International Conference on Adaptive Modeling and Simulation
ADMOS 2023
F.Larsson and P.Díez

ERROR ASSESSMENT FOR A FINITE ELEMENTS - NEURAL NETWORKS APPROACH APPLIED TO PARAMETRIC PDES. THE TRANSPORT EQUATION

ALEXANDRE CABOUSSAT[1], MAUDE GIRARDIN[1,2] AND MARCO PICASSO[2]

[1] Geneva School of Business Administration (HEG), University of Applied Sciences and Arts Western Switzerland (HES-SO), alexandre.caboussat@hesge.ch, maude.girardin@hesge.ch

[2] Institute of Mathematics, Ecole polytechnique fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland, marco.picasso@epfl.ch, maude.girardin@epfl.ch

Key words: Error estimates; Parametric PDEs; Neural networks; Hybrid methods; Transport equation.

Summary. We consider a hybrid approach for the approximation of the solution to parametric partial differential equations based on finite elements and deep neural networks. Finite element simulations with adaptive mesh refinement are used to generate input data for the training of a neural network. A deep feedforward neural network is then used to approximate the solution of the partial differential equation. We aim at balancing the numerical errors introduced by the finite element method and the neural network approximation respectively. Numerical results are presented for the transport equation.

1 INTRODUCTION

We consider a parametric partial differential equation

$$\mathcal{F}(u(\mathbf{x},t;\boldsymbol{\mu});\boldsymbol{\mu}) = 0 \quad \mathbf{x}\in\Omega, \quad t\in(0,T), \quad \boldsymbol{\mu}\in\mathcal{P}, \tag{1}$$

where $0 < T < \infty$ is the final time, $\Omega \subseteq \mathbb{R}^2$ is the physical space, $\mathcal{P} \subset \mathbb{R}^p$ (with $p \geq 1$ possibly large) denotes the parameter space, and $\mathcal{F}$ is a differential operator. We consider here a scalar function $u : \Omega \times (0,T) \times \mathcal{P} \to \mathbb{R}$.

Finite elements can be used to approximate the solution of (1) for a fixed $\boldsymbol{\mu} \in \mathcal{P}$. When solved repeatedly, several reduced order modeling approaches have been proposed to overcome the increasing computational time, such as, e.g., reduced basis [9], proper orthogonal decomposition [13] or polynomial chaos expansion [6]. Neural networks have also been used to approximate solutions of parametric partial differential equations [3, 8, 10, 14], and offer a viable alternative in the many-query context.

The objective of this work is to build a neural network to approximate the *solution map* $(\mathbf{x},t;\boldsymbol{\mu}) \mapsto u(\mathbf{x},t;\boldsymbol{\mu})$, where u is the solution of (1). In order to do so, we use

a fully connected feedforward neural network and we generate training data with finite element simulations. These are potentially time consuming but can be performed *offline*. Moreover, in order to increase the efficiency and accuracy of the simulations, an adaptive method is advocated. After training, the neural network approximation $u_{\mathcal{N}} : \Omega \times (0,T) \times \mathcal{P} \to \mathbb{R}$ can be efficiently evaluated *online*.

The error assessment of $u_{\mathcal{N}}$ involves the norm defined by:

$$|||u - u_{\mathcal{N}}|||^2 := \frac{1}{|\mathcal{P}|} \int_{\mathcal{P}} \frac{1}{T} \int_0^T ||u(\cdot,t;\boldsymbol{\mu}) - u_{\mathcal{N}}(\cdot,t;\boldsymbol{\mu})||^2 dt \; d\boldsymbol{\mu} \tag{2}$$

where

$$||u(\cdot,t;\boldsymbol{\mu}) - u_{\mathcal{N}}(\cdot,t;\boldsymbol{\mu})||^2 := \frac{1}{|\Omega|} \int_\Omega |u(\mathbf{x},t;\boldsymbol{\mu}) - u_{\mathcal{N}}(\mathbf{x},t;\boldsymbol{\mu})|^2 d\mathbf{x}.$$

For any value of the parameter $\boldsymbol{\mu} \in \mathcal{P}$, let us denote by $u_h(\cdot,\cdot;\boldsymbol{\mu}) : \Omega \times (0,T) \to \mathbb{R}$ the finite element approximation of $u(\cdot,\cdot;\boldsymbol{\mu})$. The error (2) can thus be decomposed as

$$|||u - u_{\mathcal{N}}||| \leq |||u - u_h||| + |||u_h - u_{\mathcal{N}}|||. \tag{3}$$

In this article, we apply more specifically this framework to a 2D transport equation with a given velocity field. The objective is to estimate the two terms on the right-hand side of (3) and to investigate how to balance them. This article is organized as follows. The finite element approximation framework is discussed in Section 2, and an adaptive finite element method is briefly sketched. Section 3 details how the approximation $u_{\mathcal{N}}$ can be obtained using a fully connected neural network. Finally, Section 4 illustrates the error assessment with one particular numerical experiment.

2 FINITE ELEMENT APPROXIMATION

We start by building finite element approximations u_h of u to generate training data for the neural network. To do so, we consider the adaptive finite element algorithm in space and time presented in [5], in order to ensure that the finite element error at the final time T is close to a preset tolerance TOL, for each parameter $\boldsymbol{\mu}$. More precisely, we want to ensure that:

$$0.75\ TOL \leq ||u(\cdot,T;\boldsymbol{\mu}) - u_h(\cdot,T;\boldsymbol{\mu})||^2_{L^2(\Omega)} \leq 1.25\ TOL. \tag{4}$$

Since the exact solution is not known in general, the error is replaced by an a posteriori estimator η; we adapt the time steps and the corresponding meshes on which the finite element solution is computed in such a way that

$$0.75\ TOL \leq \eta \leq 1.25\ TOL.$$

The meshes are adapted using the `BL2D` mesh generator [1]. The expression of the error estimator η and a precise description of the adaptive algorithm are given in [5, Section 3.3] and [5, Section 4.2] respectively.

In order to estimate the finite element error $|||u-u_h|||$, we use the Monte-Carlo method; we draw M parameters $\{(t_k, \boldsymbol{\mu}_k)\}_{k=1}^M$ randomly according to a uniform distribution in $[0,T] \times \mathcal{P}$ and we approximate:

$$|||u - u_h|||^2 \simeq |||u - u_h|||_M^2 := \frac{1}{M} \sum_{k=1}^{M} ||u(\cdot, t_k; \boldsymbol{\mu}_k) - u_h(\cdot, t_k; \boldsymbol{\mu}_k)||^2.$$

The error $|||u - u_h|||_M^2$ will be compared to the corresponding estimated error

$$\eta_M^2(u_h) := \frac{1}{M} \frac{1}{|\Omega|} \sum_{k=1}^{M} \eta^2(u_h(\cdot, t_k; \boldsymbol{\mu}_k); \boldsymbol{\mu}_k),$$

where η has been introduced in (4).

3 NEURAL NETWORKS

A fully connected feedforward neural network – see, e.g., [11] – is made up of an input layer, an output layer and $L \geq 1$ hidden layers. We denote by n_j the number of neurons of the j^{th} layer, $j = 0, \ldots, L+1$, and by σ_i^j and z_i^j the activation function and the value associated to the i^{th} neuron of the j^{th} layer respectively, $j = 0, \ldots, L+1, i = 1, \ldots, n_j$. Possible activation functions for neurons in hidden layers are the hyperbolic tangent, the Rectified Linear Unit ($ReLU(x) = \max\{x, 0\}$), or the softplus function ($softplus(x) = \ln(1 + e^x)$). For neurons in the output layer, the activation function is the identity. The value associated to a neuron is recursively given by

$$z_i^j = \sigma_i^j \Big(\sum_{k=1}^{n_{j-1}} a_{ik}^j z_k^{j-1} + b_i^j \Big), \quad j = 1, \ldots L+1, \quad i = 1, \ldots, n_j,$$

where a_{ik}^j and b_i^j are respectively the weights and the biases of the neural network. We denote by $\boldsymbol{\theta}$ the set of trainable parameters of the network, i.e. the set of all a_{ik}^j and b_i^j. Similarly as in [4], we denote by $\Upsilon^{W,L}(\sigma; d_{in}, d_{out})$ the set of fully-connected feedforward neural networks with input dimension d_{in}, output dimension d_{out} and L hidden layers, each constituted of W neurons having σ as activation function.

To build feedforward neural networks that approximate the solution to (1), we consider a neural network $\mathcal{N} \in \Upsilon^{W,L}(\sigma; 2+1+p, 1)$, for which $(\mathbf{x}, t; \boldsymbol{\mu})$ is the input and $u_{\mathcal{N}}(\mathbf{x}, t; \boldsymbol{\mu})$ is the output. The full algorithm reads as follows:

1. Choose the training parameters $\{\boldsymbol{\mu}_j\}_{j=1}^{N_\mu} \subseteq \mathcal{P}$.

2. For each $\boldsymbol{\mu}_j$, run the adaptive algorithm so that (4) is satisfied. Thus, the finite element solution is known at times $0 < t_j^n < T$, $n = 1, \ldots, N_j^T$. For every t_j^n, an

adapted mesh $\mathcal{T}_{j,n}$ of Ω with vertices $\{\mathbf{x}_{j,n}^i\}_{i=1}^{N_{j,n}}$ is available so that the finite element approximation of $u(\mathbf{x}, t_j^n; \boldsymbol{\mu}_j)$ then reads

$$u_h(\mathbf{x}, t_j^n; \boldsymbol{\mu}_j) = \sum_{i=1}^{N_{j,n}} U_{j,n}^i \varphi_{j,n}^i(\mathbf{x}),$$

where $\{\varphi_{j,n}^i\}_{i=1}^{N_{j,n}}$ are the piecewise linear finite element basis functions associated with the particular mesh $\mathcal{T}_{j,n}$, and $U_{j,n}^i = u_h(\mathbf{x}_{j,n}^i, t_j^n; \boldsymbol{\mu}_j)$ are the values at the nodes. Note that, in practice, we keep in the training set only the data at every 100 time steps of each finite element simulation.

3. Choose the architecture of the neural network, i.e., the parameters L, W and σ. Determine the optimal parameters $\boldsymbol{\theta}$ of $\mathcal{N} \in \Upsilon^{W,L}(\sigma; 2+1+p, 1)$ by minimizing:

$$\Phi(\boldsymbol{\theta}) := \mathcal{L}_{N_\mu}(u_{\mathcal{N}}(\cdot; \boldsymbol{\theta}); u_h).$$

where

$$\mathcal{L}_{N_\mu}(u_{\mathcal{N}}(\cdot; \boldsymbol{\theta}); u_h) := \frac{1}{\sum_{j=1}^{N_\mu} \sum_{n=1}^{N_j^T} N_{j,n}} \sum_{j=1}^{N_\mu} \sum_{n=1}^{N_j^T} \sum_{i=1}^{N_{j,n}} \frac{|\Omega(\mathbf{x}_{j,n}^i)|}{3|\Omega|} |u_{\mathcal{N}}(\mathbf{x}_{j,n}^i, t_j^n; \boldsymbol{\mu}_j; \boldsymbol{\theta}) - u_h(\mathbf{x}_{j,n}^i, t_j^n; \boldsymbol{\mu}_j)|^2,$$

and $|\Omega(\mathbf{x}_{j,n}^i)| = \sum_{\substack{K \in \mathcal{T}_h \\ \mathbf{x}_{j,n}^i \in K}} |K|$. Let $\boldsymbol{\theta}^*$ be the set of optimal parameters, obtained by a gradient descent type algorithm and denote the optimal solution $u_{\mathcal{N}}(\mathbf{x}, t; \boldsymbol{\mu}; \boldsymbol{\theta}^*)$ simply by $u_{\mathcal{N}}(\mathbf{x}, t; \boldsymbol{\mu})$.

4. The map provided by the network

$$\begin{aligned} u_{\mathcal{N}} : \mathbb{R}^2 \times \mathbb{R} \times \mathbb{R}^p &\to \mathbb{R} \\ (\mathbf{x}, t; \boldsymbol{\mu}) &\mapsto u_{\mathcal{N}}(\mathbf{x}, t; \boldsymbol{\mu}) \end{aligned}$$

can then be used to approximate u_h and u.

The neural networks are built and trained using the open source library `Keras` [2]. We take $\sigma = softplus$ as activation function and we initialize the weights of the networks with the Glorot Normal initialization [7]. The neural networks are trained with the `Nadam` optimizer [12], using an initial learning rate of 0.001, which is decreased when a plateau is reached, batches of size $bs = 1024$ and early stopping. The training set $(\mathbf{x}_{j,n}^i, t_j^n; \boldsymbol{\mu}_j)$ is normalized such that all the components have mean zero and standard deviation one; this ensures that all data have the same magnitude [2].

To estimate the error of the neural network $|||u_{\mathcal{N}} - u_h|||$, we again use the Monte-Carlo method to approximate the integral over $[0,T] \times \mathcal{P}$, and then a quadrature formula of (sufficiently) high order to compute the integrals over the domain Ω.

In the next section, a model problem based on a particular transport equation is presented. Both the accuracy of the finite element method and of the neural network are studied, in an attempt to balance both error contributions.

4 NUMERICAL EXPERIMENTS

As a model problem, we consider the parametric transport equation in two dimensions of space and set $p = 3$. Given $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3) \in \mathcal{P}$, the problem consists in finding $u : \Omega \times [0,T] \mapsto \mathbb{R}$ satisfying:

$$\left\{ \begin{array}{rcll} \dfrac{\partial u}{\partial t}(\mathbf{x},t;\boldsymbol{\mu}) \ + \mathbf{a}(\mathbf{x},t;\boldsymbol{\mu})\nabla u(\mathbf{x},t;\boldsymbol{\mu}) & = & 0 & \mathbf{x} \in \Omega,\ t \in (0,T),\ \boldsymbol{\mu} \in \mathcal{P} \\ u(\mathbf{x},0;\boldsymbol{\mu}) & = & u_0(\mathbf{x};\boldsymbol{\mu}) & \mathbf{x} \in \Omega,\ \boldsymbol{\mu} \in \mathcal{P}, \end{array} \right. \tag{5}$$

with $\Omega = (0,4) \times (0,4)$ and $T = 2|\mu_1|^{-1}$. The vector field $\mathbf{a}$ is chosen such that there is no inflow boundary and thus no boundary condition to enforce. We consider

$$\mu_1 \in [-2,-0.5] \cup [2,0.5], \qquad \mu_2 \in [0.15,0.3], \qquad \mu_3 \in [50,150],$$

and we set

$$\begin{aligned} \mathbf{a}(\mathbf{x},t;\boldsymbol{\mu}) &= \frac{\mu_1 \pi}{2} \begin{pmatrix} 2 - x_2 \\ x_1 - 2 \end{pmatrix} \\ u_0(\mathbf{x};\boldsymbol{\mu}) &= \tanh\left(-\mu_3 \sqrt{(x_1-2)^2 + (x_2-2.5)^2} - \mu_2\right). \end{aligned}$$

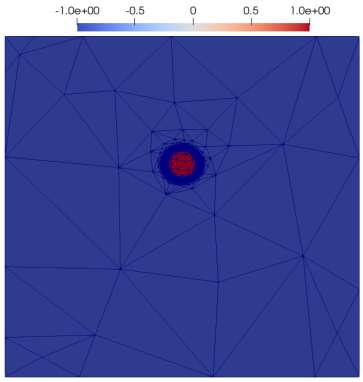
In this particular case, μ_1 is a parameter ruling the velocity, μ_2 characterizes the size of the support of the initial condition u_0, and μ_3 is a regularization parameter for the initial condition. Note that, for this special case, the exact solution u is known and given by

$$u(\mathbf{x},t;\boldsymbol{\mu}) = u_0(\mathbf{X}(\mathbf{x},t;\boldsymbol{\mu});\boldsymbol{\mu}),$$

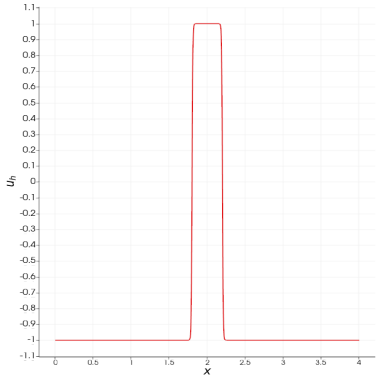
with

$$\mathbf{X}(\mathbf{x},t;\boldsymbol{\mu}) = \begin{pmatrix} \cos\left(\dfrac{\mu_1\pi}{2} t(x_1-2)\right) + \sin\left(\dfrac{\mu_1\pi}{2} t(x_2-2)\right) + 2 \\ -\sin\left(\dfrac{\mu_1\pi}{2} t(x_1-2)\right) + \cos\left(\dfrac{\mu_1\pi}{2} t(x_2-2)\right) + 2 \end{pmatrix}.$$

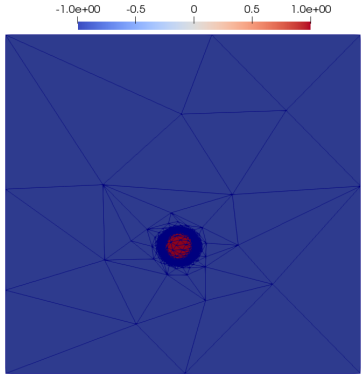
Figure 1 shows an example of the finite element approximation $u_h(\cdot,t;\boldsymbol{\mu})$ and the corresponding mesh at initial and final times, when the adaptive algorithm is performed with $TOL = 0.025$.



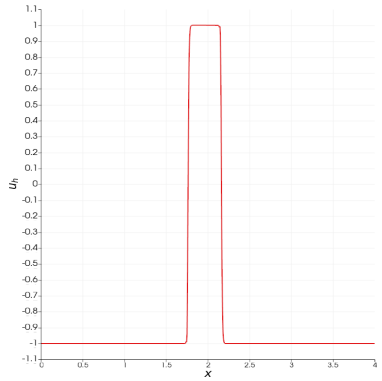
(a) Adapted mesh at $t = 0$.



(b) Cut along x_1-axis, $x_2 = 2.5$, $t = 0$.



(c) Adapted mesh at $t = 1$.



(d) Cut along x_1-axis, $x_2 = 1.5$, $t = 1$.

Figure 1: Finite element solution $u_h(\cdot, t; \boldsymbol{\mu})$ for $\boldsymbol{\mu} = (2, 0.2, 100)$ and $TOL = 0.025$. Left: snapshot of the solution and corresponding adapted mesh. Right: Cut of the solution along horizontal axes.

To approximate the solution of (5) and since $p = 3$, we consider here neural networks belonging to $\Upsilon^{W,L}(\sigma; 6, 1)$. We are interested in the convergence of the finite element error $|||u - u_h|||_M^2$, of the error estimator η_M^2 and of the neural network error $|||u_{\mathcal{N}} - u_h|||_M^2$ with respect to the tolerance TOL. Numerical results are reported in Figure 2 for $N_\mu = 100$ and $M = 100$. We first note that both the finite element error $|||u - u_h|||_M^2$ and the estimator η_M^2 converge as $\mathcal{O}(TOL^2)$, as required. Next, we observe that the accuracy of the neural network also increases as TOL decreases. This can be expected since, as the tolerance decreases, the number of times at which the finite element solution is computed increases, as well as the number of vertices in each adapted mesh. Therefore, the number of training data – for the same N_μ – increases as TOL decreases. Thanks to this fact, we were able to balance the error of the neural network and of the finite element method for $N_\mu = 100$ and for the four tolerances tested here. Figure 3 illustrates the finite element solution and the neural network approximation for various choices of parameters, and shows similar accuracy, a good approximation of the boundary layers, and limited overshoot/undershoot phenomena.

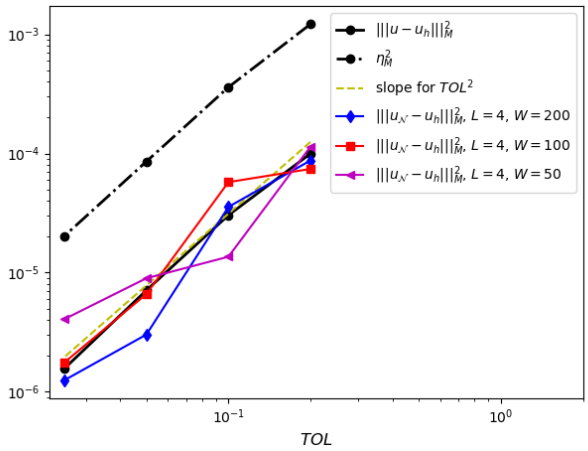
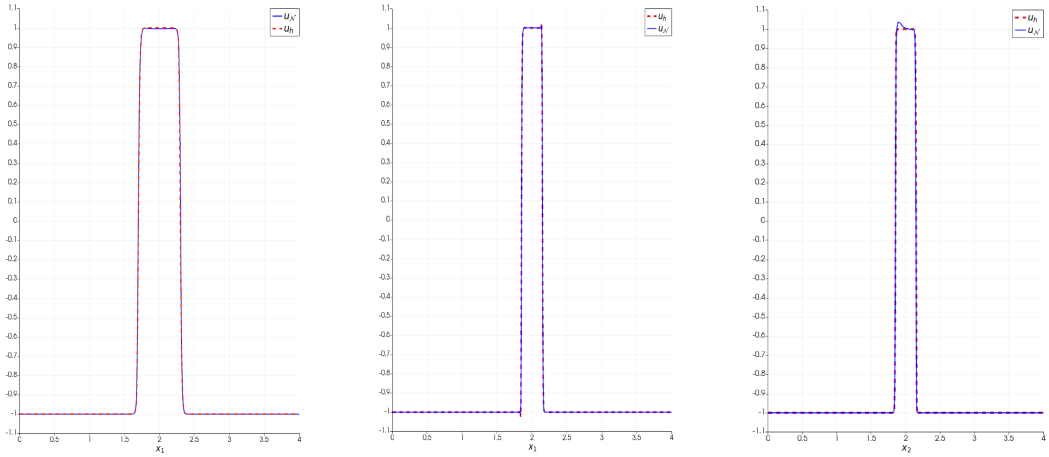


Figure 2: Convergence behaviour with respect to TOL of the finite element approximation, the a posteriori error estimator, and several neural network approximations for $N_\mu = 100$ and $M = 100$.



(a) Cut along x_1-axis, $x_2 = 1.5$, $\boldsymbol{\mu} = (2, 0.3, 50)$, $t = 1$.

(b) Cut along x_1-axis, $x_2 = 1.5$, $\boldsymbol{\mu} = (2, 0.15, 150)$, $t = 1$.

(c) Cut along x_2-axis, $x_1 = 1.5$, $\boldsymbol{\mu} = (5, 0.15, 150)$, $t = 2$.

Figure 3: Snapshots of u_h and $u_{\mathcal{N}}$ for $\mathcal{N} \in \Upsilon^{200,4}(6, 1, softplus)$, $N_\mu = 100$ and $TOL = 0.025$.

REFERENCES

[1] H. Borouchaki and P. Laug. The BL2D mesh generator: Beginner's guide, user's and programmer's manual. 08 1996.

[2] F. Chollet et al. Keras. https://keras.io, 2015.

[3] N. Dal Santo, S. Deparis, and L. Pegolotti. Data driven approximation of parametrized PDEs by reduced basis and neural networks. *J. Comput. Phys.*, 416:109550, 2020.

[4] R. DeVore, B. Hanin, and G. Petrova. Neural network approximation. *Acta Numerica*, 30:327–444, 2021.

[5] S. Dubuis and M. Picasso. An adaptive algorithm for the time dependent transport equation with anisotropic finite elements and the Crank–Nicolson scheme. *J. Sci. Comput.*, 75:350–375, 2018.

[6] Oliver G Ernst, Antje Mugler, Hans-Jörg Starkloff, and Elisabeth Ullmann. On the convergence of generalized polynomial chaos expansions. *ESAIM: Mathematical Modelling and Numerical Analysis*, 46(2):317–339, 2012.

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Sardinia, Italy, 13–15 May 2010. PMLR.

[8] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, aug 2018.

[9] J. S. Hesthaven, G. Rozza, B. Stamm, et al. *Certified reduced basis methods for parametrized partial differential equations*, volume 590. Springer, 2016.

[10] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[11] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[12] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[13] S. Volkwein. Model reduction using proper orthogonal decomposition. *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz. see http://www. uni-graz. at/imawww/volkwein/POD. pdf*, 1025, 2011.

[14] N. Yadav, A. Yadav, M. Kumar, et al. *An introduction to neural network methods for differential equations*. Springer, 2015.