained by setting  $n = 3$ , equal to the dimension of the solution manifold  $n_\mu + 1$ , for the testing-parameter instance  $\boldsymbol{\mu}_{test} = (0.154375, 0.6375)$ , along with the relative error  $\epsilon_k$ , defined in (26), which is larger near the jump of the FOM solution.

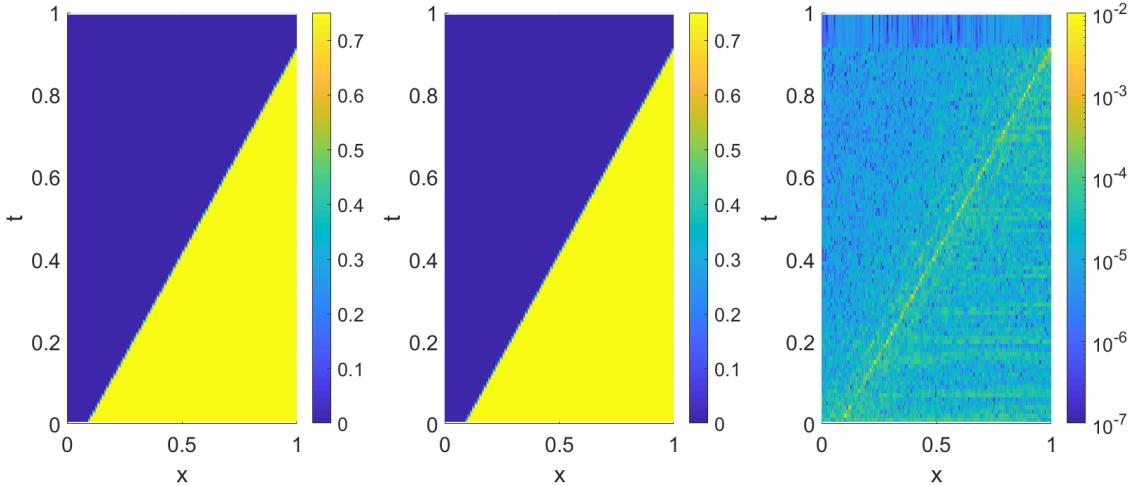


Figure 11: *Test 2.2*: Exact solution (left), DL-ROM solution with  $n = 3$  (center) and relative error  $\epsilon_k$  (right) for the testing-parameter instance  $\boldsymbol{\mu}_{test} = (0.154375, 0.6375)$  in the space-time domain.

In Figure 12 we report the DL-ROM and optimal-POD reconstructions, together with the FOM solution, for the time instances  $t = 0.245, 0.495$  and  $0.745$ , and the testing-parameter instance  $\boldsymbol{\mu}_{test} =$

$(0.154375, 0.6375)$ . The dimension of the reduced manifolds are  $n = 3$  and  $n = 50$  for the DL-ROM and POD techniques, respectively. By considering a linear ROM technique, even by setting the dimension of the reduced manifold equal to  $n = 50$ , the reconstructed solution presents spurious oscillations which lead to a poor approximation of the FOM solution. Moreover, the optimal-POD solution is not able to fit the discontinuity of the FOM solution in a sharp way. These oscillations are significantly mitigated by the use of our DL-ROM and the jump discontinuity is accurately fit by the DL-ROM solution, as shown in Figure 12.

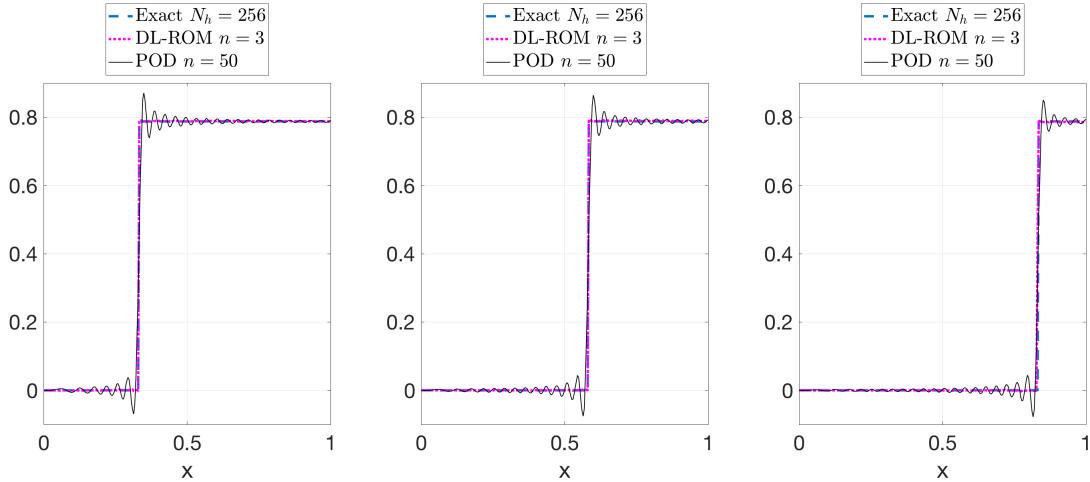


Figure 12: *Test 2.2:* Exact, DL-ROM and optimal-POD solutions for the testing-parameter instance  $\mu_{test} = (0.154375, 0.6375)$  at  $t = 0.245, 0.495$  and  $0.745$ .

Finally, in Figure 13 we highlight the accuracy properties of both the DL-ROM and POD techniques. In particular, the same conclusions observed in Test 2.1, namely those regarding the behaviour of the error indicator (23) with respect to the reduced dimension  $n$ , still hold. The developed DL-ROM technique allows us to obtain a value for the error indicator equal to  $\epsilon_{rel} = 2.85 \cdot 10^{-2}$  with  $n = 3$ , which instead is achieved by POD only by selecting 165 basis functions, i.e. by building a linear trial manifold of dimension  $n = 165$ .

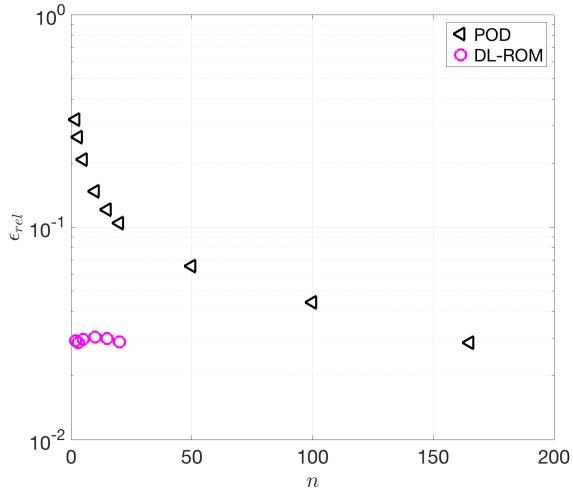


Figure 13: *Test 2.2:* Error indicator  $\epsilon_{rel}$  vs.  $n$  on the testing set.

#### 4.3. Test 3: Monodomain Equation

We now consider the following one-dimensional coupled PDE-ODE nonlinear system

$$\begin{cases} \mu \frac{\partial u}{\partial t} - \mu^2 \frac{\partial^2 u}{\partial x^2} + u(u-0.1)(u-1) + w = 0, & (x,t) \in (0,L) \times (0,T) \\ \frac{dw}{dt} + (\gamma w - \beta u) = 0, & (x,t) \in (0,L) \times (0,T) \\ \frac{\partial u}{\partial x}(0,t) = 50000t^3 e^{-15t}, & t \in (0,T) \\ \frac{\partial u}{\partial x}(L,t) = 0, & t \in (0,T) \\ u(x,0) = 0, w(x,0) = 0, & x \in (0,L), \end{cases} \quad (29)$$

where  $L = 1$ ,  $T = 2$ ,  $\gamma = 2$  and  $\beta = 0.5$ . The parameter  $\mu$  ( $n_\mu = 1$ ) belongs to the parameter space  $\mathcal{P} = 5 \cdot [10^{-3}, 10^{-2}]$ . This system consists in a parametrized version of the Monodomain equation coupled with the FitzHugh-Nagumo cellular model which describes the excitation-relaxation of the cell membrane in the cardiac tissue [47, 48]. In such a model, the ionic current is a cubic function of the electrical potential  $v$  and linear in the recovery variable  $w$ . Eqs (29) have been discretized in space through linear finite elements by considering  $N_h = 256$  grid points. We use a one-step, semi-implicit, first order scheme similar to the one discussed in [6] for time discretization and the treatment of the nonlinear term<sup>1</sup>. The solution of the former problem consists in a parameter-depending travelling wave, which exhibits sharper and sharper fronts as the parameter  $\mu$  gets smaller (see Figure 14). We consider  $N_{train} = 20$  training-parameter instances uniformly

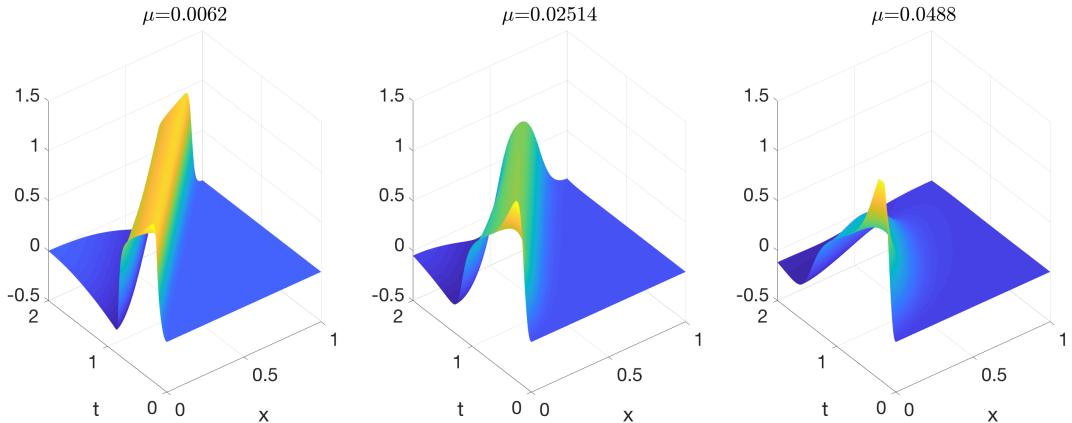


Figure 14: *Test 3*: FOM solutions for different testing-parameter instances.

distributed in the parameter space  $\mathcal{P}$  and  $N_{test} = 19$  testing-parameter instances, each of them corresponding to the midpoint between two consecutive training parameter instances.

Figure 15 shows the FOM solution and the DL-ROM one obtained by setting  $n = 2$ , the dimension of the solution manifold, for the testing-parameter instance  $\mu_{test} = 0.0062$ . We also report in Figure 15 the error indicator  $\epsilon_k$  (26), which is higher in correspondence of the large gradients of the FOM solution.

The accuracy obtained by our DL-ROM technique, with  $n = 2$ , on the testing set is  $\epsilon_{rel} = 3.42 \cdot 10^{-3}$ . In order to assess the performance of DL-ROM with respect to a linear ROM technique we point out in Table 5 the maximum number of basis functions among all the clusters, i.e. the dimension of the largest linear trial manifold, required by the (local) RB method in order to achieve the same accuracy obtained through

<sup>1</sup>The Matlab library used to compute snapshots and the numerical results regarding the (local) RB method for problem (29) is freely available at <https://github.com/StefanoPagani/LocalROM>

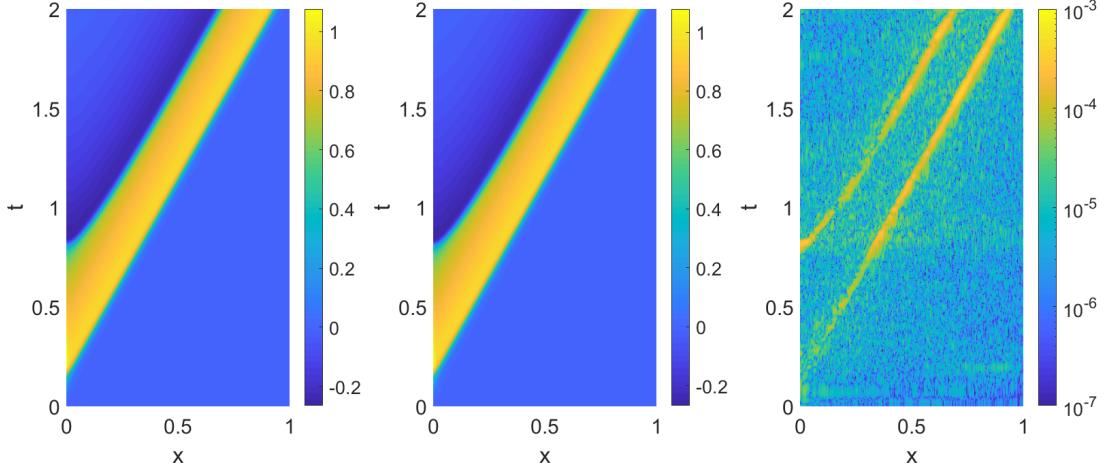


Figure 15: *Test 3*: FOM solution (left), DL-ROM solution with  $n = 2$  (center) and error indicator  $\epsilon_k$  (right) for the testing-parameter instance  $\mu_{test} = 0.0062$  in the space-time domain.

DL-ROM. By increasing the number of clusters, the dimension of the largest linear trial subspace decreases; this does not hold as long as the number of clusters is larger than  $k = 32$ . Indeed, the dimension of some linear subspaces become so small that the error increases with respect the one obtained with fewer clusters. In particular, in Figure 16 the RB solutions obtained by considering  $n = 2$  and  $n = 66$  basis functions are shown. In Figure 17 we compare the FOM solution with the DL-ROM one, obtained for  $n = 2$ , and the FOM solution with the RB one by setting  $n = 2, 20$  and  $66$ , for  $\mu_{test} = 0.0157$  at  $t = 0.4962, 0.9975$  and  $1.4987$ .

$k = 1$	$k = 2$	$k = 4$	$k = 8$	$k = 16$	$k = 32$
66	68	55	34	26	20

Table 5: *Test 3*: Maximum number of basis functions for the (local) RB method.

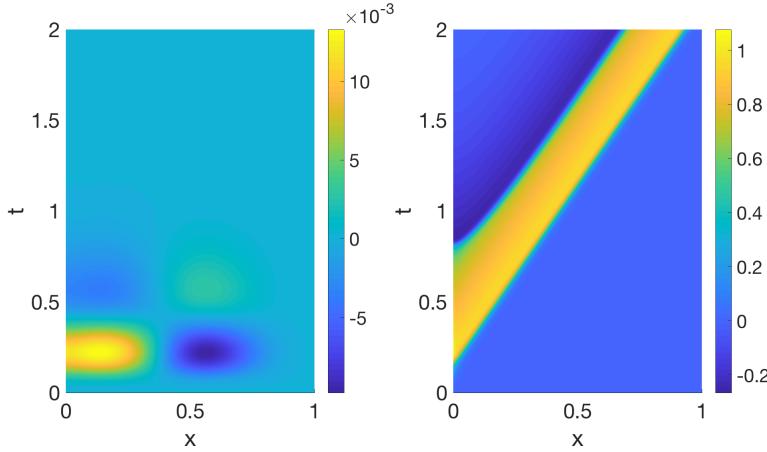


Figure 16: *Test 3*: RB solutions for the testing parameter instance  $\mu_{test} = 0.0062$  with  $n = 2$  (left) and  $n = 66$  (right).

The convergence of the error indicator (23) as a function of the reduced dimension  $n$  is shown in Figure 18. For the (local) RB method, by increasing the dimension of the largest linear trial manifold, the error indicator decreases, this occurs also by applying the DL-ROM technique for  $n \leq 20$ . The decay of the error indicator

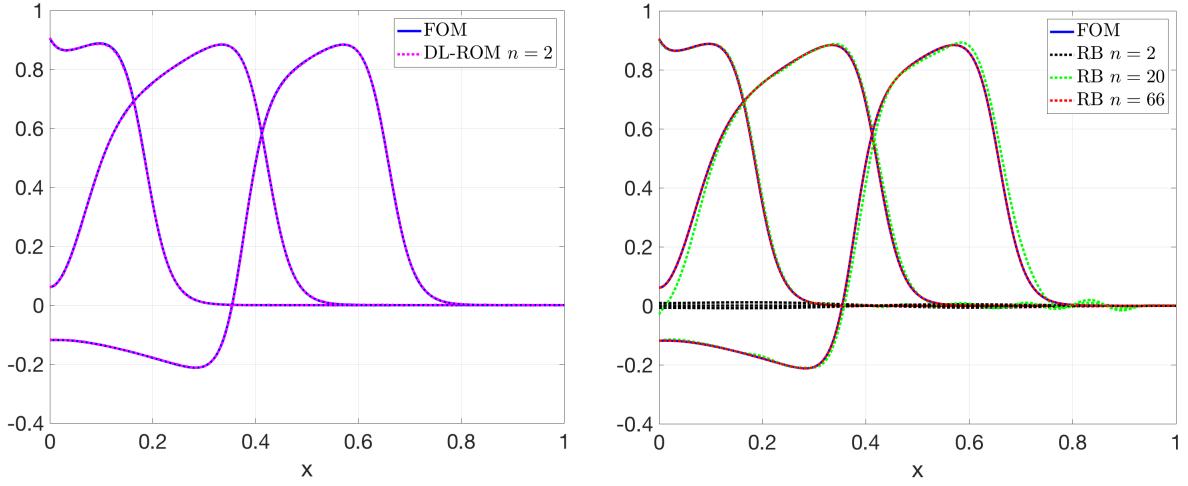


Figure 17: *Test 3*: FOM and DL-ROM solutions (left) and FOM and RB solutions (right) for the testing-parameter instance  $\mu_{test} = 0.0157$  at  $t = 0.4962, 0.9975$  and  $1.4987$ .

in the latter case is not so remarkable for the same reason pointed out in Test 2.1. If we consider larger values of  $n$ , e.g.  $n = 40$ , overfitting occurs, meaning that the neural network model is too complex with respect to the amount of data provided it. For this reason, by considering, for example  $n = 40$ , the error indicator  $\epsilon_{rel}$  increases.

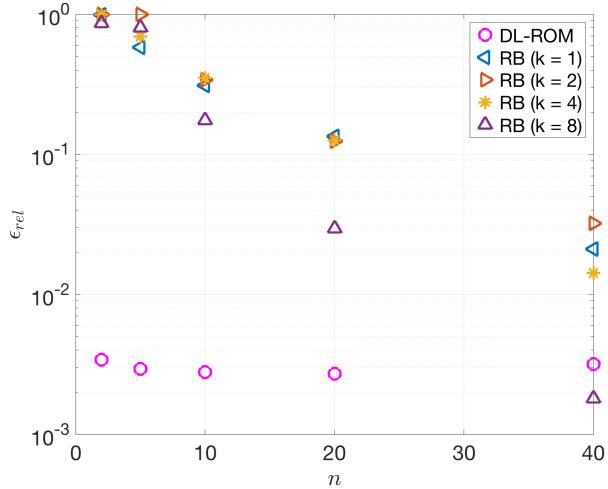


Figure 18: *Test 3*: Error indicator  $\epsilon_{rel}$  vs.  $n$  on the testing set.

Finally, in Figure 19 we report the behavior of the loss function and of the error indicator (23) with respect to the number of training-parameter instances, i.e. the size of the training dataset. By providing more data to the DL-ROM neural network, its approximation capability increases, thus yielding a decrease in the generalization error and the error indicator. In particular, the decay of the loss function with respect to the number of training-parameter instances  $N_{train}$  is approximately proportional to  $1/N_{train}^3$  and the one of the error indicator (23) is about  $1/N_{train}^2$ .

**Remark 2.** (*Hyperparameters Tuning*). *In order to perform hyperparameters tuning we follow the same procedure used for Test 2.1. We start from the default configuration and we tune the size of the (transposed) convolutional kernels in the (decoder) encoder function, the number of hidden layers in the feedforward neural*

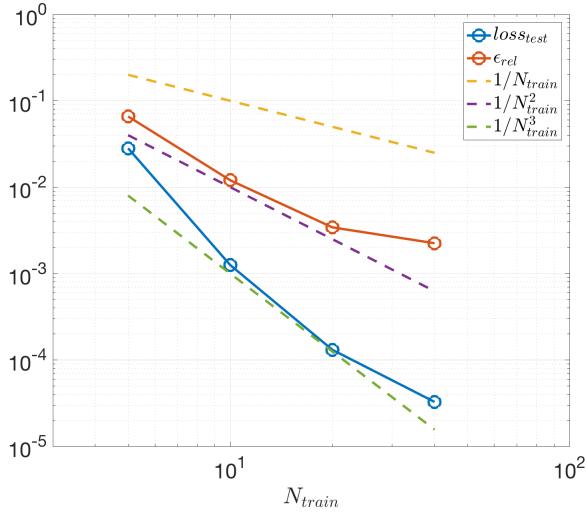


Figure 19: *Test 3*: Loss and error indicator  $\epsilon_{rel}$  on the testing set vs. number of training-parameter instances of the parameter  $\mu$ .

*network and the number of neurons for each hidden layer. In Figure 20 we show the impact of the different hyperparameters on the validation and testing losses. The final configuration of the DL-ROM neural network is the one provided in Table 6.*

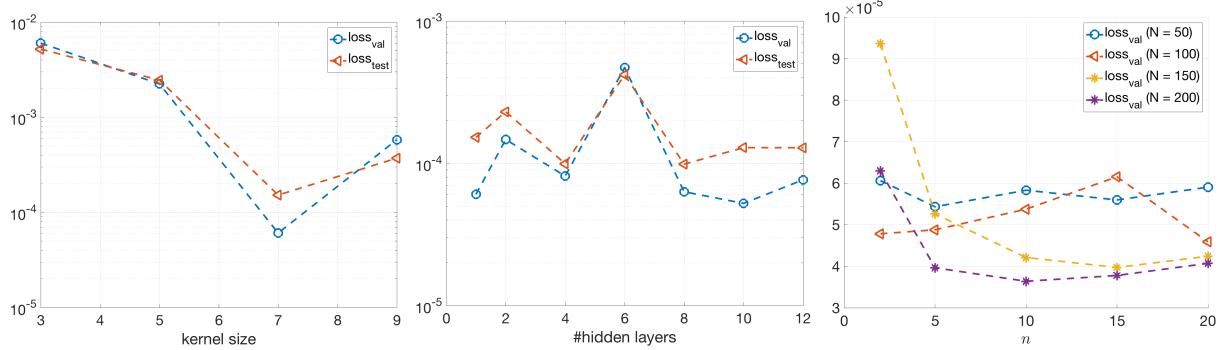


Figure 20: *Test 3*: Impact of the kernel size (left), the number of hidden layers (center) and the number of neurons (right) on the validation and testing loss.

Kernel Size	# Hidden Layers	# Neurons
[7, 7]	1	200

Table 6: *Test 3*: Final configuration of DL-ROM.

**Remark 3.** (*Sensitivity with respect to the weight  $\omega_h$* ). For all the test cases analyzed we set the parameter  $\omega_h$  in the loss function (20) equal to  $\omega_h = 1/2$ . In order to justify this choice we performed a sensitivity analysis for problem (29) as shown in Figure 21. For extreme values of  $\omega_h$ , the error indicator (23) worsens of about one order of magnitude. In particular, not considering the encoder function  $\mathbf{f}_n^E$ , that corresponds to the case  $\omega_h = 1$ , yields worse performance of the DL-ROM neural network, as highlighted in Figure 21. Similarly, by taking  $\omega_h = 0$ , we would neglect the reconstruction error (that is, the first term in the per-example loss function (20)); this is why the error indicator is large for  $\omega_h = 0.1$ . All the values of  $\omega_h$  in the

range  $[0.2, 0.9]$  do not yield significant differences in terms of error indicator, so we decided to set  $\omega_h = 1/2$  – and, as a matter of fact,  $1 - \omega_h = 1/2$ .

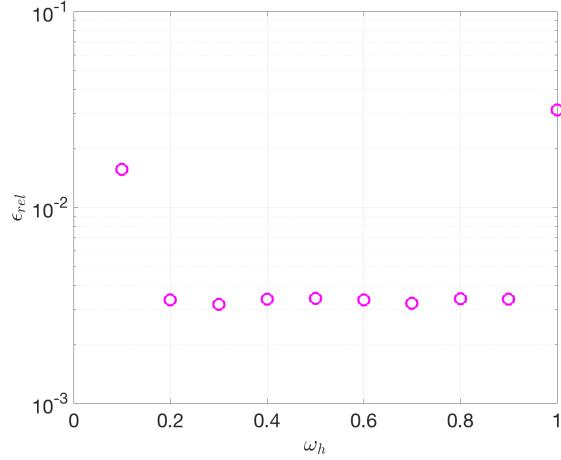


Figure 21: *Test 3*: Error indicator  $\epsilon_{rel}$  vs.  $\omega_h$ .

## 5. Conclusions

In this work we have proposed a novel technique to build low-dimensional ROMs exploiting deep learning models in order to overcome the usual computational bottlenecks shown by classical, linear projection-based ROM techniques (such as the reduced basis method relying on proper orthogonal decomposition) when dealing with problems featuring coherent structures that propagate over time, such as transport and wave-type phenomena, or convection-dominated flows.

The proposed Deep Learning-based Reduced Order Model (DL-ROM) allows to approximate both the solution manifold of a given parametrized nonlinear, time-dependent PDE by means of a low-dimensional, nonlinear trial manifold, and the nonlinear dynamics of the generalized coordinates on such reduced trial manifold, as a function of the time coordinate and the parameters. Both *(i)* the nonlinear trial manifold and *(ii)* the reduced dynamics are learnt in a non-intrusive way, thus avoiding to query the arrays related to the FOM; the former is learnt by means of the decoder function of a convolutional autoencoder neural network, whereas the latter through a (deep) feedforward neural network, and the encoder function of the convolutional autoencoder.

The numerical results obtained for three different test cases show that the proposed DL-ROM technique provides sufficiently accurate solutions to the parametrized PDEs involving a low-dimensional solution manifold whose dimension is  $n_\mu + 1$ . The proposed DL-ROM outperforms linear ROMs such as the RB method (relying on a global POD basis), as well as nonlinear approaches exploiting local POD bases, when applied both to *(i)* problems which are extremely challenging for linear ROMs, such as the linear transport equation or nonlinear diffusion-reaction PDEs coupled to ODEs, and *(ii)* problems which are more tractable using a linear ROM, like Burgers equation, however featuring POD bases with much higher dimension.

Regarding numerical accuracy, the proposed DL-ROM technique provides approximations that are orders of magnitude more accurate than the ones provided by linear ROMs, when keeping the same dimension. We do not obtain remarkable error decays when considering low-dimensional spaces of increasing dimensions, thus making the accuracy of both approximations comparable when dealing with  $\mathcal{O}(10^2)$  POD basis functions – a dimension which makes however linear ROMs infeasible when moving to more involved parametrized problems in higher space dimensions. Regarding computational efficiency, we deem not appropriate to perform comparisons with one-dimensional test cases (on meshes featuring no more than  $\mathcal{O}(10^3)$  degrees of freedom). We will perform the assessment of the computational speedup of our DL-ROM technique

compared to linear ROMs in future publications; we expect however to obtain remarkable computational gains when dealing with two and three-dimensional problems for which linear ROMs are not well-suited to approximate the solution to parametrized, nonlinear time-dependent PDEs. Numerical results shown that DL-ROM allows to generate approximation spaces of dimension close to the intrinsic dimension of the solution manifold, by providing also remarkably improvements in terms of efficiency, will be published in a forthcoming paper.

### Acknowledgments

We gratefully acknowledge Prof. A. Quarteroni (MOX, Politecnico di Milano) for his stimulating discussions, Dr. S. Pagani (MOX, Politecnico di Milano) for his useful remarks and M. Salvador (MOX, Politecnico di Milano) for kindly sharing the code implementing the FOM of Test 1.

### Appendix A. Basic concepts of deep learning

Deep learning (DL) techniques have gained great attention in recent years in several areas like computer vision [49, 50], natural language processing [51, 52] and speech recognition [53, 54], due to their ability to discover pattern and extract features from massive datasets, in order to make predictions without providing hand-crafted features. In this section we provide an overview of those deep-learning models which the proposed DL-ROM technique presented in this work relies on.

#### Appendix A.1. Deep feedforward neural network

A remarkable example of DL model is the deep feedforward neural network (DFNN). A DFNN is a mathematical function modeling the relationship between a set of input values and some output values [43]. This mathematical function is obtained through composition of simpler (nonlinear) functions, or layers, and allows to learn complex hierarchies of features. More formally, provided an input  $\mathbf{x} \in \mathbb{R}^{N_0}$  a DFNN with  $L$  layers takes the form

$$\phi^{DF} : (\mathbf{x}; \boldsymbol{\theta}_{DF}) \mapsto \phi_L(\cdot; \boldsymbol{\theta}_L) \circ \phi_{L-1}(\cdot; \boldsymbol{\theta}_{L-1}) \circ \dots \circ \phi_1(\mathbf{x}; \boldsymbol{\theta}_1), \quad (\text{A.1})$$

where  $\phi_i(\cdot; \boldsymbol{\theta}_i) : \mathbb{R}^{N_{i-1}} \mapsto \mathbb{R}^{N_i}$ ,  $i = 1, \dots, L$ , refers to the activation function applied at layer  $i$  of the DFNN and  $\boldsymbol{\theta}_i = (W_i, \mathbf{b}_i)$ , with  $W_i \in \mathbb{R}^{N_i \times N_{i-1}}$  and  $\mathbf{b}_i \in \mathbb{R}^{N_i}$ ,  $i = 1, \dots, L$ , are the weights and the bias of layer  $i$  such that  $\boldsymbol{\theta}_{DF} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_L)$ . We usually refer to the collection of all weights and biases as to the *parameters vector*. Each layer of the network corresponds to a matrix whose values are computed by applying a linear transformation to the previous layer followed by the application of a nonlinear activation function. In particular, referring to Figure A.22,  $\mathbf{y}_0 = \mathbf{x} \in \mathbb{R}^{N_0}$  is the input layer,  $\mathbf{y}_L = \phi^{DF}(\mathbf{x}; \boldsymbol{\theta}_{DF}) \in \mathbb{R}^{N_L}$  is the output layer, and each hidden layer  $\mathbf{y}_i \in \mathbb{R}^{N_i}$ ,  $i = 1, \dots, L - 1$ , takes the form

$$\mathbf{y}_i = \phi_i(W_i \mathbf{y}_{i-1} + \mathbf{b}_i).$$

Given a set of  $M$  input-output pair observations  $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^M$  and considering a *supervised learning* paradigm [43], the learning task consists in finding the optimal parameters vector  $\boldsymbol{\theta}_{DF}^*$  by solving the optimization problem

$$\min_{\boldsymbol{\theta}_{DF}} \mathcal{J}(\boldsymbol{\theta}_{DF}) = \min_{\boldsymbol{\theta}_{DF}} \frac{1}{M} \sum_{i=1}^M \mathcal{L}(\mathbf{y}^i, \mathbf{y}_L^i; \boldsymbol{\theta}_{DF}) \quad (\text{A.2})$$

where  $\mathcal{J}$  is the loss (or cost) function, and  $\mathcal{L}$  is the per-example loss function, measuring the mismatch between the desired observed output  $\mathbf{y}^i$  and the approximated one  $\mathbf{y}_L^i$ . Problem (A.2) is usually solved by means of the gradient descent method exploiting the back-propagation algorithm [55] to compute the

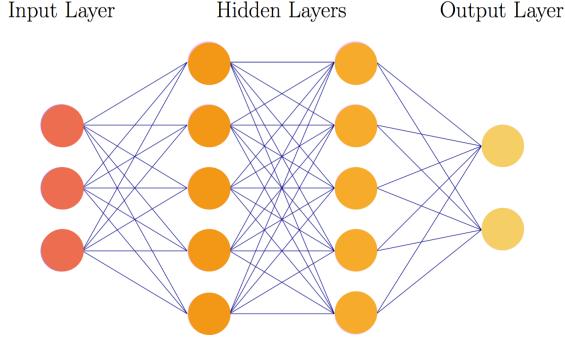


Figure A.22: Feedforward neural network

derivatives of the loss function with respect to parameters. In particular, the gradient descent method requires to evaluate

$$\nabla_{\boldsymbol{\theta}_{DF}} \mathcal{J}(\boldsymbol{\theta}_{DF}) = \frac{1}{M} \sum_{i=1}^M \nabla_{\boldsymbol{\theta}_{DF}} \mathcal{L}(\mathbf{y}^i, \mathbf{y}_L^i; \boldsymbol{\theta}_{DF}), \quad (\text{A.3})$$

a task which might easily become prohibitive when the size  $M$  of the training dataset is very large, thus causing a single step of the gradient descent method to require a huge amount of time. The stochastic gradient descent (SGD) method allows to reduce the computational cost associated to the computation of the gradient of the loss function, by exploiting the fact that (A.3) can be considered as an expectation over the entire training dataset. Such an expectation can be approximated using a small set (or *minibatch*) of samples; hence, at each iteration the SGD method samples a minibatch of  $m < M$  data points, drawn (e.g., uniformly) from the training dataset [43], and approximates the gradient (A.3) of the loss function by

$$\hat{\nabla}_{\boldsymbol{\theta}_{DF}} \mathcal{J}(\boldsymbol{\theta}_{DF}) = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}_{DF}} \mathcal{L}(\mathbf{y}^i, \mathbf{y}_L^i; \boldsymbol{\theta}_{DF}).$$

#### Appendix A.2. Convolutional neural network

Convolutional neural networks (CNNs) [56] are the standard neural network architecture in computer vision tasks, since they are well-suited to high-dimensional and spatially distributed data like images. This is due to the local approach of convolutional layers which enables them to exploit spatial correlations among pixels in order to extract low-level features of the input to carry out the task. The main ingredients of a convolutional layer are convolutional kernels, or filters, which consist in tensors of smaller dimensions with respect to the input. Each element of a feature map is obtained by sliding the kernel over the image and by computing the discrete convolution, as shown in Figure A.23.

Considering a 3-dimensional input  $Y_0 = X \in \mathbb{R}^{N_0^1 \times N_0^2 \times N_0^3}$  and a bank of  $K_i$  convolutional filters in layer  $i$  denoted as  $W_i^k \in \mathbb{R}^{n_i^1 \times n_i^2 \times n_i^3}$ ,  $i = 1, \dots, L$  and  $k = 1, \dots, K_i$ , the  $k$ -th feature map is computed as

$$Y_i^k = \phi_i(W_i^k * Y_{i-1} + b_i^k).$$

where  $Y_i \in \mathbb{R}^{N_i^1 \times N_i^2 \times N_i^3}$  (or, equivalently,  $Y_i^k \in \mathbb{R}^{N_i^1 \times N_i^2}$ ) with  $N_i^1$  and  $N_i^2$  depending on  $n_i^1$  and  $n_i^2$ , respectively, the padding and the striding strategies, and  $N_i^3 = K_i$ .

Convolutional layers are characterized by *shared parameters*, that is, weights are shared by all the elements (neurons) in a particular feature map, and *local connectivity*, that is, each neuron in a feature map is connected only to a local region of the input. Parameter sharing allows convolutional layers to enjoy another property: translation invariance or, more precisely, translation equivariance. This means that if the input varies, the output changes accordingly [43]. In particular, if we apply a transformation to the input  $Y_0$  and

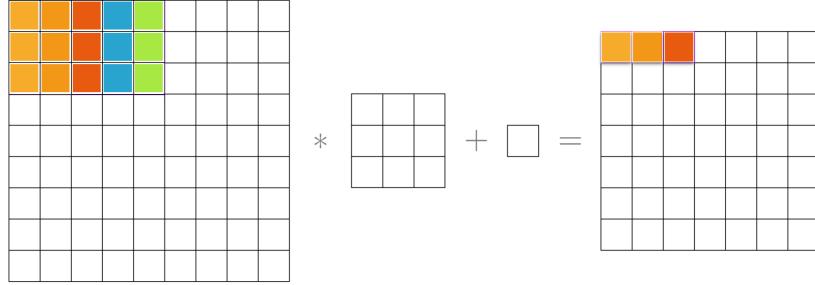


Figure A.23: Computation of the elements of a feature map in a convolutional layer.

then compute the convolution, the result is the same we would obtain by computing the convolution and then applying the transformation to the output. The two properties above increase efficiency of CNNs, both in terms of memory and computational costs, with respect to DFNNs, thus making them preferable to the latter when dealing with extremely high-dimensional data.

### Appendix A.3. Autoencoder neural network

Autoencoders (AEs) [57, 58] are a particular type of feedforward neural networks aiming at learning, under suitable constraints, the identity function

$$\mathbf{f}^{AE}(\cdot; \boldsymbol{\theta}_E, \boldsymbol{\theta}_D) : \mathbf{x}_h \mapsto \tilde{\mathbf{x}}_h \quad \text{with} \quad \tilde{\mathbf{x}}_h \simeq \mathbf{x}_h. \quad (\text{A.4})$$

Internally, an autoencoder has a hidden layer consisting in a *code* used to represent the input. We focus on undercomplete autoencoders [43] where the constraint imposed is the reduction of the dimension of the code with respect to the input and output dimension.

By considering the input  $\mathbf{y}_0 = \mathbf{x}_h \in \mathbb{R}^{N_h}$  and the output  $\mathbf{y}_L = \tilde{\mathbf{x}}_h \in \mathbb{R}^{N_h}$ , an autoencoder is composed by two main parts (see Figure A.24)

- the *encoder* function  $\mathbf{f}_n^E(\cdot; \boldsymbol{\theta}_E) : \mathbf{x}_h \mapsto \tilde{\mathbf{x}}_n = \mathbf{f}_n^E(\mathbf{x}_h; \boldsymbol{\theta}_E)$ , where  $\mathbf{f}_n^E(\cdot; \boldsymbol{\theta}_E) : \mathbb{R}^{N_h} \rightarrow \mathbb{R}^n$  and  $n \ll N_h$ , mapping the high-dimensional input  $\mathbf{x}_h$  onto the low-dimensional code  $\tilde{\mathbf{x}}_n$ . The encoder function depends on a vector of parameters  $\boldsymbol{\theta}_E \in \mathbb{R}^{N_E}$  collecting all the weights and biases specifying the function itself;
- the *decoder* function  $\mathbf{f}_h^D(\cdot; \boldsymbol{\theta}_D) : \tilde{\mathbf{x}}_n \mapsto \tilde{\mathbf{x}}_h = \mathbf{f}_h^D(\tilde{\mathbf{x}}_n; \boldsymbol{\theta}_D)$ , where  $\mathbf{f}_h^D(\cdot; \boldsymbol{\theta}_D) : \mathbb{R}^n \rightarrow \mathbb{R}^{N_h}$ , mapping the code  $\tilde{\mathbf{x}}_n$  to an approximation of the original high-dimensional input  $\tilde{\mathbf{x}}_h$ . Similarly to the encoder function, the decoder function depends on a vector of parameters  $\boldsymbol{\theta}_D \in \mathbb{R}^{N_D}$  collecting all the weights and biases specifying the function itself.

The autoencoder is then defined as

$$\mathbf{f}^{AE}(\cdot; \boldsymbol{\theta}_E, \boldsymbol{\theta}_D) : \mathbf{x}_h \mapsto \tilde{\mathbf{x}}_h = \mathbf{f}_h^D(\mathbf{f}_n^E(\mathbf{x}_h; \boldsymbol{\theta}_E); \boldsymbol{\theta}_D).$$

Autoencoder learning lays within the *unsupervised learning* paradigm [43] since its goal is to reconstruct the input being the target output an approximation of the input. An autoencoder not only learns a low-dimensional representation of the high-dimensional input but also learns how to reconstruct the input from the code through the encoder and the decoder functions.

When dealing with large inputs, as the ones arising from the discretization of system (1), the use of a feedforward autoencoder may become prohibitive as the number of parameters (weights and biases) required may be very large. As pointed out in Appendix A.2, parameter sharing and local connectivity allow to reduce the numbers of parameters of the network and the number of associated computations, both in the forward and in the backward pass, hence the idea of relying on convolutional autoencoders for the sake of building our DL-ROM technique.

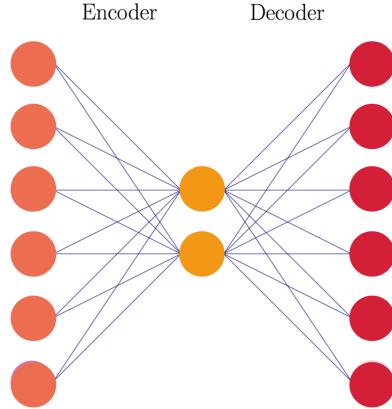


Figure A.24: Autoencoder neural network.

## References

- [1] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.
- [2] P. Benner, A. Cohen, M. Ohlberger, K. Willcox, *Model reduction and approximation: Theory and algorithms*, SIAM, 2017.
- [3] P. Benner, S. Gugercin, K. Willcox, *A survey of projection-based model reduction methods for parametric dynamical systems*, *SIAM Review* 57 4 (2015) 483–531.
- [4] A. Quarteroni, A. Manzoni, F. Negri, *Reduced basis methods for partial differential equations: An introduction*, volume 92, Springer, 2016.
- [5] M. Ohlberger, S. Rave, *Reduced basis methods: Success, limitations and future challenges*, *Proceedings of ALGORITMY* (2016) 1–12.
- [6] S. Pagani, A. Manzoni, A. Quarteroni, *Numerical approximation of parametrized problems in cardiac electrophysiology by a local reduced basis method*, *Computer Methods in Applied Mechanics and Engineering* 340 (2018) 530–558.
- [7] G. Kutyniok, P. Petersen, M. Raslan, R. Schneider, *A theoretical analysis of deep neural networks and parametric PDEs*, *arXiv preprint arXiv:1904.00377* (2019).
- [8] F. Regazzoni, L. Dede', A. Quarteroni, *Machine learning for fast and reliable solution of time-dependent differential equations*, *Journal of Computational Physics* 397 (2019).
- [9] M. Guo, J. S. Hesthaven, *Reduced order modeling for nonlinear structural analysis using gaussian process regression*, *Computer Methods in Applied Mechanics and Engineering* 341 (2018) 807–826.
- [10] M. Guo, J. S. Hesthaven, *Data-driven reduced order modeling for time-dependent problems*, *Computer Methods in Applied Mechanics and Engineering* 345 (2019) 75–99.
- [11] J. Hesthaven, S. Ubbiali, *Non-intrusive reduced order modeling of nonlinear problems using neural networks*, *Journal of Computational Physics* 363 (2018) 55–78.
- [12] O. San, R. Maulik, *Neural network closures for nonlinear model order reduction*, *Advances in Computational Mathematics* (2018).
- [13] J. N. Kani, A. H. Elsheikh, *DR-RNN: A deep residual recurrent neural network for model reduction*, *arXiv preprint arXiv:1709.00939* (2017).
- [14] A. Mohan, D. V. Gaitonde, *A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks*, *arXiv preprint arXiv:1804.0926* (2018).
- [15] Z. Wan, P. Vlachas, P. Koumoutsakos, T. Sapsis, *Data-assisted reduced-order modeling of extreme events in complex dynamical systems*, *PLOS ONE* 13 (2018).
- [16] F. J. Miranda González, M. Balajewicz, *Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems*, *arXiv preprint arXiv:1808.01346* (2018).
- [17] K. Lee, K. Carlberg, *Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders*, *arXiv preprint arXiv:1812.08373* (2018).
- [18] B. A. Freno, K. T. Carlberg, *Machine-learning error models for approximate solutions to parameterized systems of nonlinear equations*, *arXiv preprint arXiv:1808.02097* (2018).
- [19] S. Pagani, A. Manzoni, K. Carlberg, *Statistical closure modeling for reduced-order models of stationary systems by the ROMES method*, *arXiv preprint arXiv:1901.02792* (2019).
- [20] E. Parish, K. Carlberg, *Time-series machine-learning error models for approximate solutions to parameterized dynamical systems*, *arXiv preprint arXiv:1907.11822* (2019).
- [21] S. Trehan, K. T. Carlberg, L. J. Durlofsky, *Error modeling for surrogates of dynamical systems using machine learning*, *International Journal for Numerical Methods in Engineering* 112 (2017) 1801–1827.

- [22] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, *Journal of Computational Physics* 357 (2018) 125–141.
- [23] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, *arXiv preprint arXiv:1711.10561* (2017).
- [24] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, *arXiv preprint arXiv:1711.10566* (2017).
- [25] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational Physics* 378 (2019) 686–707.
- [26] M. Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, *Journal of Machine Learning Research* 19 (2018) 1–24.
- [27] A. Quarteroni, R. Sacco, F. Saleri, *Matematica Numerica*, Springer Milan, 2008.
- [28] A. Manzoni, S. Pagani, T. Lassila, Accurate solution of Bayesian inverse uncertainty quantification problems combining reduced basis methods and reduction error models, *SIAM/ASA Journal on Uncertainty Quantification* 4 (2016) 380–412.
- [29] M. Barrault, Y. Maday, N. C. Nguyen, A. T. Patera, An ‘empirical interpolation’ method: Application to efficient reduced-basis discretization of partial differential equations, *Comptes Rendus Mathématique de l’Académie des Sciences* 339 (2004) 667–672.
- [30] S. Chaturantabut, D. C. Sorensen, Nonlinear model reduction via discrete empirical interpolation, *SIAM Journal on Scientific Computing* 32 (2010) 2737–2764.
- [31] C. Farhat, S. Grimberg, A. Manzoni, A. Quarteroni, Computational bottlenecks for PROMs: Pre-computation and hyperreduction, Technical Report, 2019. Submitted.
- [32] D. Amsallem, M. J. Zahr, C. Farhat, Nonlinear model order reduction based on local reduced-order bases, *International Journal for Numerical Methods in Engineering* 92 (2012) 891–916.
- [33] D. Amsallem, M. J. Zahr, K. Washabaugh, Fast local reduced basis updates for the efficient reduction of nonlinear systems with hyper-reduction, *Advances in Computational Mathematics* 41 (2015) 1187–1230.
- [34] D. Amsallem, B. Haasdonk, PEBL-ROM: Projection-error based local reduced-order models, *Advanced Modeling and Simulation in Engineering Sciences* 3 (2016) 6.
- [35] R. Abgrall, D. Amsallem, R. Crisovan, Robust model reduction by  $l^1$ -norm minimization and approximation via dictionaries: application to nonlinear hyperbolic problems, *Advanced Modeling and Simulation in Engineering Sciences* 3 (2016).
- [36] J.-F. Gerbeau, D. Lombardi, Approximated lax pairs for the reduced order integration of nonlinear evolution equations, *Journal of Computational Physics* 265 (2014) 246–269.
- [37] A. Iollo, D. Lombardi, Advection modes by optimal mass transfer, *Physical Review E* 89 (2014) 022923.
- [38] N. Cagniart, Y. Maday, B. Stamm, Model order reduction for problems with large convection effects, in: Contributions to partial differential equations and applications, volume 47 of *Computational Methods in Applied Sciences*, Springer, Cham, 2019, pp. 131–150.
- [39] J. Reiss, P. Schulze, J. Sesterhenn, V. Mehrmann, The shifted proper orthogonal decomposition: A mode decomposition for multiple transport phenomena, *SIAM Journal on Scientific Computing* 40 (2018) A1322–A1344.
- [40] B. Peherstorfer, Model reduction for transport-dominated problems via online adaptive bases and adaptive sampling, *arXiv preprint arXiv:1812.02094* (2018).
- [41] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (ICLR), 2015.
- [42] H. Robbins, S. Monro, A stochastic approximation method, *The Annals of Mathematical Statistics* 22 (1951) 400–407.
- [43] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [44] D. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), *arXiv preprint arXiv:1511.07289* (2015).
- [45] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2015) 1026–1034.
- [46] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, Tensorflow: A system for large-scale machine learning (2016) 265–283.
- [47] R. FitzHugh, Impulses and physiological states in theoretical models of nerve membrane, *Biophysical Journal* 1 (1961) 455–466.
- [48] J. Nagumo, S. Arimoto, S. Yoshizawa, An active pulse transmission line simulating nerve axon, *Proceedings of the IRE* 50 (1962) 2061–2070.
- [49] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS’2012)* 1 (2012) 1097–1105.
- [50] G. Antipov, M. Baccouche, J. Dugelay, Face aging with conditional generative adversarial networks, *Proceedings of the IEEE International Conference on Image Processing (ICIP)* (2017).
- [51] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS’2014)* 2 (2014) 3104–3112.
- [52] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, *arXiv preprint arXiv:1810.04805* (2018).
- [53] H. Bourlard, C. Wellekens, Speech pattern discrimination and multi-layered perceptrons, *Computer Speech and Language* 3 (1989) 1–19.

- [54] J. S. Chung, A. W. Senior, O. Vinyals, S. Zisserman, Lip reading sentences in the wild, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017) 3444–3453.
- [55] D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors, Nature (1986) 533–536.
- [56] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient based learning applied to document recognition, Proceedings of the IEEE (1998) 533–536.
- [57] H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biological Cybernetics 59 (1998) 291–294.
- [58] G. E. Hinton, R. S. Zemel, Autoencoders, minimum description length, and Helmholtz free energy, Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS'1993) (1994).