Physica D 414 (2020) 132736

Contents lists available at ScienceDirect

Physica D

journal homepage: www.elsevier.com/locate/physd

Bridging the gap: Machine learning to resolve improperly modeled dynamics☆

Maan Qraitem [a,*], Dhanushka Kularatne [b], Eric Forgoston [c], M. Ani Hsieh [b]

[a] Department of Computer Science, Colby College, Waterville, ME 04901, USA
[b] Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104, USA
[c] Department of Applied Mathematics and Statistics, Montclair State University, Montclair, NJ 07043, USA

ARTICLE INFO

Article history:
Received 31 December 2019
Received in revised form 13 August 2020
Accepted 7 September 2020
Available online 11 September 2020
Communicated by B. Hamzi

Keywords:
Machine learning
Data-driven modeling
Neural networks
Nonlinear dynamical systems
Long Short-Term Memory (LSTM)

ABSTRACT

We present a data-driven modeling strategy to overcome improperly modeled dynamics for systems exhibiting complex spatio-temporal behaviors. We propose a Deep Learning framework to resolve the differences between the true dynamics of the system and the dynamics given by a model of the system that is either inaccurately or inadequately described. Our machine learning strategy leverages data generated from the improper system model and observational data from the actual system to create a neural network to model the dynamics of the actual system. We evaluate the proposed framework using numerical solutions obtained from three increasingly complex dynamical systems. Our results show that our system is capable of learning a data-driven model that provides accurate estimates of the system states both in previously unobserved regions as well as for future states. Our results show the power of state-of-the-art machine learning frameworks in estimating an accurate prior of the system's true dynamics that can be used for prediction up to a finite horizon.

1. Introduction

Recent breakthroughs in machine learning (ML) and artificial intelligence (AI) have shown a remarkable ability to extract relationships and correlations in data and events. Indeed, there now exist highly scalable solutions for object detection and recognition, machine translation, text-to-speech conversion, recommender systems, and information retrieval. Recent advances in machine learning and data analytics have yielded transformative results across diverse scientific disciplines [1–5]. Enabled by the decreasing price to performance ratio of sensing, data storage, and computational resources in the past decade, data-driven machine learning strategies are taking center stage across many scientific disciplines.

In the realm of complex spatiotemporal dynamical systems, data-driven machine learning strategies have been employed for reduced-order models (ROMs) [6–10], discovery of system dynamics [11–22], computation of dynamical system solutions [23–28], and prediction of future dynamics [26,28–32]. These recent developments spurred by the current enthusiasm surrounding ML and AI strategies can be broadly classified into two categories: works that investigate the feasibility of existing ML/AI algorithms and architectures, and those centered around the development of new algorithms and architectures. Existing work whose main objective is the former have focused on the power of ML/AI techniques to significantly reduce the steep computation and data storage costs associated with high-fidelity computational fluid dynamics (CFD) efforts [6–10,26,29–32]. These works often leverage existing CFD models to generate ground truth, training, and testing datasets to evaluate well-studied convolutional neural networks (CNN) [6,30,32], long short-term memory (LSTM) networks [9], generative adversarial networks (GAN) [10], and existing ML/AI frameworks [8,29,31]. Nevertheless, existing ML/AI strategies are predicated on access to large amounts of *labeled* data where explicit knowledge derived from well-established first principles are difficult to encode.

Works in the second category that directly address these challenges include sparse regression techniques [12,15–18,22] and physics-informed neural networks (PINNs) [23–26]. Sparse identification is a data-driven system identification strategy that balances model complexity with descriptiveness [17]. Since the dynamics of most physical systems are governed by only a few important terms [17], sparse identification selects from a finite set of candidate dictionary functions whose linear combination describes the system dynamics [16]. On the other hand, PINNs are neural networks that are trained to solve supervised

☆ We gratefully acknowledge the support of ONR, USA Award No. 14-19-1-2253 and NSF, USA DUE Award 1839686.

* Corresponding author.
E-mail addresses: mqrait20@colby.edu (M. Qraitem), dkul@seas.upenn.edu (D. Kularatne), eric.forgoston@montclair.edu (E. Forgoston), m.hsieh@seas.upenn.edu (M.A. Hsieh).

https://doi.org/10.1016/j.physd.2020.132736
0167-2789/

learning tasks whose dynamics can be described by general nonlinear PDEs. The key advantage of PINNs is their *data-efficiency* in the training phase. Sparse regression techniques such as those found in [12,15–18,22] require large amounts of relatively clean data to accurately compute numerical gradients, whereas PINNs do not require any data on gradients of the flow field (nor their numerical approximations). As such, PINNs perform more robustly when data is sparse and/or noisy relative to the complexity of the underlying system dynamics [23,24]. In contrast, Ayed et al. ([21]) use actual observations of a system whose dynamics are given by an ordinary differential equation to train the neural network weights. Once trained, the network provides an equation-free model representation of the system dynamics. Different from [12,15–18,22–24], the work does not directly address the issue of data-efficiency but assumes the network has access to a sufficiently large set of training data.

In this work, we take inspiration from [12,15,17,19,20,22–24,28] and present a data-driven Deep Learning framework capable of resolving the differences between the actual dynamics of a complex nonlinear system and that of the same system which has been improperly or inaccurately modeled. Given an inaccurate or inadequate model of a system, our proposed ML strategy combines data from this inaccurate/inadequate model with observational data from the actual system to learn the dynamics of the actual system. The result is a neural network model that can accurately estimate the system states in regions with no observations and/or provide predictions for future states. Different from [12,15,17,22–24], our approach provides an equation-free representation of the system dynamics that successfully estimates the underlying physics that drives the process. We evaluate the proposed framework using three different dynamical systems each with increasing complexity. Our results show how the proposed strategy is not only capable of resolving improperly or inaccurately modeled dynamics but also can learn the dynamics of the actual system and provide accurate future predictions.

While our approach is similar to [28,33], we make use of LSTMs in our deep learning network rather than a simple multilayer perceptron [33] or reservoir computer [28]. Our approach is general and may be used for a wide range of dynamical systems of different dimension and complexity, including examples in which the known model is missing external forcing functions or other known dynamics. Even for these complicated scenarios, we demonstrate in this article the power of our method to successfully predict the dynamics wherein simpler approaches will fail. Since our output is a neural network representation of the system model, the output of our network can be fed into existing data-driven model discovery techniques [11,13,14,16,17] to obtain closed-form equation representations of the dynamical system.

The paper is organized as follows: we list our assumptions and provide a concise formulation of our problem in Section 2. The design of the network architecture and our methodology is described in Section 3. We discuss how we evaluate our methodology in Fig. 3 and present our results with discussion in Section 5. Conclusions and directions for future work are contained in Section 6.

2. Problem formulation

We consider a spatio-temporal process $u(x,t) \in \mathbb{R}^m$, where $x \in \mathbb{W}$ represents a point in the environment $\mathbb{W} \subset \mathbb{R}^n$ and $t \in [t_s,\ t_f]$ represents the time within an observation interval of interest. The actual model of the process that governs u is denoted by M_{act} and is given by a partial differential equation (PDE) of the form

$$u_t = \mathcal{N}[u, f_1, \ldots, f_p, g_1, \ldots, g_r], \tag{1}$$

where $\mathcal{N}[\cdot]$ is a nonlinear differential operator, where $f_i = f_i(x,t) \in \mathbb{R}^{n_{f_i}},\ i = 1,\ldots,p$ and $g_i = g_i(x,t) \in \mathbb{R}^{n_{g_i}},\ i = 1,\ldots,r$ are external phenomena that impact u. Let M_{curr} denote the model that is obtained from the current understanding of the physics of u. Then M_{curr} is given by the PDE with form

$$u_t = \tilde{\mathcal{N}}[u, f_1, \ldots, f_p], \tag{2}$$

where $\tilde{\mathcal{N}}[\cdot]$ is also a nonlinear differential operator. Here, the f_i denote the p external phenomena whose impact on u is currently *known* and the g_i denote the r external phenomena that affect u but are not captured in M_{curr}. Note that in general, g_i could represent some error in f_i so that $g_i = f_i + \epsilon$ where ϵ denotes the difference between f_i and g_i. Furthermore, $\tilde{\mathcal{N}}$ is used to denote any differences in system parameters between M_{curr} and M_{act}. Thus, while M_{curr} represents the current understanding of the process, this understanding is incomplete or inadequate and thus M_{curr} is not an accurate representation of the process model.

Given a set of coordinates $S = \{s_j | s_j = (x_j, t_j),\ x_j \in \mathbb{W},\ t_j \in [t_s,\ t_f],\ j = 1,\ldots,n_{data}\}$, let $\hat{U}_{act} = \{\hat{u}_{act_j} | j = 1,\ldots,n_{data}\}$ be the set of observations of u obtained by measuring the actual process at coordinates $s_j \in S$. Similarly, let $U_{curr} = \{u_{curr_j}\}$ and $U_{act} = \{u_{act_j}\}$ be the solution sets obtained from M_{curr} and M_{act} respectively, at the coordinates in S. In this work, U_{act} is based on computer simulations, but could in fact be measured experimentally. For simplicity, we assume that there are no measurement errors, *i.e.*, $\hat{u}_{act_j} \equiv u_{act_j}$ for each $\hat{u}_{act_j} \in \hat{U}_{act}$ and $u_{act_j} \in U_{act}$ obtained at the same coordinate $s_j \in S$.

Given U_{act}, U_{curr} and observations of a subset of the g_i at the coordinates in S, the objective of this work is to develop a neural network based model M_{nn} that better estimates the process u in and potentially beyond the space–time domain $\mathbb{W} \times [t_s,\ t_f]$. Let $e_* = \|M_{act} - M_*\| \geq 0$ represent some measure of the error of the output of a given model with respect to the output of M_{act} in a given domain. We want $e_{nn} \leq e_{curr}$ in all domains (ideally $e_{nn} = e_{curr}$ only when $e_{curr} = 0$), *i.e.*, the neural network should be much better at predicting/estimating u than the existing model.

To illustrate, consider a mass–spring–damper system with mass m, damping coefficient c and spring constant k that is subjected to two external forcing functions given by $F_1(t) = A_1\cos(\omega_1 t)$ and $F_2(t) = A_2\cos(\omega_2 t)$. If the displacement of the mass is denoted by y, the actual model of the system M_{act} is given by the ordinary differential equation (ODE)

$$m\ddot{y} + c\dot{y} + ky = F_1 + F_2. \tag{3}$$

Let us assume that due to modeling and measurement errors, the model that we have access to, M_{curr}, is given by

$$\tilde{m}\ddot{y} + \tilde{c}\dot{y} + \tilde{k}y = F_1. \tag{4}$$

Note that this model only captures part of the forcing function and has errors in the mass, spring, and damping coefficients. Given measurements of the displacement y, our work seeks to develop a neural network, whose output closely resembles that of the actual model M_{act} for the same initial conditions. Denoting the output of the actual, current and neural network models by $y_{act}(t)$, $y_{curr}(t)$ and $y_{nn}(t)$ respectively, we would like $\|y_{nn}(t) - y_{act}(t)\| < \epsilon < \|y_{curr}(t) - y_{act}(t)\|$, where ideally ϵ is small. In other words, we would like the trained neural network output to always be a better approximation of the ground truth than the current model output or match the ground truth exactly. Lastly, in our proposed framework, the neural network model M_{nn} only provides outputs for the ODE, *e.g.*, y, $\dot{y}$, and $\ddot{y}$ rather than the equation of the actual ODE.

3. Methodology

The proposed method uses a neural network based framework to "bridge the gap" between M_{curr} and M_{act}. Neural networks have recently been used in a plethora of prediction and estimation problems. However, in most of these solutions, large quantities of training data are required to obtain good prediction performance. This is especially true for prediction/estimation problems involving complex dynamical systems. In this work, we mitigate this data inefficiency problem by incorporating existing knowledge of the process into the neural network architecture.

The fundamental hypothesis of our work is that the current understanding of the physics of u given by M_{curr}, has substantial information that the neural network can exploit in order to provide better predictions of the process. Thus, in addition to the space–time coordinates $(x,\ t)$ and, where applicable, external forcing terms g_i, we also use the output from M_{curr} as an input to the neural network. This input may be presented to the network in different formats, e.g., data generated from a reduced-order model [34,35], coefficients and functions from a sparse identification of the process [17,22], output data from a numerical model, etc.

Furthermore, the behavior of any dynamical system depends heavily on the initial and boundary conditions. In the absence of explicit initial and boundary conditions, these spatio-temporal dependencies have to be captured by the network in a purely data-driven manner. We facilitate this by (1) using Long Short-Term Memory (LSTM) stages in our network to capture temporal dependencies, and (2) providing the network with data in a space–time hypercube around the point of interest.

Neural networks and LSTM networks

Artificial neural networks (ANN) are powerful nonlinear statistical models which consist of multiple layers of interconnected nodes such that every connection represents a weight. Each node calculates a weighted sum of the outputs of neurons which are connected to it as well as a bias term. By representing the system in terms of layers, neural networks are able to learn features exhibited by highly nonlinear and complex data in a powerful hierarchical fashion. The nonlinearity of these networks comes from the use of nonlinear activation functions in the neural net nodes. The neural net is trained by minimizing a loss function. The minimization is commonly done by a gradient-based optimization algorithm that makes use of backpropagation – a computationally efficient algorithm that computes the gradient of the loss function with respect to the weights at each layer. Common optimization algorithms include stochastic gradient descent, Adam [36], and Adagrad [37]. The optimization algorithm commonly performs updates to the weights using batches of the dataset. A complete pass through all the dataset batches is usually referred to as an *epoch*.

The most basic structure of a neural network is a fully connected or dense ANN as displayed in Fig. 1. Each node in the neural network is governed by an activation function $a_{l+1}(W_l a_l + b_l)$ where W_l and b_l denote the weights matrix and bias vector for layer l respectively. Common choices for a_{l+1} include the sigmoid function commonly denoted by $\sigma(\cdot)$, the hyperbolic tangent function $tanh(\cdot)$, and rectified linear unit function $ReLU(\cdot)$. We refer the reader to [38] for a detailed review of activation functions.

In choosing a neural network architecture, we make note that our problem is in nature time-dependent. More concretely, the problem imposes an order on the sequence of observations that must be preserved. In general, standard artificial neural networks are not well-suited to learn such orders since the weights in each ANN layer are fully connected to the previous layer. This forces the ANN to consider the entire sequence at once. Recurrent

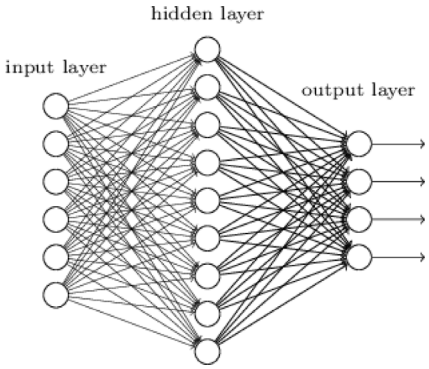


Fig. 1. A general dense layer architecture.

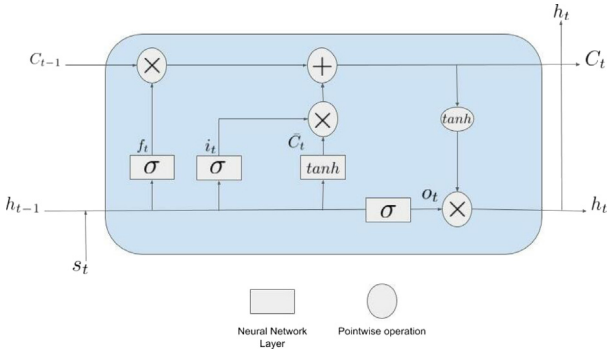


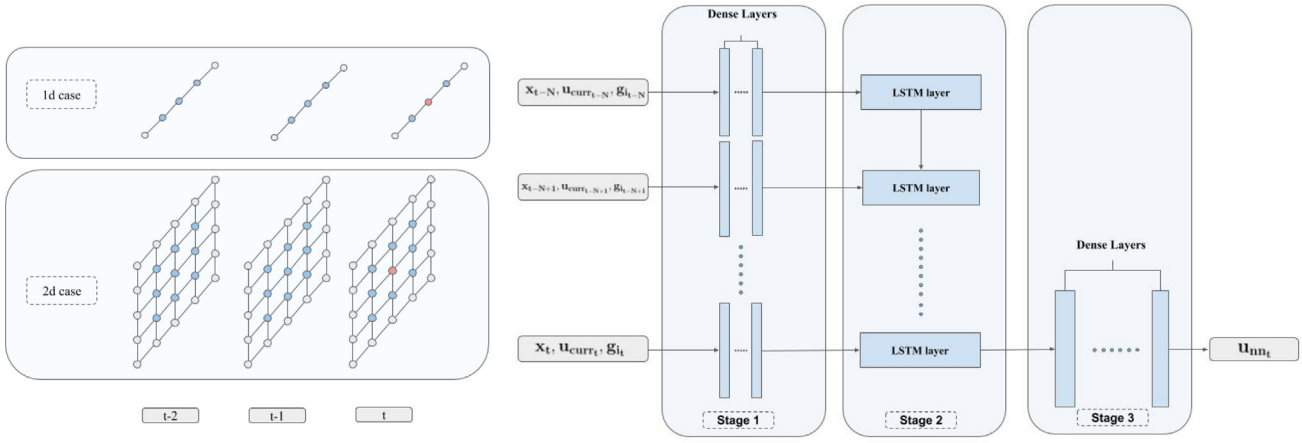
Fig. 2. A general architecture for an LSTM layer.

Neural Networks (RNNs), on the other hand, are a different type of neural network that is well suited for sequence learning problems. They are equipped with a memory unit which is updated for each new observation. Thus, parameters of the network are shared for each step in the sequence. As such, RNNs rather than ANNs are most commonly employed to learn time dependencies.

The Long Short-Term Memory (LSTM) network is a variant of RNNs. LSTMs address the bottlenecks in traditional RNNs such as the vanishing gradient problem [39] which hampers learning of long data sequences. The LSTM memory unit is usually called the cell, denoted by C, which is regulated by three gates: an input gate $\mathcal{I}$, a forget gate $\mathcal{F}$, and an output gate $\mathcal{O}$. The input gate controls the contribution of the input to the cell, the forget gate controls what parts of the cell to keep, and the output gate controls the contribution of the cell to the output of the LSTM. A schematic of the architecture can be found in Fig. 2, with h representing the output of the network while the input of the network is represented with s. The equations to compute the gates and states are given by

$$\begin{aligned}
\mathcal{F}_t &= \sigma(W_{\mathcal{F}} \cdot [h_{t-1}, s_t] + b_{\mathcal{F}}),\\
\mathcal{I}_t &= \sigma(W_{\mathcal{I}} \cdot [h_{t-1}, s_t] + b_{\mathcal{I}}),\\
\bar{C}_t &= tanh(W_C \cdot [h_{t-1}, s_t] + b_C),\\
C_t &= \mathcal{F}_t * C_{t-1} + \mathcal{I}_t * \bar{C}_t,\\
\mathcal{O}_t &= \sigma(W_{\mathcal{O}} \cdot [h_{t-1}, s_t] + b_{\mathcal{O}}),\\
h_t &= \mathcal{O}_t * tanh(C_t),
\end{aligned} \tag{5}$$

where $\bar{C}$ is the updated state, W is the weights matrix, b is the bias vector for each gate, s_t is the input to the network at time t, and $*$ denotes the Hadamard product. The forget gate reduces overfitting by controlling how an incoming input contributes to the hidden state. This structure is the key reason why LSTMs



(a) (b)

Fig. 3. (a) Format of the inputs to the network. For each input to the network, we consider data in a $n+1$ hypercube around the point of interest. Along each dimension, k data points are included resulting in k^{n+1} number of data points for each input. In this figure, $k=3$. (b) The general architecture of our final model which is composed of three stages. The number of layers and nodes in each stage depends on the problem.

do not suffer from the vanishing gradient problem exhibited by RNNs. For more detailed discussions on ANNs, RNNs, and LSTMs, we refer the interested reader to [40–43].

Input data format to the network

The network predicts/estimates the process on a point by point basis. In order to capture the spatio-temporal dependencies between the inputs and the output at each coordinate j, we consider a $n+1$ dimensional space–time hypercube of the inputs around this coordinate. We consider k data points along each dimension, resulting in k^{n+1} number of data points for each input. In general, the larger the choice of k, the larger the input data and thus the higher the computational load. In this work, we choose $k=3$ to limit the computational burden. Thus for scenarios where $n=2$, as shown in Fig. 3(a), we would consider a hypercube with 27 vertices for each input.

3.1. Architecture of the neural network

Our proposed neural network architecture is composed of three stages as shown in Fig. 3(b). We modify the architecture for each problem by changing the number of layers/nodes at different stages of the architecture. The three stages of the network are:

- Stage 1: Time distributed dense stage with D_1 layers;
- Stage 2: Long Short-Term Memory (LSTM) stage with D_2 layers; and
- Stage 3: Dense output stage with D_3 layers.

The three stages are described below in detail.

Stage 1: Time distributed dense layers

This stage consists of a set of parallel dense layers that work on the inputs at each time slice independently. The purpose of this stage is to give the network the ability to pre-process the data and learn a representation that is most optimal for the LSTM stage. While most research in the literature employing LSTM networks do so without this pre-processing layer, our experiments have demonstrated that adding this stage improves the convergence of the network. The activation function for layer l in this stage is denoted as $a_{l,t}$ where t denotes the time step. In this case, W and b are shared for each time step. The output of this layer is then passed to Stage 2.

Stage 2: LSTM stage

The LSTM is a type of Deep Learning architecture that is designed to exploit long term dependencies in time series data. Given the nature of dynamical systems data where time-based dependencies are abundant, LSTMs are a powerful choice to model such data. Thus, after the data has been processed by a sequence of dense layers in Stage 1, we apply a sequence of LSTM layers in Stage 2. The equations for the LSTM layer are given by Eq. (5) with s_t replaced by $a_{L,t}$, where L is the number of the last layer in Stage 1. The output of the LSTM layer from the final time step is then used as the input to the Stage 3.

Stage 3: Dense output stage

Stage 3 consists of a sequence of dense layers. This stage serves as a final stop for processing the data before producing the output. The output of the last dense layer is the final predicted output u_{nn} from the neural network. The output of the network is used in the following loss function to train the network

$$Loss(u_{act}, u_{nn}) = \frac{1}{M}\sum_{i=1}^{M}(u_{act_i} - u_{nn_i})^2, \tag{6}$$

where M is the dimension of the output u.

4. Methodology evaluation

To quantitatively and qualitatively evaluate our methodology, we consider different dynamical systems each with increasing complexity. The proposed learning framework is evaluated with respect to its ability to reproduce the dynamics of the actual system and its ability to predict future observations on a point-by-point basis.

4.1. Candidate systems

We consider three candidate systems to test our hypothesis on, with each system being progressively more complex. Each candidate system exhibits one of the three types of differences between M_{act} and M_{curr}: (1) differences in system parameters, *e.g.*, $u_t = \tilde{\mathcal{N}}[u, f_1, \ldots, f_p]$ with $g_i = 0$ for all $i = 1, \ldots, r$; (2) differences in external forcing functions and/or boundary conditions, *e.g.*, $u_t = \tilde{\mathcal{N}}[u, f_1, \ldots, f_p]$ with $g_i = f_i + \epsilon$ for $i = 1, \ldots, r$ with $r \leq p$; and (3) missing terms in the partial differential equation describing the dynamics of the system, *e.g.*, $u_t = \tilde{\mathcal{N}}[u, f_1, \ldots, f_p]$ with $g_1 \neq 0$. We briefly describe the candidate systems below.

System 1: 1D heat equation

In our first system, we assume both the actual model M_{act} and current model M_{curr} system dynamics are given by

$$u_t = D_* u_{xx}, \tag{7}$$

where $x \in \mathbb{R}$, $u \in \mathbb{R}$ is the temperature, and D_* is the diffusion coefficient and is set to either D_{act} or D_{curr}. In this scenario, the discrepancy in the models arises due to a mismatch in the actual and assumed diffusion coefficients.

System 2: Lid cavity problem

For our second system, we consider a modified version of the lid cavity problem presented in [44]. The actual model, M_{act}, is given by

$$u_t = -(u \cdot \nabla)u - \nabla p + \frac{1}{Re}\nabla^2 u + F. \tag{8}$$

In [44], F is chosen to be an external body force with a whirlpool effect. In this work, we employ the same F as in [44] but include a periodic element to F whose components are given by

$$\begin{aligned}
F_x &= (12 - 24y)x^4 + (-24 + 48y)x^3 + \\
&\quad (-48y + 72y^2 - 48y^3 + 12)x^2 + \\
&\quad (-2 + 24y - 72y^2 + 48y^3)x + \\
&\quad (1 - 4y + 12y^2 - 8y^3)120\sin(e^{1.3t} + 80t), \\
F_y &= (8 - 48y + 48y^2)x^3 + \\
&\quad (-12 + 72y - 72y^2)x^2 + \\
&\quad (4 - 24y + 48y^2 - 48y^3 + 24y^4)x + \\
&\quad (-12y^2 + 24y^3 - 12y^4)120\cos(e^{1.3t} + 80t).
\end{aligned}$$

In this system, the assumed model, M_{curr}, is given by the Navier–Stokes equation for incompressible flows,

$$u_t = -(u \cdot \nabla)u - \nabla p + \frac{1}{Re}\nabla^2 u \tag{9}$$

with $\nabla u = 0$, where $x \in \mathbb{W} \subset \mathbb{R}^2$ denotes the position, $u \in \mathbb{R}^2$ is the flow velocity, Re is the Reynolds number, and p is the pressure. In contrast to the classical lid cavity problem, where the domain $\mathbb{W}$ is a square in which the top boundary moves with a constant speed, we assume the dynamics are subject to periodic boundary conditions at the top and bottom boundaries of the square given by

$$\begin{aligned}
u_{top} &= [2\sin((e^{1.2t} + 60)t)], \\
u_{bottom} &= [2\sin((e^{1.2t} + 50)t)].
\end{aligned}$$

System 3: Flow around a cylinder

For our third system, we consider the 2D flow around a cylinder modeled using the Navier–Stokes equations. The cylinder has a $1m$ radius and is centered at $(20, 20)$ in a 50m $\times$ 40 m rectangular workspace. For the actual system, M_{act}, the cylinder moves vertically along the $y = 20$ axis such that its center moves periodically between $(20, 21)$ and $(20, 19)$ at a frequency of 0.3927 rad/sec. The velocity profile at the left boundary is set to be a uniform stream while a zero pressure outflow condition is imposed at the right boundary. The Reynolds number is set to 200. In this scenario, the system model or dynamics, M_{curr}, is assumed to be that of the stationary cylinder placed in the same uniform free stream flow, at the same location, with the same radius, operating at the same Reynolds number. We note that the oscillation frequency for the moving cylinder in M_{act} is set to be approximately the vortex shedding frequency of M_{curr}.

Table 1
Neural network parameters for System 1. Note that TDDL stands for Time Distributed Dense Layer, LSTM stands for Long Short-Term Memory, and ReLU stands for Rectified Linear Unit.

Layer Kind	Activation Function	Number of Nodes
Input 0: U_{curr}, Coordinates	N/A	N/A
Layer 1: TDDL[Input 0]	ReLU	32
Layer 2: LSTM Layer[Layer 1]	Tanh/Sigmoid	64
Layer 3: LSTM Layer[Layer 2]	Tanh/Sigmoid	32
Layer 4: LSTM Layer[Layer 3]	Tanh/Sigmoid	32
Layer 5: Dense Layer[Layer 4]	ReLU	10
Layer 6: Dense Layer [Layer 5]	Linear	1

Table 2
Neural network parameters for System 2. Note that TDDL stands for Time Distributed Dense Layer, LSTM stands for Long Short-Term Memory, and ReLU stands for Rectified Linear Unit.

Layer Kind	Activation Function	Number of Nodes
Input 0: U_{curr}, F, Coordinates	N/A	N/A
Layer 1: TDDL[Input 0]	ReLU	32
Layer 2:TDDL[Layer 1]	ReLU	64
Layer 3: LSTM Layer[Layer 2]	Tanh/Sigmoid	64
Layer 4: LSTM Layer[Layer 3]	Tanh/Sigmoid	32
Layer 5: LSTM Layer[Layer 4]	Tanh/Sigmoid	32
Layer 6: Dense Layer [Layer 5]	ReLU	10
Layer 7: Dense Layer [Layer 6]	Linear	2

Table 3
Neural network parameters for System 3. Note that TDDL stands for Time Distributed Dense Layer, LSTM stands for Long Short-Term Memory, and ReLU stands for Rectified Linear Unit.

Layer Kind	Activation Function	Number of Nodes
Input 0: U_{curr}, Coordinates, Cylinder Position	N/A	N/A
Layer 1: TDDL[Input 0]	ReLU	32
Layer 2: TDDL[Layer 1]	ReLU	64
Layer 3: LSTM Layer[Layer 2]	Tanh/Sigmoid	64
Layer 4: LSTM Layer[Layer 3]	Tanh/Sigmoid	32
Layer 5: LSTM Layer[Layer 4]	Tanh/Sigmoid	32
Layer 6: Dense Layer [Layer 5]	ReLU	10
Layer 7: Dense Layer [Layer 6]	Linear	2

4.2. Implementation

The details of each system's architecture are summarized in Tables 1, 2, and 3. We use Adam [36], a powerful and computationally efficient optimization algorithm with the recommended default parameters to initialize the algorithm. We set the algorithm batch size to 64, and used the Python package Keras [45] to train the network for a total of 50 epochs. Note that for our dense layers, we chose ReLU as our activation function. The function demonstrated the best performance on our tasks.

Given the lightweight nature of our networks and the small size of input data, we trained the networks on a CPU Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz. Tensorflow, the backend of Keras, automatically distributes training on multiple cores. The average time for completing one epoch for Systems 1, 2, and 3 is 1, 60, and 40 s respectively. The differences in training time between each system are mostly due to the training set size. The marginal difference between each system architecture does not significantly change the training time.

It is important to note that expanding the neural net input size will impact the computational time. Adding more points

to the hypercube will result in d more connections where d is the number of nodes in the Stage 1 first layer. These d new connections represent the new input contribution to each node in the first layer. We can also apply the network on longer data sequences. This would not result in any new connections, but it will result in applying Stages 1 and 2 of the network on the added time steps. Both of these changes, when studied independently, will result in a constant increase in the number of operations for both prediction and training.

There is also an impact on computational time through the addition of more data. In training neural networks, we apply the same vectorized operations, mostly matrix multiplications, on batches of data. The nature of this computational process means that for each new data point, the number of operations for both training and prediction increases by a constant factor.

Finally we note that in solving new problems, we might need to expand the network representational capacity by adding more nodes and layers. The change in the computational cost of the network will heavily depend on the size and complexity of the new network. However, recent advances in GPU development tailored specifically for Deep Learning offer a range of solutions for building optimized and scalable implementations of complicated and heavy architectures.

4.3. Datasets

In this work, we employ numerical solutions to the actual and assumed models, M_{act} and M_{curr}, to generate the ground truth, test, and training datasets. The ground truth and actual system observations, U_{act}, are obtained by numerically solving M_{act}. Similarly, the values for U_{curr} are obtained by numerically solving M_{curr} for the assumed parameter values, e.g., $\tilde{\mathcal{N}}$. For System 1, the 1D heat equation given by Eq. (7) was solved using the finite volume based PDE solver in Python (FiPy). The equation was discretized on a spatial 50×1 grid over 0.006 s with time step of 0.000012. The number of time frames is 500. For System 2, a finite difference scheme was used to solve the Navier Stokes equations given by (8)–(9). The equation was discretized on a spatial 30×30 grid over 2 s with time step of 0.001. The number of time frames is 2000. Lastly, numerical solutions for System 3, flow around a cylinder, were obtained using OpenFoam [46]. The datasets for U_{act} and U_{curr} were obtained on a largely uniform grid consisting of approximately 100×70 points over 2000 s using OpenFoam's `pimpleFoam` solver with a solution time step of 0.001 s. However, U_{act} and U_{curr} consist of data obtained at every second over the total 2000 s simulation run and thus the data consist of 2000 time frames.

Training and test datasets are comprised of both U_{curr} and U_{act}. Assuming U_{act} and U_{curr} consists of N frames ($N_x \times N_y$ grid points in each frame), we partition the data into four sets: training, validation, local test, and future test. For the training, validation, and local test sets, we consider K consecutive frames. In order to split the data in the K frames between the three sets, we split the set of $N_x \times N_y$ grid points at each frame randomly between training, validation and local test sets. We choose to include 60% of the grid points in the training set, 10% of the grid points in the validation set, and 30% of the grid points in the local test set. Note that the split is the same for each frame. For the future test set, we consider the remaining $T = N - K$ frames with all the grid points. We use the training set to train the network, the validation set to test the effect of hyperparameter optimization and different network architectures on the network performance, the local test set to measure the chosen network ability to generalize over unseen grid points, and the future test set to measure the network ability to generalize over unseen dynamics, *i.e.*, predict future observations.

For System 1, we include the first 150 frames in the training, validation, and local test sets ($K = 150$) and the last 350 frames ($T = 350$) in the future test set. This setup results in a total of 4,144 points for training, 740 points for validation, 2,220 for local test, and 16,800 for future test. For System 2, we use the first 1,000 frames for training ($K = 1,000$), validation and local test and use the last 1,000 ($T = 1,000$) frames for the future test. This results in a total of 469,060 points for training, 77,844 points for validation, 235,528 points for local test, and 784,000 points for future test. For System 3, we use 100 frames of data between the frames 500 and 600, *i.e.*, between 500 and 600 s, for the training, validation and local test set. We used the 1,400 frames after 600 s for the future test. This results in a total of 399,800 points for training, 66,600 for validation, 200,000 points for local test, and 9,786,000 points for future test.

4.4. Evaluation metrics

To assess model performance, we use two benchmarks to measure the difference between two sets of F frames: Set_1 (tested set) and Set_2 (ground truth). In our evaluation, Set_1 will either be U_{curr} or U_{nn} and Set_2 will be U_{act}.

Mean Squared Errors (MSE). The first benchmark uses the mean squared difference between Set_1 and Set_2. For 2D output, we take the average of the two outputs for every point before computing the MSE. We also include the mean magnitude square difference (MMSD) and the mean cosine similarity (MCS) between the two sets in the benchmark for the Lid Cavity and the Flow Around a Cylinder problems in Systems 2 and 3 since both are 2D systems.

Proper orthogonal decomposition. The second benchmark compares the proper orthogonal decomposition (POD) modes that accounts for 99% of the system variation. Complex nonlinear dynamical systems can exhibit significant spatiotemporal variations, often at differing scales. To extract the dominant dynamics of these systems, techniques for modal analysis are often used to construct a reduced order representation of the dynamics. POD is a data-driven reduced order modeling strategy that is often used to identify the dominant dynamics of a system purely from observations [34,35].

Given N snapshots of the system states which can be obtained either through measurements and/or numerical simulations, let $\boldsymbol{x}(\boldsymbol{t}) = [x_1(t), \ldots, x_k(t)]^\top$ denote the set of spatial coordinates in $\mathbb{W}$ at $t = 1, \ldots, N$. We note that the points in $\boldsymbol{x}(\boldsymbol{t})$ correspond to the grid points in which u_{act} and u_{curr} values are provided at some given time t. Using $\boldsymbol{x}(\boldsymbol{t})$, we can construct a covariance matrix as

$$\boldsymbol{K} = \frac{1}{m}\sum_{t=1}^{m} \boldsymbol{x}(\boldsymbol{t})\boldsymbol{x}(\boldsymbol{t})^\top = \frac{1}{m}\boldsymbol{X}\boldsymbol{X}^\top, \tag{10}$$

where $\boldsymbol{X} \in \mathbb{R}^{n\times m}$ with its columns as $\boldsymbol{x}(\boldsymbol{t})$. To extract the dominant dynamic modes from the data given by $\boldsymbol{x}(\boldsymbol{t}) = [x_1(t), \ldots, x_k(t)]^\top$ for $t = 1, \ldots, N$, we obtain the low dimensional basis for the data by solving the symmetric eigenvalue problem

$$\boldsymbol{K}\boldsymbol{\phi_i} = \lambda_i \boldsymbol{\phi_i},$$

where $\boldsymbol{K}$ has N eigenvalues such that $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_N \geq 0$ and the eigenvectors $\boldsymbol{\phi}$ are pairwise orthonormal.

The original basis is then truncated into a new basis $\boldsymbol{\Phi}$ by choosing k eigenvectors that capture the desired fraction, E, of the total variance of the system, such that their eigenvalues satisfy

$$\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{n}\lambda_i} \geq E.$$

Each term $\boldsymbol{x}(\boldsymbol{t})$ can be written as

$$\boldsymbol{x}(\boldsymbol{t}) = \boldsymbol{\Phi}\boldsymbol{c}(\boldsymbol{t}), \tag{11}$$

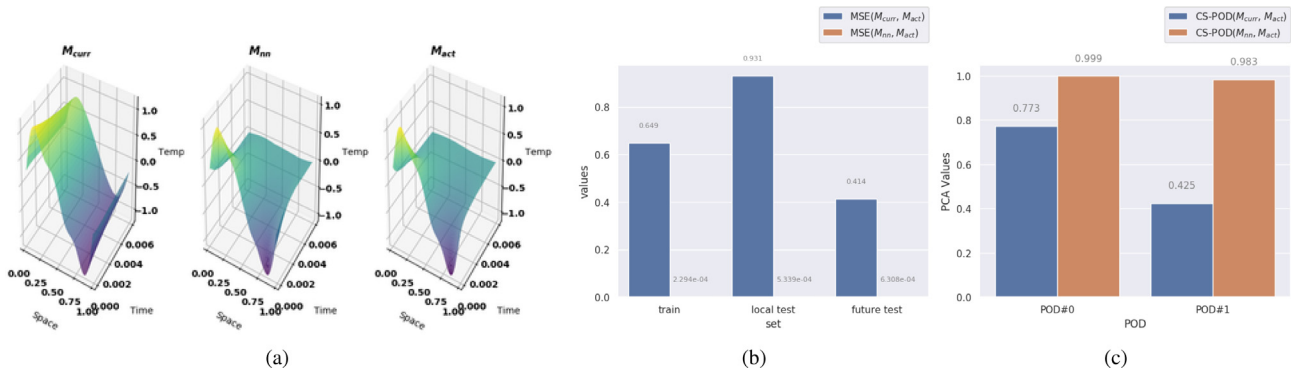


Fig. 4. (a) System 1: Temperature as a function of the spatial and temporal coordinates for (left) U_{curr}, (middle) U_{nn}, and (right) U_{act}. (b) MSE between U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 1. (c) Cosine similarity between the first principal POD mode of U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

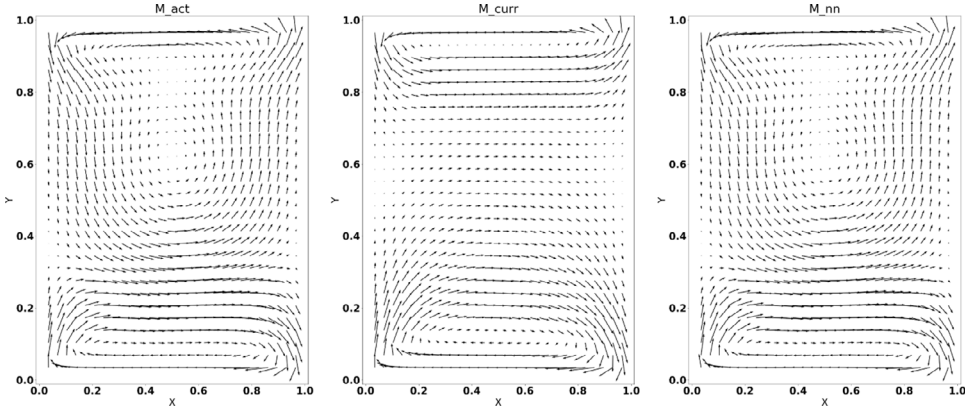


Fig. 5. System 2: Vector field at $t = 1.147$ s for (left) U_{curr}, (middle) U_{nn}, and (right) U_{act}.

where $\boldsymbol{c(t)} = [c_1(t), \ldots, c_k(t)]^\top$ holds time-dependent coefficients and $\boldsymbol{\Phi} \in \mathbb{R}^{n\times k}$ with its columns as $\boldsymbol{\phi_1}$, ..., $\boldsymbol{\phi_k}$. The low-dimensional, orthogonal subspace associated with $\boldsymbol{\Phi}$ is an optimal approximation of the data with respect to minimizing least squares error.

To compare the POD modes, we compute the inner product, *i.e.* the cosine similarity, between the two sets of principal components obtained for Set_1 and Set_2. We call this metric CS-POD, for short. We calculate the statistics of both benchmarks on two cases: Case 1, Set_1 is M_{curr} and Case 2, Set_1 is M_{nn}. Case 1 provides a relative baseline for measuring the performance of the neural net in Case 2. We report the first benchmark statistics over training, local test, and future test sets, and the second benchmark statistics over the entire simulation.

5. Results and discussion

We present and discuss the results of our proposed learning framework for each of the candidate systems.

System 1: 1D heat equation

Fig. 4(a) shows the temperature data generated by M_{act}, M_{curr}, and M_{nn} for the entire spatiotemporal domain. In these simulations, D_{act} and D_{curr} were set to 15 and 1 mm^2/s respectively. Qualitatively we see that the network model does an excellent job in resolving the inaccurately modeled dynamics and accurately captures the true dynamics of the system. Fig. 4(b) quantitatively shows the network's ability to generalize over local unseen grid points as well as data in the future set. In fact, one can see that the error between U_{act} and U_{nn} is orders of magnitudes less than that of U_{act} and U_{curr}. Moreover, the error bars between U_{act} and U_{nn} are so small that the orange bars are not visible in the graph (the exact values for comparison are denoted in the figure). The quantitative results are further confirmed in Fig. 4(c) which shows the CS-POD for the POD modes. In this problem, the POD decomposition of U_{nn} over the entire simulation yielded two principal modes as did the POD decomposition of U_{act}. The agreement between these POD modes is excellent as demonstrated by a CS-POD values that are very close to unity, as seen in Fig. 4(c).

System 2: Lid cavity problem

Fig. 5 shows a snapshot of the vector field generated by M_{act}, M_{curr}, and M_{nn} for the entire domain at $t = 1.147$ s. As with System 1, we see qualitatively that the network model does an excellent job in resolving the inaccurately modeled dynamics and does accurately capture the actual dynamics of the system. Fig. 6(a)–6(c) quantitatively show the network's ability to combine U_{curr} and observations of $g_1 = F$ to correctly predict U_{act} over local unseen grid points as well as data in the future set. Figs. 6(a) and 6(b) show respectively that the MSE and MMSD between U_{act} and U_{nn} are orders of magnitudes less than that of U_{act} and U_{curr}. The quantitative results are further confirmed in Figs. 6(c) which shows that the mean cosine similarity between U_{act} and U_{nn} is close to unity thus demonstrating that U_{nn} is resolving the actual dynamics to a far greater degree than is U_{curr}. Similarly, Fig. 6(d) shows the CS-POD for the first six principal POD modes over the entire simulation. The CS-POD values again demonstrate the network's ability to resolve the actual system's dynamics. In short, the high degree of accuracy shows that our network is capable of correctly predicting observations both in previously unseen regions in the workspace as well as in future time steps.

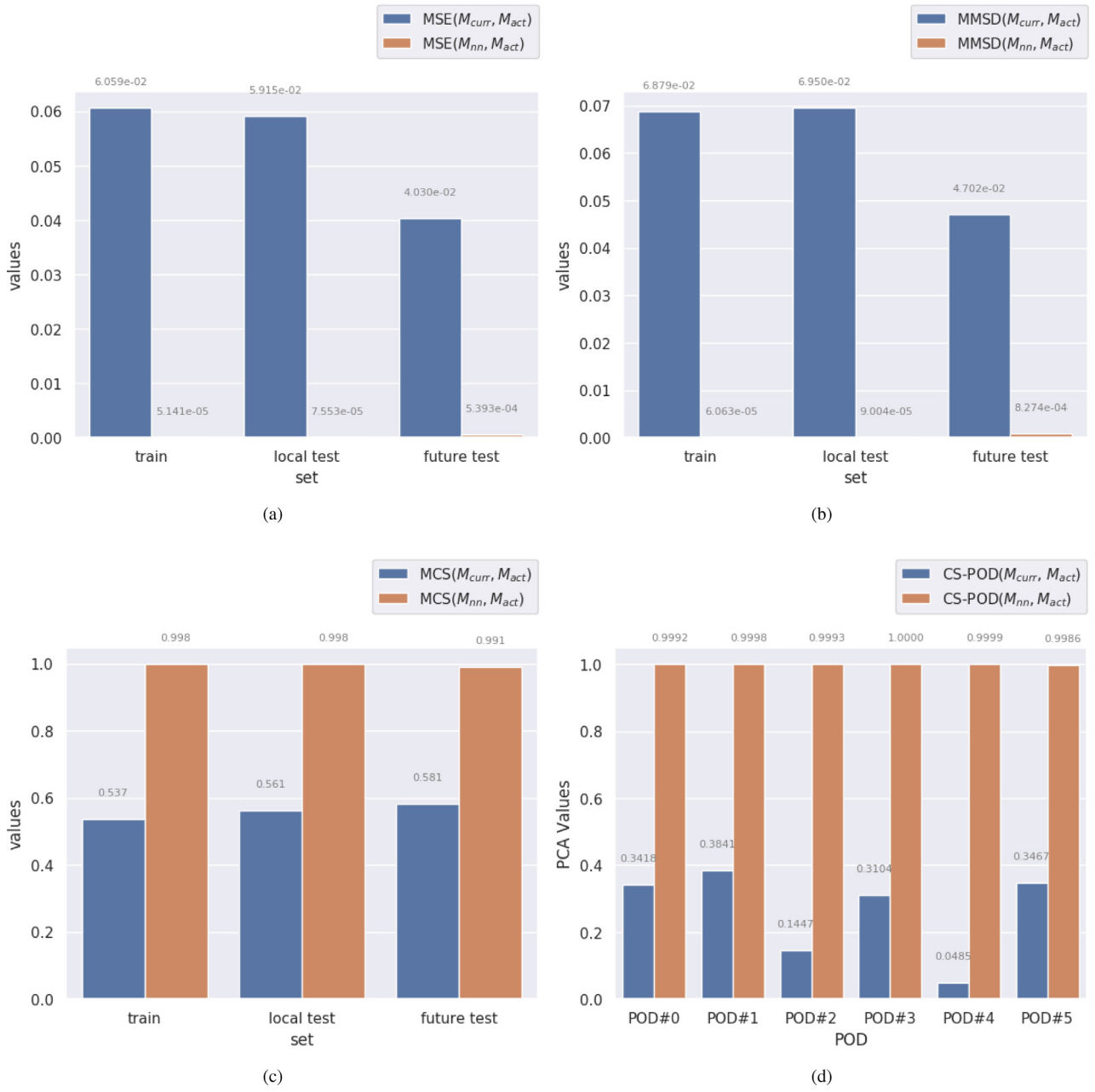


Fig. 6. (a) MSE between U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 2. (b) MMSD between U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 2. (c) Mean cosine similarity between U_{act} and U_{curr} and U_{act} and U_{nn} for system 2. (d) Cosine similarity between the first five POD modes of U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 2. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

It is important to note that the periods of the body force and the moving upper and lower boundaries in System 2 are not constant. In fact, they change exponentially as a function of time. Ideally, the trained network should capture this exponential change in the periods and be able to accurately predict future values outside of the training frames. In reality though, the prediction accuracy would degrade the farther out the prediction times are from the training times. To quantify this behavior, three training regimes were considered with different training set lengths. The training sets for the three regimes contained the first 500 frames, first 750 frames, and the first 1000 frames of the dataset respectively. We evaluated the system's predictive power using intervals of 250 future output frames and the results are shown in Fig. 7. The metric (MSE, MCS and MMSD) for each interval is computed across all 250 frames in that interval. As expected, the prediction accuracy degrades the further out the prediction time is from the training set. For this particular case, the network is able to predict approximately one training period into the future, with a fair degree of accuracy.

System 3: Flow around the cylinder

Fig. 8 shows a snapshot of the magnitude of the velocity field at $t = 1390$ s generated by M_{act}, M_{curr}, and M_{nn}. In these results, the Reynolds numbers for both M_{act} and M_{curr} were set to 200. As with the previous two systems, we see qualitatively that the network model does an excellent job in resolving the inaccurately modeled dynamics and does accurately capture the actual dynamics of the system. In particular, note that the network model M_{nn} is accurately capturing the vortex shedding frequency while the M_{curr} vortices are out of phase with the actual vortex shedding

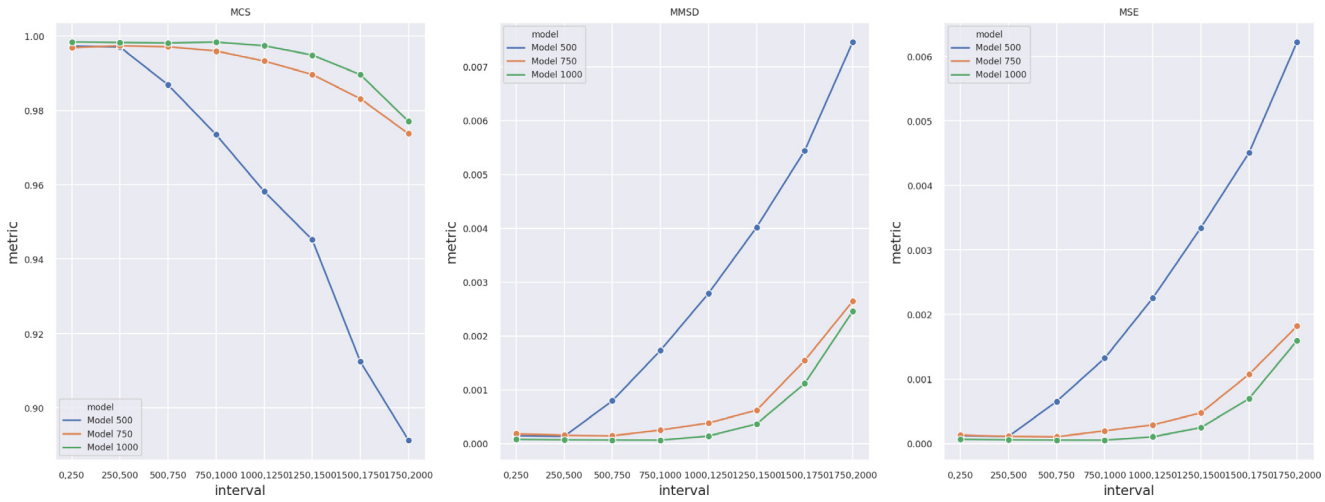


Fig. 7. Comparison between the predictive power of the M_{nn} trained using the first 500, 750, and 1000 frames. The x axis denotes the time interval in increments of 250. The metric for each interval represents the metric value computed for the frames in that interval.

pattern. As in the previous systems, Figs. 9(a)–9(c) quantitatively show the network's ability to combine values of U_{curr} as well as $g_1 = O(t)$ (where O indicates the position of the cylinder at time t) to correctly predict U_{act} over local unseen grid points as well as data in the future test set. Figs. 9(a) and 9(b) show respectively that the MSE and MMSD between U_{act} and U_{nn} is orders of magnitudes less than that of U_{act} and U_{curr}. The quantitative results are further confirmed in Figs. 9(c) which shows that the mean cosine similarity between U_{act} and U_{nn} are close to unity thus demonstrating that U_{nn} is resolving the actual dynamics to a far greater degree than is U_{curr}. Similarly, Fig. 9(d) shows the CS-POD for the first six principal POD modes over the entire simulation. The CS-POD values again demonstrate the network's ability to resolve the actual system's dynamics. In short, the high degree of accuracy shows that our network is capable of correctly predicting observations both in previously unseen regions in the workspace as well as in future time steps for systems exhibiting more complex dynamics.

To evaluate the predictive performance of M_{nn}, we focus on the network's ability to identify the periodicity of the oscillations. Since System 3 is periodic, once the network learns the true periodicity of the dynamics, it has effectively learned the true dynamics of the system for all future times. To quantify the difference in periodicity between the model output and the ground truth, the following analysis was performed. For each point in the local test set, τ_i, we consider its time series from frame 600, the last frame in the training set, to frame 2000 in both U_{nn} and U_{act}. We denote these as $U_{nn}(\tau_i, 600\text{–}2000)$ and $U_{act}(\tau_i, 600\text{–}2000)$ respectively. We start by computing the frequency spectrums of $U_{nn}(\tau_i, 600\text{–}2000)$ and $U_{act}(\tau_i, 600\text{–}200)$ using the Fast Fourier Transform (FFT) which we denote as FFT_{nn} and FFT_{act}. Consider the percentage mean absolute difference between the frequencies that corresponds to the energy peaks between FFT_{nn} and FFT_{act} which we denote by $\%\Delta(FFT_{U_{nn}}, FFT_{U_{act}})$. The mean of $\%\Delta(FFT_{U_{nn}}, FFT_{U_{act}})$ is then computed for every grid point in the local test set which resulted in a value of 0.0239. This analysis indicates that the neural network output not only accurately captures the periodicity of the underlying phenomena but it is able to correctly identify the global features of the dynamics. In short, once the network captures the periodicity, it can then predict the system's behavior at any time in the future.

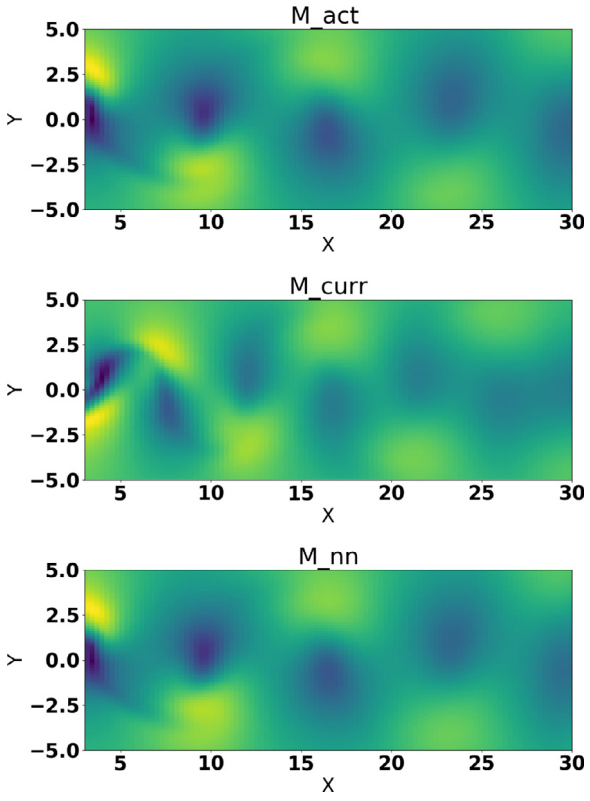


Fig. 8. System 3: Magnitude of the velocity profile given by (left) U_{curr}, (middle) U_{nn}, and (right) U_{act} at time $t = 1390$ s.

6. Conclusion and future work

We have proposed a data-driven modeling strategy based on a neural network machine learning framework that enables one to overcome improperly or inadequately modeled dynamics for systems that exhibit complex spatiotemporal behavior. Given a system model that does not accurately capture the true dynamics, our machine learning strategy uses data generated from the improper system model combined with observational data from the actual system to create a neural network model. As we have shown with three complex dynamical systems, the network model that is created is capable of accurately resolving the incomplete or inaccurate dynamics to generate solutions that compare very favorably with the actual dynamics, both in previously unobserved regions as well as for future states.

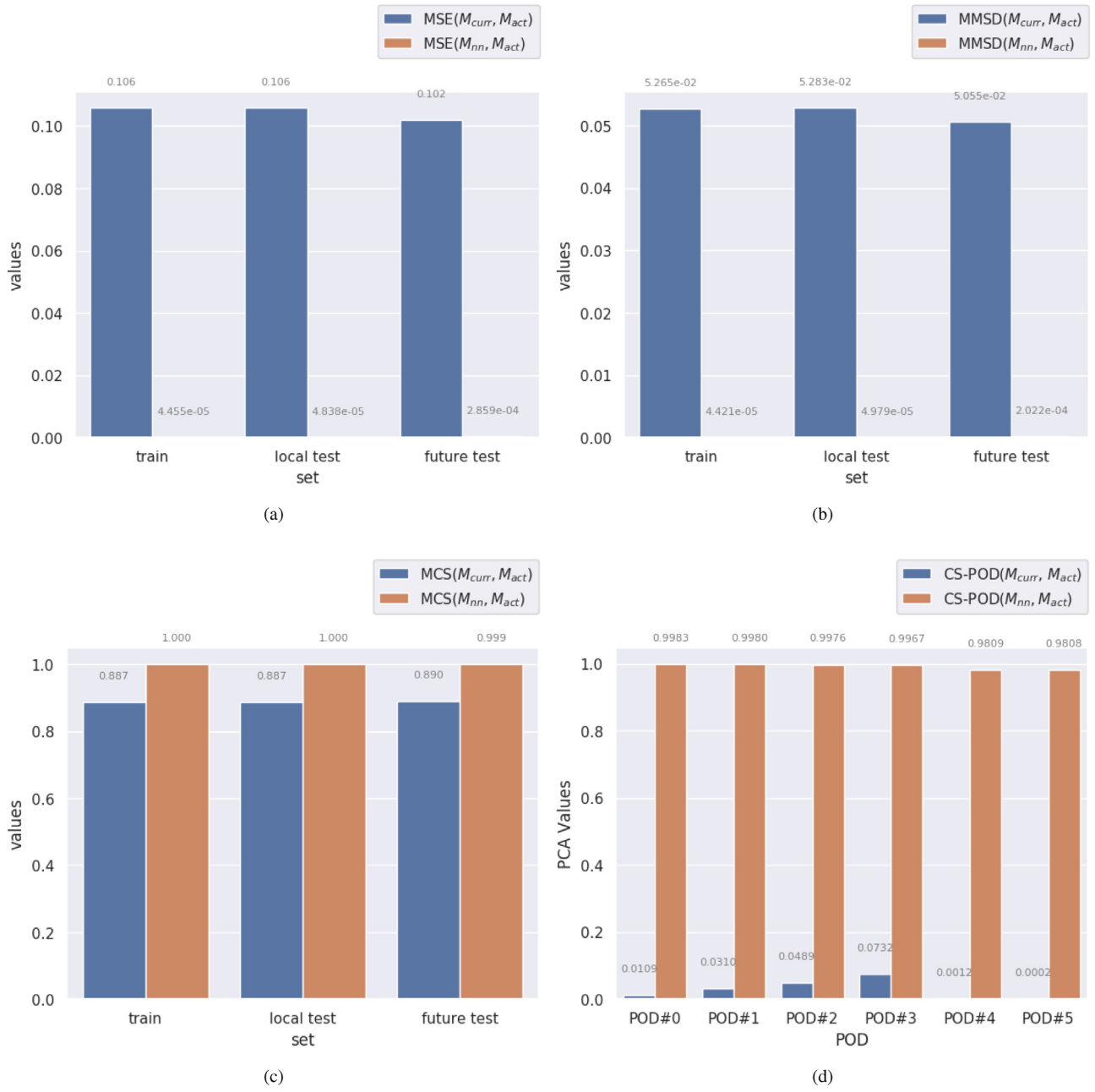


Fig. 9. (a) MSE between U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 3. (b) MMSD between U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 3. (c) Mean cosine similarity between U_{act} and U_{curr} and U_{act} and U_{nn} for system 3. (d) Cosine similarity between the first six POD modes of U_{act} and U_{curr} (light blue) and U_{act} and U_{nn} (dark blue) for system 3. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Our approach leverages state-of-the-art machine learning frameworks and existing, but limited, knowledge of the physical constraints that drives a process. The result is an equation-free representation of the system dynamics that encodes a baseline understanding of the underlying physics that drives the process. Since our output is a neural network representation of the system model, the output of our network consists of a set of pointwise inferences and thus is equation-free. Nevertheless, the output can be fed into existing data-driven model discovery techniques to obtain closed-form equation representations of the dynamical system [17,18].

In future, we plan to perform a detailed analysis on our learning framework performance for different error bounds to better understand acceptable deviations from the true model. Associated with this is the effect of noise, and to this end we plan to investigate how measurement uncertainty in $\hat{U}_{act}$ impacts the performance of M_{nn}. Since real-world systems are inherently noisy, we must be able to incorporate noisy observational data while still accurately capturing the system's dynamics. As such, it is important to be able to deal with situations where every observation is subject to a noise that is non-negligible or with situations where one has very noisy outlier observations. While the impact of noise on a network's performance is well documented and studied in the computer vision literature [47], its impacts on networks modeling more complex phenomena are less well understood. A complete analysis of the effect of noise includes consideration of both additive and multiplicative noise, and involves analyzing simulated systems where deterministic and stochastic elements can be tightly controlled to establish ground truth for comparisons.

By developing methods that can deal with negligible and non-negligible noise, we will enable the study of complex and high-dimensional systems including those found in fluid dynamics and in particular geophysical fluid dynamics. Fluid flows are complex and exhibit multi-scale phenomena whose dynamics are not at all well-understood. Even the underlying physical mechanisms for flows are not fully understood. In the future, we plan to use the framework developed in this article to make predictions and estimations. For example, in a geophysical flow, information such as wind forcing or data from depth, may not be included in the models. Even with noisy and sparse observations, we would like to investigate if our framework can be used to accurately resolve the inadequately modeled dynamics.

CRediT authorship contribution statement

Maan Qraitem: Conceptualization, Methodology, Software, Data curation, Writing - original draft, Visualization, Investigation, Software, Validation, Writing - review & editing. **Dhanushka Kularatne:** Conceptualization, Methodology, Software, Data curation, Writing - original draft, Writing - review & editing. **Eric Forgoston:** Conceptualization, Methodology, Software, Data curation, Writing - original draft, Supervision, Writing - review & editing. **M. Ani Hsieh:** Conceptualization, Methodology, Software, Data curation, Writing - original draft, Supervision, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Babak Alipanahi, Andrew Delong, Matthew T. Weirauch, Brendan J. Frey, Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning, Nature Biotechnol. 33 (2015) 831, http://dx.doi.org/10.1038/nbt.3300.

[2] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[3] Brenden Lake, Wojciech Zaremba, R Fergus, Todd Gureckis, Deep neural networks predict category typicality ratings for images, in: R Dale, C Jennings, P Maglio, T Matlock, D Noelle, A Warlaumont, J Yoshimi (Eds.), Proceedings of the 37th Annual Conference of the Cognitive Science Society, Cognitive Science Society, 2015.

[4] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[5] Zoubin Ghahramani, Probabilistic machine learning and artificial intelligence, Nature 521 (7553) (2015) 452.

[6] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, Shailendra Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Comput. Mech. (ISSN: 0178-7675) 64 (2) (2019) 525–545, http://dx.doi.org/10.1007/s00466-019-01740-0, URL https://doi.org/10.1007/s00466-019-01740-0.

[7] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, Comput. Graph. Forum 38 (2) (2019) 71–82, http://dx.doi.org/10.1111/cgf.13620, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13620.

[8] Cristina White, Daniela Ushizima, Charbel Farhat, Fast neural network predictions from constrained aerodynamics datasets, 2019.

[9] Arvind T. Mohan, Datta V. Gaitonde, A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks, 2018.

[10] Sangseung Lee, Donghyun You, Prediction of laminar vortex shedding over a cylinder using deep learning, 2017.

[11] Robert Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc. Ser. B Stat. Methodol. 58 (1) (1996) 267–288, http://dx.doi.org/10.1111/j.2517-6161.1996.tb02080.x, URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x.

[12] Chen Yao, Erik M. Bollt, Modeling and nonlinear parameter estimation with Kronecker product representation for coupled oscillators and spatiotemporal systems, Physica D (ISSN: 0167-2789) 227 (1) (2007) 78–99, http://dx.doi.org/10.1016/j.physd.2006.12.006, URL http://www.sciencedirect.com/science/article/pii/S0167278906004799.

[13] Josh Bongard, Hod Lipson, Automated reverse engineering of nonlinear dynamical systems, Proc. Natl. Acad. Sci. (ISSN: 0027-8424) 104 (24) (2007) 9943–9948, http://dx.doi.org/10.1073/pnas.0609476104, URL https://www.pnas.org/content/104/24/9943.

[14] Michael Schmidt, Hod Lipson, Distilling free-form natural laws from experimental data, Science (ISSN: 0036-8075) 324 (5923) (2009) 81–85, http://dx.doi.org/10.1126/science.1165893, URL https://science.sciencemag.org/content/324/5923/81.

[15] Pileun Kim, Jonathan Rogers, Jie Sun, Erik Bollt, Causation entropy identifies sparsity structure for parameter estimation of dynamic systems, J. Comput. Nonlinear Dyn. (ISSN: 1555-1415) 12 (1) (2016) http://dx.doi.org/10.1115/1.4034126, 011008.

[16] Wei Pan, Ye Yuan, Jorge M. Gonçalves, Guy-Bart Stan, A sparse Bayesian approach to the identification of nonlinear state-space systems, IEEE Trans. Automat. Control 61 (2016) 182–187.

[17] Steven L. Brunton, Joshua L. Proctor, J. Nathan Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. (ISSN: 0027-8424) 113 (15) (2016) 3932–3937, http://dx.doi.org/10.1073/pnas.1517384113.

[18] Bethany Lusch, J Nathan Kutz, Steven L Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, 2017, arXiv preprint arXiv:1712.09707.

[19] Romit Maulik, Omer San, Adil Rasheed, Prakash Vedula, Data-driven deconvolution for large eddy simulations of Kraichnan turbulence, Phys. Fluids 30 (12) (2018) 125109.

[20] Romit Maulik, Omer San, Adil Rasheed, Prakash Vedula, Subgrid modelling for two-dimensional turbulence using neural networks, J. Fluid Mech. 858 (2019) 122–144.

[21] Ibrahim Ayed, Emmanuel de Bézenac, Arthur Pajot, Julien Brajard, Patrick Gallinari, Learning dynamical systems from partial observations, 2019.

[22] Abd AlRahman R. AlMomani, Jie Sun, Erik Bollt, How entropic regression beats the outliers problem in nonlinear system identification, Chaos 30 (1) (2020).

[23] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations, 2017.

[24] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, 2017, arXiv preprint arXiv:1711.10566.

[25] Maziar Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, 2018, arXiv preprint arXiv:1801.06637.

[26] Maziar Raissi, Alireza Yazdani, George Em Karniadakis, Hidden fluid mechanics: A Navier-Stokes informed deep learning framework for assimilating flow visualization data, 2018.

[27] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, 2018.

[28] Jaideep Pathak, Alexander Wikner, Rebeckah Fussell, Sarthak Chandra, Brian R Hunt, Michelle Girvan, Edward Ott, Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model, Chaos 28 (4) (2018) 041101.

[29] J. Ling, J. Templeton, Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty, Phys. Fluids 27 (8) (2015) 085103, http://dx.doi.org/10.1063/1.4927765, URL https://aip.scitation.org/doi/abs/10.1063/1.4927765.

[30] Tharindu P. Miyanawala, Rajeev K. Jaiman, An efficient deep learning technique for the Navier-Stokes equations: Application to unsteady wake flow dynamics, 2017.

[31] Jonathan Viquerat, Elie Hachem, A supervised neural network for drag prediction of arbitrary 2D shapes in low Reynolds number flows, Comput. Fluids (2019) URL https://hal.archives-ouvertes.fr/hal-02401463.

[32] Corentin J. Lapeyre, Antony Misdariis, Nicolas Cazard, Denis Veynante, Thierry Poinsot, Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates, Combust. Flame (ISSN: 0010-2180) 203 (2019) 255–264, http://dx.doi.org/10.1016/j.combustflame.2019.02.019, URL http://www.sciencedirect.com/science/article/pii/S0010218019300835.

[33] Jordan Read Anuj Karpatne, Vipin Kumar, Physics-guided neural networks (PGNN):an application in lake temperature modeling, 2018.

[34] Michael Kirby, Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns, John Wiley & Sons, Inc., New York, NY, USA, ISBN: 0471239291, 2000.

[35] Lawrence Sirovich, Turbulence and the dynamics of coherent structures. I. Coherent structures, Quart. Appl. Math. (ISSN: 0033-569X) 45 (3) (1987) 561–571, http://dx.doi.org/10.1090/qam/910463.

[36] D. Kingma, J. Lei Ba, Adam: A Method for Stochastic Optimization, 2015.

[37] Yoram Singer John Duchi, Adaptive subgradient methods for online learning and stochastic optimization, Mach. Learn. Res. 12 (2011).
[38] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, Stephen Marshall, Activation functions: Comparison of trends in practice and research for deep learning, 2018.
[39] Yoshua Bengio, Patrice Simard, Paolo Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. Learn. Syst. 5 (1994).
[40] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[41] C. Olah, Understanding LSTM networks, 2015, URL http://colah.github.io/posts/2015-08-Understanding-LSTM.
[42] Alex Sherstinsky, Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network, Physica D (ISSN: 0167-2789) 404 (2020) 132306, http://dx.doi.org/10.1016/j.physd.2019.132306, URL http://www.sciencedirect.com/science/article/pii/S0167278919305974.
[43] Bo Chang, Minmin Chen, Eldad Haber, Ed H. Chi, AntisymmetricRNN: A dynamical system view on recurrent neural networks, in: International Conference on Learning Representations, 2019, URL https://openreview.net/forum?id=ryxepo0cFX.
[44] Andrea La Spina, 2D unsteady Navier-Stokes, 2019, URL https://www.mathworks.com/matlabcentral/fileexchange/60869-2d-unsteady-navier-stokes?focused=8d8fc49b-893e-4760-ac16-b2f7f6660d8b&tab=example, (Online; Accessed 30-December-2019).
[45] François Chollet, Keras, in: GitHub Repository, GitHub, 2015, https://github.com/keras-team/keras.
[46] OpenFOAM Wiki, Vortex shedding by Joel Guerrero 2D — OpenFOAM Wiki, 2019, URL https://wiki.openfoam.com/index.php?title=Vortex_shedding_by_Joel_Guerrero_2D&oldid=2706, (Online; Accessed 29-December-2019).
[47] T. Plotz, S. Roth, Benchmarking denoising algorithms with real photographs, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017, pp. 2750–2759.