Challenges and Opportunities for Machine Learning in Fluid Mechanics

M. A. Mendez, J. Dominique, M. Fiore, F. Pino,
P. Sperotto, J. Van den Berghe

14 January 2022

Abstract

Big data and machine learning are driving comprehensive economic and social transformations and are rapidly re-shaping the toolbox and the methodologies of applied scientists. Machine learning tools are designed to learn functions from data with little to no need of prior knowledge. As continuous developments in experimental and numerical methods improve our ability to collect high-quality data, machine learning tools become increasingly viable and promising also in disciplines rooted in physical principles. These notes explore how machine learning can be integrated and combined with more classic methods in fluid dynamics. After a brief review of the machine learning landscape, we show how many problems in fluid mechanics can be framed as machine learning problems and we explore challenges and opportunities. We consider several relevant applications: aeroacoustic noise prediction, turbulence modelling, reduced-order modelling and forecasting, meshless integration of (partial) differential equations, super-resolution and flow control. While this list is by no means exhaustive, the presentation will provide enough concrete examples to offer perspectives on how machine learning might impact the way we do research and learn from data.

Keywords
Machine Learning for Fluid Dynamics, Turbulence Modeling, Aeroacoustics Noise Prediction, Dimensionality Reduction, Reinforcement Learning, Meshless Methods for PDEs.

1 What is Machine Learning?

Machine learning is a subset of Artificial Intelligence (AI) at the intersection of computer science, statistics, engineering, neuroscience, and biology. The term 'machine learning' was coined by Samuel (1959), to refer to *the field of study that gives the computers the ability to learn without being explicitly programmed*. A more precise, engineering-oriented definition by Mitchell (1997) reads: *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E*. Let us delve into this definition.

Focusing on engineering applications, we replace the subject 'computer program' with 'model'. The learning of a model begins with the definition of a *task* (T): recognizing faces

in images or predict flow separation from operating conditions in an airfoil, learn how to play chess or learn how to stabilize an unstable flow. These tasks can be formulated as a function from an input $\mathbf{x} \in \mathcal{X}$ to an output $\mathbf{y} \in \mathcal{Y}$. The function might define what output matches with the input (e.g. in image recognition) or what action to take when a system is at a given state (e.g. what the best next chess move is).

The second ingredient is *experience* (E), i.e. a collection of data points in $\mathcal{X}$ and $\mathcal{Y}$. These data might be available from the beginning (as in *supervised learning*) or might be collected *while* learning (as in *reinforcement learning*). The third ingredient is an *hypothesis set*, e.g. a parametric representation of the function $f \approx \tilde{f}(\mathbf{x}, \mathbf{w})$ which depends on some parameters (weights) $\mathbf{w} \in \mathcal{W}$. The model $\tilde{f}(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2$ is an example with three parameters and rather small capacity (the set of all parabolas); an Artificial Neural Network (ANN) is an example with thousands (or millions!) parameters and a much larger capacity (potentially *any* function). The set of weights, and the associated hypothesis set, define how the computer performs the task, i.e. what $\mathbf{x} - \mathbf{y}$ association it makes, or what actions $\mathbf{y}$ it takes when in state $\mathbf{x}$.

The fourth ingredient is a *learning algorithm*, based on some *performance measure* P. This includes the definition of a cost function that must be minimized or a reward function that must be maximized. Learning is, in essence, an optimization problem that seeks to find the *best* set of weights $\mathbf{w}$ for a given task and within a hypothesis set $f(\mathbf{x}, \mathbf{w})$. The process of optimizing the weights is called *training*, and can be made offline or online: in the first case, the learner is trained from a training dataset and then deployed with seldom updates afterwards; in the second case the learning is performed incrementally, from data acquired sequentially.

Finally, the last ingredient is the definition of the final hypothesis, its validation and the quantification of uncertainties. To perform unbiased evaluation of the model, not all data is used for training: a portion is retained to test the model in unseen data and, hence, to estimate its ability to generalize. Before proceeding with examples, we introduce the three paradigms of learning in the following subsections. The reader is referred to Mendez et al. (2022); Brunton et al. (2020); Brenner et al. (2019) for reviews and perspectives on machine learning for fluid dynamics.

1.1 Supervised Learning

In supervised learning, a large set of data $(\mathbf{x}^*, \mathbf{y}^*)$ is provided by a *supervisor* and available for training. Supervised learning tasks are *classification* and *regression*. Classification targets the mapping of categorical data. Examples are the classification of an email as "spam" or "not spam", or the image-based classification of a flow as "bubbly flow" or "churn flow". Regression targets the mapping of continuous space. Examples are the prediction of a car's price based on mileage and age, or the prediction of the eddy viscosity given the mean flow velocity gradient in a turbulent flow.

Machine learning tools for classification and regression are entering in the literature of fluid flows. Examples of automatic data-driven classification of flow regimes are provided in Majors (2018); Hobold and da Silva (2018); Kang et al. (2020). Regression problem are arguably more common, encompassing surrogate model-based optimization Kim and Boukouvala (2019), turbulence modeling Duraisamy et al. (2019), non-intrusive Reduced Order Modeling Daniel et al. (2020); Hesthaven and Ubbiali (2018); Renganathan et al.

(2020), aeroacoustic noise prediction Dominique et al. (2021) and system identification for prediction and control Pan and Duraisamy (2018); Brunton et al. (2016); Huang and Kim (2008). The main challenge in setting these problems is the choice of the hypotheses set balancing model complexity versus available data, and the implementation of physical constraints in the learning. Both aspects are briefly illustrated in Section 2.

Regardless of the application, the final outcome is a "surrogate" model that can make predictions. These models comes in various shapes and sizes: examples considered in this work are linear combination of radial basis functions (RBFS) (Buhmann, 2003), Artificial Neural Networks (Goodfellow et al., 2016) or recursive expression trees as in Genetic Programming (Banzhaf et al., 1997). These models provide analytical representations, hence amenable to analytic differentiation. This enables meshless integration of Partial Differential Equations, discussed in Section 3.

1.2 Unsupervised Learning

Unsupervised learning is also known as descriptive learning. The goal is to discover patterns or characteristic features in the data. Hence the function to be learned is from the input space to itself, i.e. $f : \mathcal{X} \to \mathcal{X}$. The main unsupervised learning tasks are *dimensionality reduction* and *clustering*.

Dimensionality reduction aims at identifying a lower-dimensional representation of the data. The underlying assumption is that a few features (called *hidden* or *latent factors*) contain the essential information in the data. A successful face recognition algorithm, for example, focuses on image patterns that are associated with age, gender or pose, and constructs a reduced set of templates that encodes the essential information enabling recognition (Swets and Weng, 1996).

Clustering aims at partitioning the data into groups (clusters) that share some common features or are similar according to certain metrics. Clustering differs from classification in its unsupervised nature: no labelled data is available, and no "right answer" is known upfront– not even the number of clusters. A simple example of a clustering problem is finding customers with similar purchase behaviour as a basis for recommendation engines. Similarly, one could cluster snapshots of a flow field based on their degree of similarity and construct reduced models of a fluid flow (Kaiser et al., 2014).

Machine learning tools for clustering and dimensionality reduction are becoming increasingly popular in fluid mechanics. The quest for identifying (and objectively define) coherent structures in turbulent flows has a long history and vast literature. Linear tools such has Proper Orthogonal Decomposition (POD) have been extensively used to construct reduced order models of fluid flows (Holmes et al., 1997), to find optimally balanced control laws (Ilak and Rowley, 2008), to perform correlation-based filtering (Mendez et al., 2017) or to identify correlations between different flow quantities (Borée, 2003), to name a few examples.

Most of the decomposition methods developed in fluid mechanics are linear, and the literature has grown into a subfield of data processing often referred to as *Data-Driven Modal Analysis* (Taira et al., 2017), where the notion of 'mode' generalizes that of 'coherent structure' or 'principal component' or 'harmonic (Fourier) mode'. Nonlinear methods of dimensionality reduction have comparatively much less history and have been popularized mostly in the last few years. Notable examples are the use of manifold learning

techniques such as Locally Linear Embedding (LLE) (Ehlert et al., 2020), cluster-based reduced-order model (Kaiser et al., 2014) and autoencoders (Murata et al., 2019). An illustrative application of linear and nonlinear dimensionality reduction is provided in Section 4.

1.3 Reinforcement Learning

Reinforcement learning (RL) is about learning the mapping $f : \mathcal{X} \to \mathcal{Y}$ having no data available upfront. The only viable learning approach is by trial and error. The learner is thus a decision-making *agent* which interacts with an *environment* by taking *actions* that leads to *rewards* (or *penalties*). The agent learns from mistakes and seeks to maximize the rewards (or minimize the penalties). The mapping to be learned is called *policy*. This maps the state of the system to the action that needs to be taken in order to achieve the goal. The reader is referred to Arulkumaran et al. (2017); Hernandez-Leal et al. (2019) for extensive surveys, to Richard S. Sutton (2018); François-Lavet et al. (2018) for complete introductions to the topic and to Alexander Zai (2020) for hands-on tutorials on Python.

The recent interest in this field has been motivated by standout achievements in board games (Silver et al., 2016, 2018) and video-games (Szita, 2012), robotics (Kober and Peters, 2014) and language processing (Luketina et al., 2019). An historical turning point was the success of a hybrid RL system that defeated the Korean world champion Lee Sedol in the game of Go (Silver et al., 2016). The first applications of RL in fluid mechanics were focused on the study of collective behavior of swimmers, pioneered by Koumoutsakos's group (Wang et al., 2018; Verma et al., 2018; Novati et al., 2017; Novati and Koumoutsakos, 2019; Novati et al., 2019), while the first applications for flow control were presented by Pivot et al. (2017) and by Rabault et al. (2019). Bucci et al. (2019) used RL to control the one-dimensional Kuramoto–Sivashinsky equation while Beintema et al. (2020) used it to control heat transport in a two-dimensional Rayleigh–Bérnard systems. Verma et al. (2018) uses RL to study how fishes can reduce energy expenditure by schooling while Reddy et al. (2016) uses RL to analyze how bird and gliders minimize energy expenditure when soaring by exploiting turbulent fluctuations. An illustrative application of RL in flow control is provided in Section 5.

2 Physics Informed Regression

Consider the problem of finding a mapping $\mathbf{y} = f(\mathbf{x})$, with $\mathbf{x} \in \mathbb{R}^{n_x}$ and $\mathbf{y} \in \mathbb{R}^{n_y}$ using a dataset $\{\mathbf{x}_*, \mathbf{y}_*\}$ for training and a dataset $\{\mathbf{x}_{**}, \mathbf{y}_{**}\}$ for validation. Let $\tilde{f}(\mathbf{x}, \mathbf{w})$ denote the parametric hypothesis depending on the weights $\mathbf{w}$ and let us consider the case this is a fully connected feed forward Artificial Neural Network (ANN). These are distributed architectures consisting of a large number of simple connected units (called neurons), organized in layers (Goodfellow et al., 2016). The mapping from input to output takes the recursive form:

$$\mathbf{y} = \tilde{f}(\mathbf{x}, \mathbf{w}) = \sigma^{(L)}\left(\mathbf{z}^{(L-1)}\right) \quad \text{with} \begin{cases} \mathbf{a}^{(1)} = \mathbf{x},\ \mathbf{a}^{(l)} = \sigma^{l}\left(\mathbf{z}^{(l)}\right) \\ \mathbf{z}^{(l)} = \mathbf{W}^{(l-1)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \end{cases} \quad \text{and}\, l = 1, 2, \dots L\,. \tag{1}$$

Here $\mathbf{a}^{(l)}, \mathbf{b}^{(l)} \in \mathbb{R}^{n_l \times 1}$ are the *activation vector* and the *bias vector* of layer l, composed of n_l neurons, $\sigma^{(l)}$ is the activation function in each layer, $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the matrix containing the weights connecting layer $l-1$ with layer l, and L is the number of layers. The vector $\mathbf{w} \in \mathbb{R}^{n_w \times 1}$ collects all the weights and biases across the network, hence $n_w = n_L n_{L-1} + n_{L-2} n_{L-3} + \ldots n_2 n_1 + n_L + n_{L-1} + \ldots n_1$.

The activation functions are nonlinear functions such as hyperbolic tangents. The zoology of activation functions counts dozens of possibilities; for the scope of this talk, it suffices noticing that a sufficiently large network can 'learn' approximations of *any* function (Cybenko, 1989). Training the network simply consists in solving nonlinear least square problem which seeks to minimize a cost function. The most classic form is the mean square error $J(\mathbf{w}) = ||\mathbf{y}_* - f(\mathbf{x}_*, \mathbf{w})||_2^2$, with $|| \bullet ||_2$ denoting the l_2 norm.

Besides the unique function approximation capabilities of ANN, their burgeoning popularity is arguably due to two more factors. The first is the possibility to easily compute the cost function's gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$ using the chain rule for differentiation, which leads to the popular back-propagation algorithm (Rumelhart and McClelland, 1989). This allows to use an arsenal of optimization tools for the training. In their stochastic 'batch' formulation, these can easily handle extremely large datasets. The second is the diffusion of powerful and accessible open-source libraries such as *Tensorflow*[1] or *Pytorch*[2].

As first example of the regression capabilities of an ANN, we consider the problem of predicting wall pressure spectra in a turbulent boundary layer using integral parameters such as boundary layer thickness or friction coefficient. This work is presented by Dominique et al. (2022), who trained an ANN on a large dataset of numerical and experimental data, and compared the predictive performance to classic empirical correlations. These correlations are derived following several physical argument, but are ultimately adjusted using standard curve fitting tools. Figure 1 demonstrates the predictive capabilities of the trained ANN, consisting of three layers with ten neurons each. The input parameter consists of dimensionless numbers linking a set of carefully chosen integral parameters, so that the learned function takes the form:

$$\frac{\Phi_{pp}(\omega) U_e}{\delta^* \tau_w^2} = f\left(\frac{\omega \delta^*}{U_e}, \beta_c, R_T, C_f, H, \Delta, M, \Pi\right), \tag{2}$$

where $\Phi_{pp}(\omega)$ is the wall pressure spectrum, δ^* is the boundary layer's displacement thickness, τ_w is the wall shear stress, U_e is the (local) free stream velocity, $\beta_c = (\theta/\tau_w)(dp/dx)$ is the Clauser's parameter accounting for the pressure gradient dp/dx (Clauser, 1954), θ is the boundary layer's momentum thickness, $R_T = (\delta^*/U_e)/(\nu/u_\tau)$ is the ratio of characteristic time scales, with ν the fluid viscosity and $u_\tau = \sqrt{\tau_w/\rho}$ the friction velocity, $C_f = \tau_w/(0.5\rho U_e)$ is the skin friction coefficient, $H = \delta^*/\theta$ is the shape factor, $\Delta = \delta^*/\delta$, with δ the boundary layer thickness, $M = U_e/c_0$ is the Mach number with c_0 the free stream sound speed and Π is Coles' wake parameter (Coles, 1956). The test case is the one from Deuse's DNS dataset (Deuse and Sandberg, 2020) and features the flow past an airfoil at Mach number $M = 0.25$ and Reynolds number $Re = U_\infty c/\nu = 1.5 \times 10^5$.

In Figure 1, the prediction on the right is made at a location $x_c = 0.02$ (i.e. at two % of the chord) from the trailing edge. The pressure spectrum is compared with the

[1] See `https://www.tensorflow.org/`

[2] See `https://pytorch.org/`

two popular models of Goody (2004) and Lee (2018), together with the prediction of a recursive expression trees trained via Genetic programming in Dominique et al. (2021). Both Machine learning approaches outperform existing models in terms of predictive capabilities.

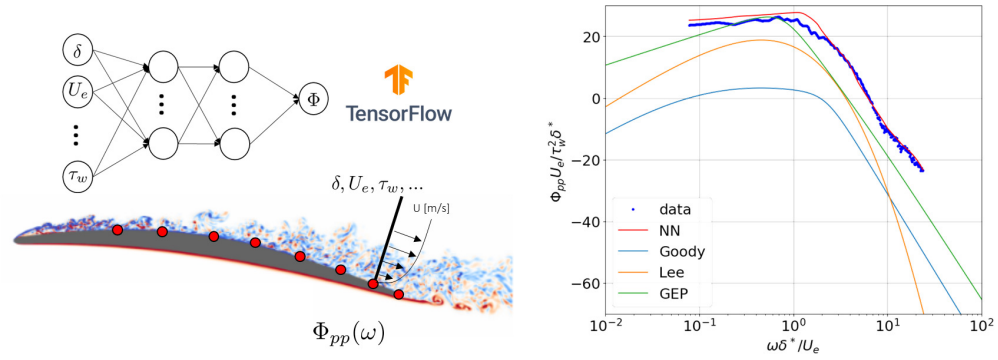

Figure 1: Wall pressure spectra prediction from from integral boundary layer parameters using an ANN trained on a large numerical and experimental database. The left figure shows the considered test case from Deuse and Sandberg (2020). The wall pressure spectra at $x_c = 0.02$ is shown on the right and compared with the prediction of Goody's and Lee's models as well as Genetic programming and ANN models.

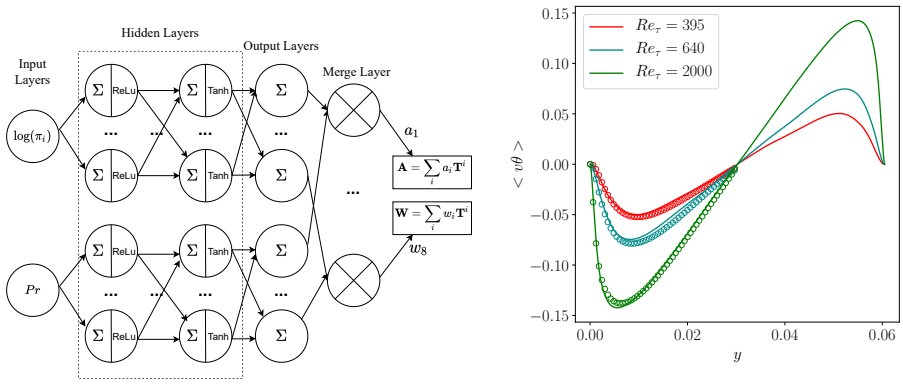

Figure 2: Thermal Turbulence Modeling via ANN. The left figure shows the scheme of the ANN used to predict the coefficients a_i and w_i in (3). The right figure depicts the wall-normal turbulent heat flux resulting from the OpenFoam simulations carried out with the data-driven thermal model in case of turbulence channel flow at $Pr = 0.025$. Images adapted from Fiore et al. (2022).

Physical consideration can be added during the training process using two possible approaches: either by defining an hypothesis that respects certain physical constraints, either by using constraints or penalization in the optimization driving the training. Both methods are illustrated by Fiore et al. (2021, 2022) for the prediction of turbulent heat flux in low Prandtl number fluids. In this context, the mapping to be learned is an algebraic model for the turbulent heat flux $\overline{\mathbf{u}\theta}$, based on a generalized gradient hypothesis. The hypothesis set is designed to be of the form:

$$\overline{\mathbf{u}\theta} = -\mathbf{D}\nabla T \quad \text{with} \quad \mathbf{D} = \left[(\mathbf{A} + \mathbf{A}^T)(\mathbf{A}^T + \mathbf{A}) + \frac{k}{\sqrt{\varepsilon}}(\mathbf{W} - \mathbf{W}^T) \right], \tag{3}$$

where T is the local average temperature, k and ε are the turbulent kinetic energy and dissipation rates, and the tensors $\mathbf{A}$ and $\mathbf{W}$ are written as linear combinations of basis tensors $\mathbf{T}_i$:

$$\mathbf{A} = \sum_{i=1}^{n} a_i \mathbf{T}_i \text{ with } a_i = f_i(\pi_i, Re_\tau, Pr) \quad \text{and} \quad \mathbf{W} = \sum_{i=1}^{n} w_i \mathbf{T}_i \text{ with } w_i = g_i(\pi_i, Re_\tau, Pr)\,. \tag{4}$$

The coefficients a_i and g_i are functions of the turbulent Reynolds number Re_τ, the Prandtl number Pr and a set of invariant basis π_i, i.e. isotropic quantities that are invariant under rotation of the coordinate system. The functions f_i and g_i for the coefficients in (4) are needs to be learned by an ANN.

Neither the invariant bases π_i nor the basis tensors $\mathbf{T}_i$ are shown in this short article and the reader is referred to Fiore et al. (2021, 2022) for more details. For the purposes of this overview, it suffices noticing that (3) and (4) limits the margin of manoeuvre of the ANN in making predictions that satisfy the Galilean and geometrical invariant properties as well as rotational invariant. Moreover, because $\mathbf{D}$ is at least positive semi-definite, it can be shown that the predicted heat flux respects the second principle of thermodynamics and only flows from higher temperatures to lower ones. Figure 2 demonstrates the predictive capabilities of the proposed ANN for turbulent channel flows at different Reynolds numbers and $Pr = 0.025$. The network was trained up to $Re_\tau = 640$. Hence, the model seems able to accurately extrapolate the turbulent heat flux at higher Reynolds numbers ($Re_\tau = 2000$).

3 From Regression to Super-resolution and Meshless PDE Integration

Most of the parametrized models $\tilde{f}(\mathbf{x}, \mathbf{w})$ employed in machine learning are easily differentiable with respect to the inputs. It is thus natural to further constrain the training by imposing that $\tilde{f}$ is solution of a Partial Differential Equation (PDE). Given a differential operator $\mathcal{N}\{u\}$ (for example $\mathcal{N}\{u(x,t)\} = \partial_t u + \partial_x u$ for the 1D advection equation) and $\mathcal{N}\{u\} = 0$ a PDE, inserting the ansatz $u \approx \tilde{f}(\mathbf{x}, \mathbf{w})$ returns a least square problem in $\mathbf{w}$, whose solution gives an *analytic* approximation of the PDE solution.

No computational mesh is needed. On the contrary, the resulting approximation can be evaluated at *any* mesh, hence enabling super-resolution (a term often borrowed from computer vision (Bashir et al., 2021)) when the training data comes from experiments.

The idea of meshless integration of PDEs using ANN is rather old (see Lagaris et al. (1998)), and the same is true for the PDE integration via RBFs (see Fornberg and Flyer (2015)). Nevertheless, recent contributions on Physics Informed Neural Networks (PINs Raissi et al. (2019)) and brilliant online tutorials available online[3] are making these tools more accessible.

[3]See `https://maziarraissi.github.io/research/1_physics_informed_neural_networks/`.

This talk shows an example of data-driven meshless integration of the Poisson equation to measure pressure fields from image velocimetry. This work is presented by Sperotto et al. (2021) and uses Gaussian Radial Basis Functions as function approximators. These are linear models (w.r.t. to the weights $\mathbf{w}$) of the form

$$\mathbf{y} = \tilde{f}(\mathbf{x}, \mathbf{w}) = \sum_{k=1}^{n_\phi} w_k \varphi_k(\mathbf{x}|\mathbf{x}^*, c_k) \quad \text{with} \quad \varphi_k(\mathbf{x}|\mathbf{x}_k^*, c_k) = \exp\left(-c_k^2 \left\|\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{k}}^*\right\|^2\right). \tag{5}$$

Here $c_k > 0$ are the *shape parameters* and $\boldsymbol{x}_{\boldsymbol{k}}^*$ are the *collocation points* of the basis functions. Derivatives of these expansions are trivially computed by replacing the basis functions with their derivatives.

In the pressure integration algorithm, the RBFs are first used to construct an analytic expression of the velocity field $\mathbf{u}(\mathbf{x})$ from data produced by Particle Image Velocimetry (PIV) or Lagrangian Particle Tracking (LPT). This step is cast in the form of a constrained least square problem ensuring that the approximation respects boundary conditions and physical priors such as the divergence free condition for an incompressible flow. The resulting velocity approximation is then introduced in the pressure Poisson equation, leading to a second least square problem. For an incompressible flow, this problem reads:

$$\text{Given } p = \sum_{k=1}^{n_\phi} w_k \varphi_k(\mathbf{x}|\mathbf{x}^*, c_k) \;\rightarrow\; \nabla^2 p = \sum_{k=1}^{n_\phi} w_k \nabla^2 \varphi_k(\mathbf{x}|\mathbf{x}^*, c_k) = \rho \nabla \cdot (\mathbf{u} \nabla \mathbf{u}) \tag{6}$$

This linear problem is solved using the same techniques used for the velocity approximation, with constraints imposing boundary conditions for the pressure field.

An illustrative result is shown in Figure 3. This is the pressure integration in the flow past a cylinder in laminar conditions. The figure on the left is a snapshot of the scattered velocity field from which the pressure field is computed via RBFs. The velocity data is taken by down-sampling a CFD simulation by Rao et al. (2020) so as to obtain a scattered field consisting of $n_s = 18755$ vectors, simulating a LPT measurements. The figure on the right compares the resulting pressure distribution along the cylinder, as a function of the angular coordinate θ, with the CFD results. The reconstruction is in excellent agreement.

4 From Modal Analysis to Manifold Learning

The quest for identifying coherent structures in turbulent flows has a long history in fluid mechanics (see Holmes et al. (1997); Hussain (1983)). The most ubiquitous tool developed at the scope is the Proper Orthogonal Decomposition (POD), which is essentially equivalent to the Principal Component Analysis (PCA) in the machine learning literature Bishop (2006). This is a linear tool: it decomposes a dataset as a linear combination of contributions called modes. These are computed to minimize the error in an approximation built excluding some of the modes.

Many variant exist (see Mendez et al. (2019)) and the range of application spans filtering, compression, feature identification, reduced order modeling and more. However, linear tools are only a small subset in the arsenal of data compression and manifold

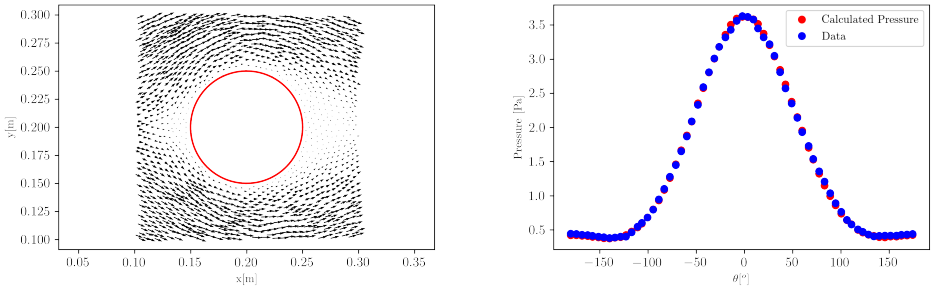

Figure 3: Data-driven pressure reconstruction from 2D scattered velocity fields. The left figure shows a snapshot of the scattered velocity data in the flow past a cylinder (dataset re-sampled from CFD simulations in Rao et al. (2020)). The right figure compares the meshless RBF integration (red circles) to the CFD data (blue circles), along the cylinder surface.

learning tools available to the machine learner (Bishop, 2006). We illustrate the use of a nonlinear reduced order modeling tool known as kernel (kPCA) (Schölkopf et al., 1997). This is essentially a *kernelized* version of the POD. The underlying idea is to perform the PCA/POD on a dataset which has been first transformed by a nonlinear function called kernel function ξ. Let us briefly illustrate the conceptual differences between kPCA and PCA.

In a linear decomposition, each mode represents the component of the dataset along a certain *basis element*. Assume that the dataset is reshaped into a matrix $\mathbf{X} \in \mathbb{R}^{n_s \times n_t}$, collecting the snapshots of n_s spatial points along its columns and the temporal evolution (time series of length n_t) in each point along its rows. The PCA seeks the optimal basis from the eigenvalues of $\mathbf{X}\mathbf{X}^T$. An approximation $\tilde{\mathbf{X}}$ is computed as:

$$\tilde{\mathbf{X}}(x_i, t_k) = \sum_{r=1}^{R} c_r(t_k)\phi_r(x_i) \text{ where } c_r(t_k) = \phi_r^T(x_i)\mathbf{X}(x_i, t_k) = \mathbf{z}_r^T[k] \quad \text{with } (\mathbf{X}\mathbf{X}^T)\phi_r = \lambda_r \phi_r \,. \tag{7}$$

Here x_i and t_k are, respectively, the spatial and temporal meshes. Both the basis elements (principal components) $\phi_r(x_i)$ and the coefficients $c_r(t_k) = \mathbf{z}_r^T$ are stored as column vectors $\phi_r \in \mathbb{R}^{n_s \times 1}$ and $\mathbf{z}_r \in \mathbb{R}^{n_t \times 1}$. These coefficients are the projection of the data onto the basis vectors. Each mode is a field $\phi_r(x_i)$ which evolves in time as $c_r(t_k)$. By truncating the expansion to $R \ll \text{rank}(\mathbf{X})$, a low order representation is built.

In kPCA, we construct the basis from the eigenvectors of the kernelized matrix $\xi(\mathbf{X}) \in \mathbb{R}^{n_F \times n_t}$. This represents a nonlinear mapping onto a space, called the *feature space*, of dimension n_F (possibly infinite). Linear operations in the feature space (e.g. projections) are nonlinear in the original space. The principal components $\phi_{\xi r} \in \mathbb{R}^{n_F \times 1}$ (i.e. the eigenvectors of $\xi(\mathbf{X})\xi^T(\mathbf{X})$) live in the feature space and are nonlinear functions which offer more model capacity than the linear expansion in (7).

In practice, we might not even need an explicit definition of ξ because we are solely concerned with inner products in the feature space. We can take these using the *kernel trick*(Bishop, 2006; Schölkopf et al., 1997). This allows for avoiding all operations in the feature space and for computing the projection of the data $\mathbf{X}$ onto the principal

components of the feature space $\phi_{\xi r}$ without ever computing $\phi_{\xi r}$. The trick goes as follows.

Let us write the $\phi_{\xi r}$ as linear combinations of the features:

$$\phi_{\xi r} = \sum_{i=1}^{n_p} a_r(t_k)\xi(\mathbf{X}(x_i, t_k)) = \xi(\mathbf{X})\mathbf{a}_r \,. \tag{8}$$

We introduce this ansatz into the eigenvalue problem, which we multiply by $\xi(\mathbf{X})^T$:

$$\xi(\mathbf{X})^T(\xi(\mathbf{X})\xi(\mathbf{X})^T)\xi(\mathbf{X})\mathbf{a}_r = \lambda_r\xi(\mathbf{X})^T\xi(\mathbf{X})\mathbf{a}_r \rightarrow \mathbf{K}^2\mathbf{a}_r = \lambda_r\mathbf{K}\mathbf{a}_r \,. \tag{9}$$

where $\mathbf{K} = \xi(\mathbf{X})^T\xi(\mathbf{X})$ collects the inner products in the feature space. If the kernel function is carefully chosen (Bishop, 2006), it is possible to compute these inner products via a *kernel function*:

$$\mathbf{K}_{i,j} = \xi(\mathbf{x}_i)^T\xi(\mathbf{x}_j) = \kappa(\mathbf{x}_i, \mathbf{x}_j) \,, \tag{10}$$

where $\mathbf{x}_i$ and $\mathbf{x_j}$ are two snapshots of the data. If this matrix is invertible, equation (9) reduces to $\mathbf{K}\mathbf{a}_r = \lambda_r\mathbf{a}_r$, i.e. an eigenvalue problem for $\mathbf{K}$. Given the eigenvectors $\mathbf{a}_r$, the projection of a feature vector $\xi(\mathbf{x}_i)$ onto the principal component $\phi_{\xi r}$ in the feature space gives:

$$\mathbf{z}_r^T = \phi_{\xi r}^T\xi(\mathbf{X}) = \mathbf{a}_r^T\xi(\mathbf{X})^T\xi(\mathbf{X}) = \mathbf{a}_r^T\kappa(\mathbf{X}, \mathbf{X}) \,. \tag{11}$$

Reconstructing an approximation of the data from a few set of kPCA components is non trivial problem and its presentation is here omitted. Interested readers are referred to Bakir et al. (1999). We close this section with an application, illustrated in Figure 4.

The dataset consists of $n_t = 13200$ velocity fields of the flow past a cylinder of diameter $d = 5$mm, measured via Time Resolved Particle Image Velocimetry (TR-PIV) at $f_s = 3kHz$ over a grid of 71×30 points. Details on the experimental work are provided in Mendez et al. (2020). This test case is characterized by a large scale variation of the free-stream velocity, which is decreased from a first steady state at $U_\infty \approx 12$m/s to second steady state at $U_\infty \approx 8$m/s. The figures on the right show an approximation of a velocity field snapshot using the three leading modes from the PCA (top) and from the kPCA (bottom). The velocity fields are snapshot vectors $\mathbf{x}_i$ in a $n_s = 2 \times 71 \times 30 = 4260$ dimensional space.

The trajectory of the dataset along the three leading modes is shown on the left for both the PCA (top) and the kPCA (bottom). The manifolds are quite different: in the PCA, the learned 3D representation collapse around a paraboloid with principal axis along $\mathbf{z}_3$. Each of the steady states corresponds to circles at constant $\mathbf{z}_1$, and the transient leads the shift from the first circular orbit (with larger radius) to the second one. The existence of such paraboloid in an optimally linear basis of eigenfunctions was firstly derived in Noack et al. (2003) for much lower Reynolds numbers. The 3D mapping produced by the kernel PCA appears as a distorted version of the one produced by the PCA. The kinematics of the data in this space is similar to the PCA, but the region of higher velocities are nonlinearly compressed in orbits of much lower radius.

The global (i.e. over both space and time) root mean square (RMS) error is reported on the top of each figure while the local (i.e. on the shown snapshot) RMS is shown on the

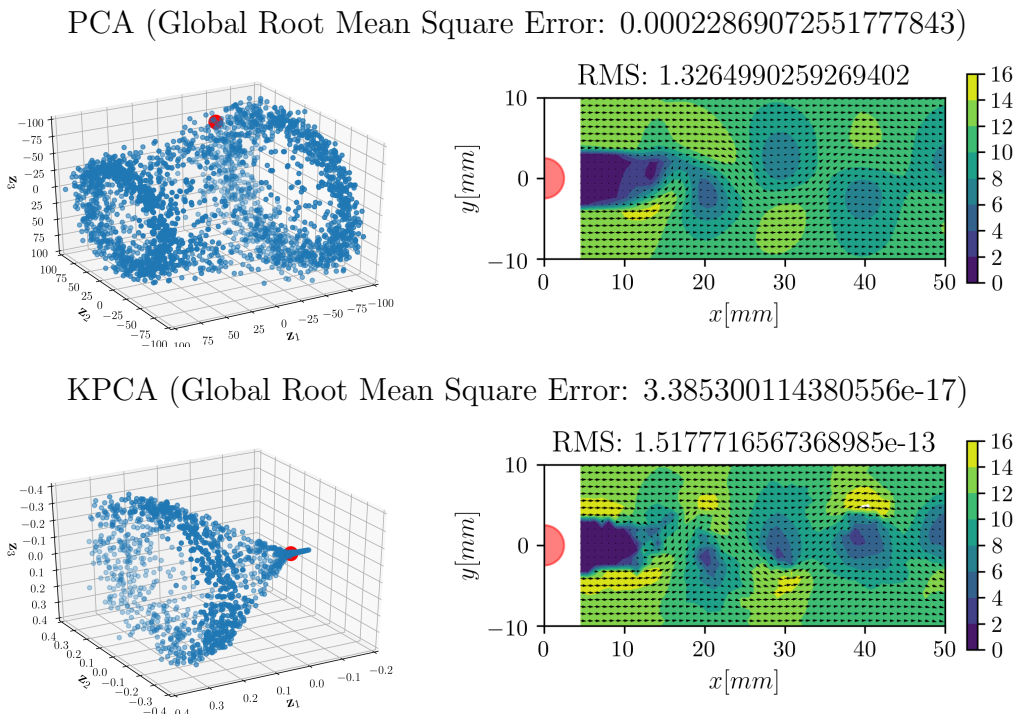

Figure 4: Linear and nonlinear dimensionality reduction via PCA and kPCA. The test case is the flow past a cylinder analyzed via TR-PIV in Mendez et al. (2020). The 3D scatter plots on the left shows the manifold identified projecting of the fields or the kernelized fields onto the first three PCA and kPCA components. The red points on these scatter plot correspond to the snapshot shown on the right. The global and instantaneous RMS are indicated for both approximation.

title of each contour plot. While the PCA does an excellent job in approximating the flow field with only three modes, the kPCA approximation is perfect, with the global RMS hitting machine precision. In other words, all the information contained in the trajectory of a system in $\mathbb{R}^{4260}$ has been mapped onto $\mathbb{R}^3$. Time series forecasting and modeling is much easier in $\mathbb{R}^3$ than in $\mathbb{R}^{4260}$.

5 From Optimal Flow Control to Reinforcement Learning

Flow control aims at interacting and manipulating flow systems to improve their engineering benefits. This is essential in countless applications, including drag or noise reduction, optimal wind energy extraction, stability of combustion systems or flight, and more el Hak (2000). In its most abstract formulation, the (flow) control problem is essentially a functional optimization problem constrained by the (fluid) systems' dynamicsStengel (1994). The goal is to optimize a function that measures the controller performances, i.e. its ability to keep the flow (the plant or the environment) close to the desired state (or dynamics). Although solid theoretical tools are available from optimal control theoryStengel (1994), the analytic derivation of these optimal laws is a formidable endeavour. To be practical, it is currently out of the reach for the most relevant problems in fluid mechanics.

Recent advances in machine learning are giving prominence to the so-called "black-

box" or "model-free" paradigm, whereby an algorithm learns the optimal control action by trial and error. A comprehensive survey of the most popular machine learning tools for this scope is presented by Pino *et al*Pino et al. (2021). This article presents a synopsis of Ref.Pino et al. (2021), considering only one algorithm from Reinforcement Learning and one simple illustrative test case: the problem of cancelling waves in a 1D advection equation.

The general framework of machine learning control methods is shown in Figure 5, on the left. One aims at controlling a dynamical system in an episodic framework. Within an episode (ep) of duration T, we interact with the system $N = T/\Delta t$ times, at regular intervals Δt, by performing a sequence of actions $\mathbf{a}_1, \mathbf{a}_2 \dots \mathbf{a}_N$ and observing the sequence of states $\mathbf{s}_1, \mathbf{s}_2 \dots \mathbf{s}_N$. We denote as $\mathbf{s}_k \in \mathbb{R}^{n_s}$ and $\mathbf{a}_k \in \mathbb{R}^{n_a}$ the state vector of the system and the action at the time step $t_k = k\Delta t$. The agent (aka controller) must learn a policy $\mathbf{a}_t = \pi(\mathbf{s}_k)$. This can be deterministic or stochastic, in which case the parametrization returns the parameters of a distribution from which the action is sampled.

We define a reward function $r_k = r(\mathbf{a_k}, \mathbf{s}_k) \in \mathbb{R}$ which measures the control performances and which is large when the agent is close to the objectives. At the end of each episode, the agent collects a cumulative reward $R = \sum_{k=1}^{N} \gamma^k r(\mathbf{a_k}, \mathbf{s}_k)$, with $\gamma \in [0, 1]$ a discount factor which prioritize immediate rewards. We give a value $V^\pi(\mathbf{s}_k)$ to a state as the cumulative rewards achievable from k to N following a certain policy. This is called *value function*. Similarly, we give a value for the state-action pair $Q^\pi(\tilde{\mathbf{a}}_k, \mathbf{s}_k)$, as the reward achievable if an action $\tilde{\mathbf{a}}_k$ is taken at the iteration k and the policy is followed for the rest of the episode. This is called *Q-function*. Both functions can be written in a recursive form. Assuming that the environment is stochastic (usually modeled as Markov Decision Process), the recursive form of the Q-function reads:

$$Q^\pi(\mathbf{s}_t, \tilde{\mathbf{a}}_t) = \mathbb{E}_{\sim \mathrm{ep}}\left[r(\tilde{\mathbf{a}_k}, \mathbf{s}_k) + \sum_{k=t+1}^{N} \gamma^{k-t} r(\mathbf{a_k}, \mathbf{s}_k) \right] = \mathbb{E}_{\sim \mathrm{ep}}\left[r(\tilde{\mathbf{a}_k}, \mathbf{s}_k) + \gamma Q^\pi(\mathbf{s}_t, \tilde{\mathbf{a}}_t) \right] \quad (12)$$

where $\mathbb{E}_{\sim \mathrm{ep}}$ is the expected value computed along the various episodes. This recursion is known as Bellman equation and is the foundation of dynamic programming.

We now have two paths to derive the optimal policy. In the so called *actor-only*, we seek to find the policy that maximized the cumulative rewards. That is, we look for the weights, in the parametric function $\mathbf{a}_t = \pi(\mathbf{s}_k, \mathbf{w}^\pi)$, that maximize the value of the initial state $V^\pi(\mathbf{s}_1) = \mathbb{E}_{\sim \mathrm{ep}}[r(\tilde{a_k}, \mathbf{s}_k) + \gamma V^\pi(\mathbf{s}_2)]$. In the so called *critic-only*, we take an indirect path: we focus on learning the value of each state and each action-state pair. We might thus use (12) to train a parametric function of the form $Q_t = Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{w}^Q)$. If this correctly estimates the Q-function, then the best action to take is simply the greedy $\mathbf{a}_t = \mathrm{argmax}_a Q(\mathbf{s}_t, \mathbf{a})$.

Modern RL algorithms are *actor-critic*, i.e. combine both approaches and employ two ANNs: one for learning the policy and the other to learn the Q-function. In this work, we illustrate the application of Deep Deterministic Policy Gradient (DDPG), by Lillicrap*et al* Lillicrap et al. (2015), which aims at learning a deterministic policy in an actor-critic framework. The essence of the algorithm is the calculation of the gradient $\nabla_{\mathbf{w}} V^\pi(\mathbf{s}_1)$ which is computed in batches of M interactions and reads:

$$\nabla_{\mathbf{w}} V^{\pi}(\mathbf{s}_1) \approx \frac{1}{M} \sum_i \nabla_a Q\big(\mathbf{s}_t, \mathbf{a}_t = \pi(\mathbf{s}_t, \mathbf{w}^{\pi}), \mathbf{w}^{Q}\big)\, \nabla_{\mathbf{w}}^{\pi} \pi(\mathbf{s}_t, \mathbf{w}^{\pi})\,, \tag{13}$$

where i is the index scanning the batch of interactions taken. The algorithm contains several other tricks, such as the implementation of a replay buffer collecting previous experience and the use of a sort of under-relaxation when updating the weights of both the policy and the Q networks. Details can be found in Lillicrap et al. (2015).

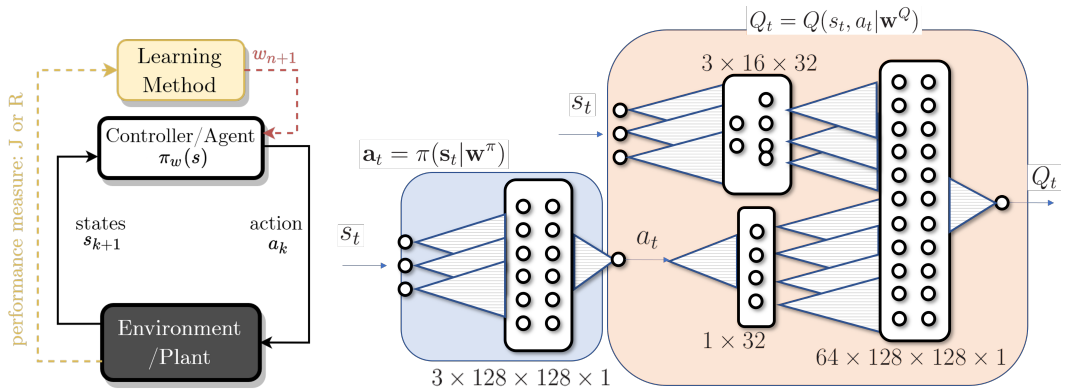

Figure 5: Application of Reinforcement Learning for flow control. The figure on the left shows the general idea: a learning method calibrate a controller which interacts with an environment via actions and collects rewards measuring its closeness to the objective. On the right: ANN architectures used in the DDPG implementation illustrated in this work. The network π (actor) parametrizes the state-action policy while the network π (critic) learns to estimate the value of each state-action pair.

The architecture implemented in this example is shown in Figure 5, on the right. The policy network is connected to the Q network. The environment to be controlled is a 1D linear advection equation subject to a disturbance and a control action:

$$\frac{\partial u}{\partial t} + 330 \frac{\partial u}{\partial x} = \underbrace{300 \sin(100\pi t) \cdot \mathcal{N}(x-5, 0.2)}_{\textit{disturbance}} + \underbrace{a_t \cdot \mathcal{N}(x-18.2, 0.2)}_{\textit{control}}, \tag{14}$$

over a domain $x \in [0, 50]$. Both sides of the domain are open (with non-reflecting conditions). The domain and a snapshot of the system is illustrated in Figure 6. The disturbance term consists of a Gaussian function multiplied by a sinusoid with given frequency and amplitude. The control term consist of an identical Gaussian placed downstream and having amplitude driven by the agent (controller).

The amplitude a_t constitutes the action of the agent. These are taken as a function of the system state s_t, which in this case consists of a vector of 3 points sampled at three locations. The agent's goal is to cancel the disturbance downstream of its location, i.e. achieving $u(x > 18.2) \approx 0$. The reward is defined as $r_t = -||u_O(t,x)||_2$, with u_O the sampled solution in the interval x_1^*-x_2^*, also indicated in Figure 6.

The architecture of the two ANNs is also detailed in Figure 5, on the right. The policy is an ANN with three inputs, two hidden layers of 128 neurons and one output. Relu activation functions are used in all layers. The Q network has 4 inputs and two

joining layers that merge into two hidden layers of 128 neurons each. More details on the implemented structures are available in Pino et al. (2021).

Figure 6 shows three snapshots of the solution at three episodes, namely **ep** $= 2, 5, 15$. The control performance is excellent within dozens of episodes, and the incoming waves are nearly entirely cancelled. While it is easy to show that a linear controller could easily solve this problem, no information about the physics of the system has been used in the RL approach.

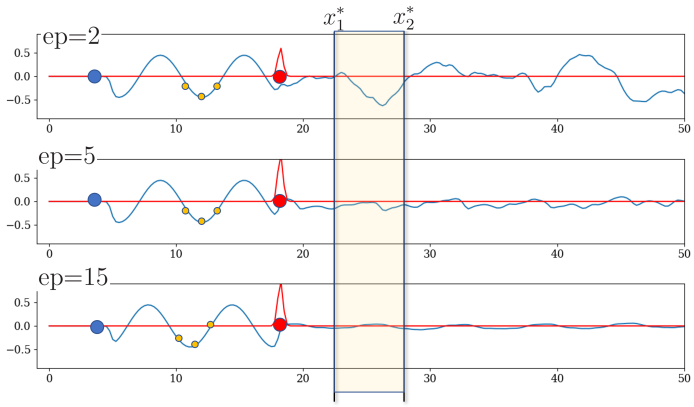

Figure 6: Results of the DDG implementation for a wave cancelling problem. Waves are produced at $x = 5$ (blue marker) and travel downstream $x \to \infty$. A controller is located at $x = 18.2$ (red marker) and seeks to cancel the waves by creating destructive interference. The controller is informed by three sensors (orange markers), and its performances are measured in the interval $[x_1^*, x_2^*]$ in terms of l_2 norm of the solution. The figure shows three snapshots at three episodes ep $= 2, 5, 15$. In a dozen episodes, the control is excellent.

6 Conclusions and Perspectives

This talk explored avenues for machine learning in fluid mechanics. Machine learning methods are designed to learn functions from data without leveraging on first principles. The fundamentals of supervised, unsupervised and reinforcement learning were briefly reviewed, and several representative algorithms were tested on problems of fluid mechanics.

Within the supervised learning category, deep learning and genetic programming were shown to outperform classic tools in tasks such as thermal turbulence modelling and noise prediction. These tools can be seen as adaptive function approximators, and care is needed in introducing physical priors in their calibration. This can be done upfront, when constructing the learning architecture, or during the learning, incorporating them in the cost function and its constraints. This is a natural approach considering that most machine learning models (ANN, RBFs, GP trees) are easily differentiable and analytic constraints in the form of PDEs can be easily included in the learning process. Given their predictive capabilities and the availability of powerful open-source libraries for their implementation, it is believed that ANN will become essential tools in the toolbox of the fluid dynamicist.

Besides allowing for "informing" the regression about physical priors, the differentiability of machine learning models also enables the meshless integration of PDEs. An

example using RBFs to compute pressure fields from LPT measurements was illustrated. These methods allow avoiding the difficulties of meshing and numerical integration in the presence of noise, limited resolution and complex geometries, as in the case in the pressure integration from image velocimetry. In the authors' opinion, these tools will play a prominent role in data assimilation and experimental fluid mechanics.

Within the unsupervised learning category, linear and nonlinear tools for dimensionality reduction were showcased. While the fluid mechanic's community has vast experience on linear methods (and has pioneered some of the most powerful decompositions), the use of nonlinear methods is in its infancy. The provided illustration showed the vastly superior 'data compression' performances of the kPCA over the PCA on a TR-PIV dataset. In tasks that do not demand physical interpretation, such as data compression or filtering, nonlinear tools and manifold learning techniques will likely become the standard tools. These are also precious pre-processing tools for classification and time series forecasting. On the other hand, the performances of nonlinear tools are very sensitive to hyper-parameters (e.g. the kernel parameters), and the identified manifolds might be hard to interpret.

Finally, the bridge between optimal control theory and reinforcement learning (RL) was highlighted. RL algorithms are designed to solve decision problems via trial and error and have proved capable of outstanding achievements in board games. In the provided example and the recent literature, successful applications of RL for flow control have been illustrated. These entitle the community to hope that a RL agent might one day learn control strategies in the same way it can nowadays learn how to play chess at superhuman levels. In such a scenario, which today appears on the edge of science fiction for fluid mechanics, the usual role might be inverted, and the computer (agent) would become our supervisor. This is already happening in board games. But the problems of controlling turbulence is vastly more complex than any board game, and RL appears somewhat a less mature field than supervised or unsupervised learning. New training agents are published every year and, at the time of writing, none of the most popular algorithms (DDPG, PPO or A3C) is older than six years. The field is open and is looking for a new generation of engineers trained in both fluid mechanics *and* machine learning.

References

Alexander Zai, B. B. (2020). *Deep Reinforcement Learning in Action*. Manning Publications.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Bakir, G., Weston, J., and Bernhard, S. (1999). Learning to find pre-images. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*.

Banzhaf, W., Nordin, P., and Keller, R. E. (1997). *Genetic Programming: An Introduction*. MORGAN KAUFMANN PUBL INC.

Bashir, S. M. A., Wang, Y., Khan, M., and Niu, Y. (2021). A comprehensive review of deep learning-based single image super-resolution. *PeerJ Computer Science*, 7:e621.

Beintema, G., Corbetta, A., Biferale, L., and Toschi, F. (2020). Controlling rayleigh–bénard convection via reinforcement learning. *Journal of Turbulence*, pages 1–21.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc.

Borée, J. (2003). Extended proper orthogonal decomposition: a tool to analyse correlated events in turbulent flows. *Exp. Fluids*, 35(2):188–192.

Brenner, M. P., Eldredge, J. D., and Freund, J. B. (2019). Perspective on machine learning for advancing fluid mechanics. *Physical Review Fluids*, 4(10).

Brunton, S. L., Noack, B. R., and Koumoutsakos, P. (2020). Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937.

Bucci, M. A., Semeraro, O., Allauzen, A., Wisniewski, G., Cordier, L., and Mathelin, L. (2019). Control of chaotic systems by deep reinforcement learning. *Proceedings of the Royal Society A*, 475(2231):20190351.

Buhmann, M. D. (2003). *Radial Basis Functions*. Cambridge University Press.

Clauser, F. H. (1954). Turbulent boundary layers in adverse pressure gradients. *Journal of the Aeronautical Sciences*, 21(2):91–108.

Coles, D. (1956). The law of the wake in the turbulent boundary layer. *Journal of Fluid Mechanics*, 1(2):191–226.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

Daniel, T., Casenave, F., Akkari, N., and Ryckelynck, D. (2020). Model order reduction assisted by deep neural networks (ROM-net). *Advanced Modeling and Simulation in Engineering Sciences*, 7(1).

Deuse, M. and Sandberg, R. D. (2020). Different noise generation mechanisms of a controlled diffusion aerofoil and their dependence on mach number. *Journal of Sound and Vibration*, 476:115317.

Dominique, J., Christophe, J., Schram, C., and Sandberg, R. D. (2021). Inferring empirical wall pressure spectral models with gene expression programming. *Journal of Sound and Vibration*, 506:116162.

Dominique, J., den Berghe, J. V., Schram, C., and Mendez, M. A. (2022). Artificial neural networks modelling of wall pressure spectra beneath turbulent boundary layers. *arXiv:2201.03262*.

Duraisamy, K., Gianluca, I., and Xiao, H. (2019). Turbulence modeling in the age of data. *Annual review of Fluid Mechanics*, 51:357–377.

Ehlert, A., Nayeri, C. N., Morzynski, M., and Noack, B. R. (2020). Locally linear embedding for transient cylinder wakes.

el Hak, M. G. (2000). *Flow Control*. Cambridge University Press.

Fiore, M., Koloszar, L., Mendez, M. A., Duponcheel, M., and Bartosiewicz, Y. (2021). A physics-oriented data-driven approach to thermal turbulence modelling. In *Proceedings of the 15th International Conference on Heat Transfer, Fluid Mechanics and Thermodynamics*.

Fiore, M., Koloszar, L., Mendez, M. A., Duponcheel, M., and Bartosiewicz, Y. (2022). Artificial neural networks for the modeling of the turbulent heat flux in forced convection flows. In *Proceedings of the 19th International Topical Meeting on Nuclear Reactor Thermal Hydraulics (NURETH)*.

Fornberg, B. and Flyer, N. (2015). Solving PDEs with radial basis functions. *Acta Numerica*, 24:215–258.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goody, M. (2004). Empirical spectral model of surface pressure fluctuations. *AIAA journal*, 42(9):1788–1794.

Hernandez-Leal, P., Kartal, B., and Taylor, M. E. (2019). A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797.

Hesthaven, J. and Ubbiali, S. (2018). Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78.

Hobold, G. M. and da Silva, A. K. (2018). Machine learning classification of boiling regimes with low speed, direct and indirect visualization. *International Journal of Heat and Mass Transfer*, 125:1296–1309.

Holmes, P. J., Lumley, J. L., Berkooz, G., Mattingly, J. C., and Wittenberg, R. W. (1997). Low-dimensional models of coherent structures in turbulence. *Phys. Rep.*, 287(4):337–384.

Huang, S.-C. and Kim, J. (2008). Control and system identification of a separated flow. *Physics of Fluids*, 20(10):101509.

Hussain, A. K. M. F. (1983). Coherent structures—reality and myth. *Physics of Fluids*, 26(10):2816.

Ilak, M. and Rowley, C. W. (2008). Modeling of transitional channel flow using balanced proper orthogonal decomposition. *Physics of Fluids*, 20(3):034103.

Kaiser, E., Noack, B. R., Cordier, L., Spohn, A., Segond, M., Abel, M., Daviller, G., Östh, J., Krajnović, S., and Niven, R. K. (2014). Cluster-based reduced-order modelling of a mixing layer. *Journal of Fluid Mechanics*, 754:365–414.

Kang, M., Hwang, L. K., and Kwon, B. (2020). Machine learning flow regime classification in three-dimensional printed tubes. *Physical Review Fluids*, 5(8).

Kim, S. H. and Boukouvala, F. (2019). Machine learning-based surrogate modeling for data-driven optimization: a comparison of subset selection for regression techniques. *Optimization Letters*, 14(4):989–1010.

Kober, J. and Peters, J. (2014). Reinforcement learning in robotics: A survey. In *Springer Tracts in Advanced Robotics*, pages 9–67. Springer International Publishing.

Lagaris, I., Likas, A., and Fotiadis, D. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000.

Lee, S. (2018). Empirical wall-pressure spectral modeling for zero and adverse pressure gradient flows. *AIAA Journal*, 56(5):1818–1829.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning.

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. (2019). A survey of reinforcement learning informed by natural language.

Majors, J. H. (2018). Machine learning identifies flow patterns faster and better with a combined analysis approach. *Scilight*, 2018(3):030007.

Mendez, M., Raiola, M., Masullo, A., Discetti, S., Ianiro, A., Theunissen, R., and Buchlin, J.-M. (2017). POD-based background removal for particle image velocimetry. *Experimental Thermal and Fluid Science*, 80:181–192.

Mendez, M. A., Balabane, M., and Buchlin, J.-M. (2019). Multi-scale proper orthogonal decomposition of complex fluid flows. *Journal of Fluid Mechanics*, 870:988–1036.

Mendez, M. A., Hess, D., Watz, B. B., and Buchlin, J.-M. (2020). Multiscale proper orthogonal decomposition (mPOD) of TR-PIV data—a case study on stationary and transient cylinder wake flows. *Measurement Science and Technology*, 31(9):094014.

Mendez, M. A., Ianiro, A., Noack, B. R., and Brunton, S. L. (2022). *Data-Driven Fluid Mechanics: Combining First Principles and Machine Learning*. Cambridge University Press (in Press).

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Education.

Murata, T., Fukami, K., and Fukagata, K. (2019). Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. *Journal of Fluid Mechanics*, 882.

Noack, B. R., Afanesiev, K., Morzyński, M., Tadmor, G., and Thiele, F. (2003). A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *Journal of Fluid Mechanics*, 497:335–363.

Novati, G. and Koumoutsakos, P. (2019). Remember and forget for experience replay. In *Proceedings of the 36th International Conference on Machine Learning*.

Novati, G., Mahadevan, L., and Koumoutsakos, P. (2019). Controlled gliding and perching through deep-reinforcement-learning. *Phys. Rev. Fluids*, 4(9).

Novati, G., Verma, S., Alexeev, D., Rossinelli, D., van Rees, W. M., and Koumoutsakos, P. (2017). Synchronisation through learning for two self-propelled swimmers. *Bioinspir. Biomim.*, 12(3):036001.

Pan, S. and Duraisamy, K. (2018). Long-time predictive modeling of nonlinear dynamical systems using neural networks. *Complexity*, 2018:1–26.

Pino, F., Schena, L., Rabault, J., Kuhnle, A., and Mendez, M. A. (2021). Comparative analysis of machine learning methods for flow active control. *To appear in Phys of Fluids*.

Pivot, C., Cordier, L., and Mathelin, L. (2017). A continuous reinforcement learning strategy for closed-loop control in fluid dynamics. In *35th AIAA Applied Aerodynamics Conference*. American Institute of Aeronautics and Astronautics.

Rabault, J., Kuchta, M., Jensen, A., Réglade, U., and Cerardi, N. (2019). Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302.

Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

Rao, C., Sun, H., and Liu, Y. (2020). Physics-informed deep learning for incompressible laminar flows. *Theoretical and Applied Mechanics Letters*, 10(3):207–212.

Reddy, G., Celani, A., Sejnowski, T. J., and Vergassola, M. (2016). Learning to soar in turbulent environments. *Proceedings of the National Academy of Sciences*, 113(33):E4877–E4884.

Renganathan, S. A., Maulik, R., and Rao, V. (2020). Machine learning for nonintrusive model order reduction of the parametric inviscid transonic flow past an airfoil. *Physics of Fluids*, 32(4):047110.

Richard S. Sutton, A. G. B. (2018). *Reinforcement Learning*. MIT Press Ltd, Second Edition.

Rumelhart, D. E. and McClelland, J. L. (1989). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT PRess.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Schölkopf, B., Smola, A., and Müller, K.-R. (1997). Kernel principal component analysis. In *Lecture Notes in Computer Science*, pages 583–588. Springer Berlin Heidelberg.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Sperotto, P., Pieraccini, S., and Mendez, M. A. (2021). A meshless method to compute pressure fields from image velocimetry.

Stengel, R. F. (1994). *Optimal control and estimation*. Courier Corporation.

Swets, D. and Weng, J. (1996). Using discriminant eigenfeatures for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):831–836.

Szita, I. (2012). Reinforcement learning in games. In *Adaptation, Learning, and Optimization*, pages 539–577. Springer Berlin Heidelberg.

Taira, K., Brunton, S. L., Dawson, S. T. M., Rowley, C. W., Colonius, T., McKeon, B. J., Schmidt, O. T., Gordeyev, S., Theofilis, V., and Ukeiley, L. S. (2017). Modal analysis of fluid flows: An overview. *AIAA J.*, 55(12):4013–4041.

Verma, S., Novati, G., and Koumoutsakos, P. (2018). Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854.

Wang, L., Fortunati, S., Greco, M. S., and Gini, F. (2018). Reinforcement learning-based waveform optimization for MIMO multi-target detection. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE.