Meethal *et al. Advanced Modeling and Simulation in Engineering Sciences(2023)10:6*
https://doi.org/10.1186/s40323-023-00243-1

Advanced Modeling and Simulation in Engineering Sciences

RESEARCH ARTICLE Open Access

Finite element method-enhanced neural network for forward and inverse problems

Rishith E. Meethal[1,2*], Anoop Kodakkal[2*], Mohamed Khalil[1], Aditya Ghantasala[2], Birgit Obst[1], Kai-Uwe Bletzinger[2] and Roland Wüchner[3]

*Correspondence: rishith.ellath_meethal@siemens.com; anoop.kodakkal@tum.de
[1]Technology, Siemens AG, Munich, Germany
[2]Chair of Structural Analysis, Technical University of Munich, Munich, Germany
[3]Institute of Structural Analysis, Technische Universität Braunschweig, Braunschweig, Germany

Abstract

We introduce a novel hybrid methodology that combines classical finite element methods (FEM) with neural networks to create a well-performing and generalizable surrogate model for forward and inverse problems. The residual from finite element methods and custom loss functions from neural networks are merged to form the algorithm. The Finite Element Method-enhanced Neural Network hybrid model (FEM-NN hybrid) is data-efficient and physics-conforming. The proposed methodology can be used for surrogate models in real-time simulation, uncertainty quantification, and optimization in the case of forward problems. It can be used to update models for inverse problems. The method is demonstrated with examples and the accuracy of the results and performance is compared to the conventional way of network training and the classical finite element method. An application of the forward-solving algorithm is demonstrated for the uncertainty quantification of wind effects on a high-rise buildings. The inverse algorithm is demonstrated in the speed-dependent bearing coefficient identification of fluid bearings. Hybrid methodology of this kind will serve as a paradigm shift in the simulation methods currently used.

Keywords: Hybrid models, Informed machine learning, FEM-based neural network, Self-supervised learning

Introduction

Developments in the field of Artificial Intelligence (AI) [1] have lead to substantial improvements in everyday life. Advances in the sub-fields of AI such as data science, machine learning, deep learning, neural networks are contributing to diverse research and application fields. These include computer vision [2], speech and language processing [3], drug discovery [4,5], genomics [6], computer games [7], animation [8], robotics [9], and many more. Deep learning has boosted the development of computer vision by contributing extensively to image classification [10], object detection [11], and semantic segmentation [12]. Similarly, AI has also made a large number of contributions to computer games [13–15]. An important direction in this field is game physics, where smoke and fluid flows for computer graphics are simulated with the help of neural networks. Jonathan et al. [16] proposed a data-driven solution to the inviscid-Euler equations that

Springer Open

© The Author(s) 2023. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

is faster than traditional methods used in computer graphics animations. Similar to the developments in game physics, different AI methods have found their application in solving Partial Differential Equations (PDE) for physical problems.

Neural networks [17] are employed to perform physical systems simulations as well [18–20]. One of the initial work was on designing simple beams using the perceptron (basic unit of a neural network). Recently neural networks are used to perform simulations of complex systems as well. Thuerey et al. [19] used U-net architecture to model the flow around airfoils. Their study resulted in surrogate models that can predict velocity and pressure fields around an unseen airfoil profile with relative error less than 3%. Tobias et al. [20] showed that graph networks can be used to learn mesh-based simulations. The method is demonstrated on different types of physical systems such as fluids and structural. Different types of neural networks such as LSTM, CNN have found their applications in performing physical simulations. Guo et al. [21] used convolutional neural networks (CNNs) for the steady flow approximations. Zhang et al. [22] applied a multiLSTM neural network for mapping the excitation force to the response of the system.

Generally, conventional approaches use input and output data to create neural network-based surrogate models approximating the mapping function between them. Such surrogate models, once trained, can be used for performing faster simulation. They are useful when large number of simulation runs with different input parameters are required for applications such as optimization or uncertainty quantification. However, this approach has two significant shortcomings. The first is the high computational cost of creating training data. This is because the simulation results required to make a surrogate model are generated by running a large number of numerical simulations. Running a large number of simulations requires extensive computational power and simulation time. The second problem is that the training algorithm does not consider underlying physics. It results in neural networks that are loosely informed about the underlying physics. A loosely informed surrogate does not extrapolate well if the training data does not cover all of the required range. However, it is crucial for a numerical simulation that the surrogate model created follows the underlying physics described by the PDE.

To address the issues mentioned above, more recent developments exploit the prior information about the data by incorporating it into the learning process. Laura et al. [23] provide a structured overview of various approaches to integrate different prior knowledge into the training process. They consider source of knowledge, its representation, and its integration into the machine learning pipeline to introduce a taxonomy that serves as a classification for informed machine learning frameworks. They describe how different knowledge representations such as logic rules, algebraic equations, or simulation results can be used in machine learning pipeline. For example, logic rules are widely used in linguistics [24,25]. For example, a rule stating a state transition in a sentence can occur after a punctuation helps the model to identify and label each part in a sentence easily in a semi-supervised learning setup. Another example for the use of prior knowledge is the use of Newtonian mechanics to improve the object detection and tracking in videos [26].

When it comes to physical simulations, there are numerous works which integrated prior-knowledge into the learning algorithm. The most appreciated work in this direction is the physics-informed neural network (PINN). The PINNs proposed in [27] have shown how existing knowledge regarding the physics of the data can be used to constrain the neural network to follow the physics. This is accomplished by embedding the physics in

the form of the partial differential equation into the loss function of the neural network using automatic differentiation [28]. By employing the measured or availble data, the same approach is extended for inverse problems as well. The resulting models showed promising results for a diverse collection of problems in computational science. Some of the notable examples are CFD [29], heat transfer [30] and power systems [31]. PINNs are also shown to be successful in dealing with integro-differential equations [32], fractional PDEs [33] and stochastic PDEs [34].

However, it is stated in [27] itself that the method does not replace classical numerical methods in terms of the robustness and computational efficiency required in practice. For examples, the training time required for the model is much higher compared to the classical numerical methods. Karniadakis et al. in [35] explains that PINNs are particularly effective in solving ill-posed and inverse problems, whereas for forward, well-posed problems that do not require any data assimilation the existing numerical grid-based solvers currently outperform PINNs. Wang et al. [36] shows that fully connected neural networks fail to learn solutions containing high-frequencies and multi-scale features. They showed that along with the "spectral bias", the PINNs also exhibit a discrepancy in the convergence rate among the different loss components contributing to the total training error. Krishnapriyan et al. in [37] also shows that PINNs works well with relatively trivial problems, but fail to learn the physics for even slightly more complex problems. They demonstrate that the soft regularization used in PINNs make the problem more ill-conditioned. They introduce curriculum learning and sequence-to-sequence to learning to address this issue. However, none of those address the ill-conditioned problem statement resulting from the soft-regularization used by PINNs.

In this contribution, we introduce a data-efficient and physics-conforming hybrid methodology for generating numerically accurate surrogate models. It falls in the category of informed machine learning [23], where expert knowledge in the form of differential equations is used in the hypothesis set and learning algorithm of neural networks. In contrast to the state-of-the-art PINNs, we use the discretized form of the well-posed problem statement for the learning purpose. The methodology involves training the neural network with FEM-based custom loss function and deploying it with the FEM. The use of FEM matrices after the application of boundary conditions makes the problem well-posed. This also results in physics-conforming training. Using the prediction along with the FEM makes the prediction error quantifiable. The introduced novel algorithm is expanded for solving the inverse problems as well.

The rest of the paper is structured as follows: "Algorithm" section explains the algorithm used behind the novel hybrid model. Algorithms for both forward and inverse problems are explained in detail here. The novelty achieved in terms of data efficiency and physics conformity is compared with conventional neural network-based surrogate models for simulation. The proposed model is demonstrated with examples in "Results and discussion" section. The results are compared with FEM-based simulation and conventional neural networks. Applications of these methods for uncertainty quantification of wind load on high-rise building and parameter identification of fluid bearing are discussed in "Applications" section. "Conclusion and outlook" section concludes the discussion and proposes different future steps.

Algorithm

Finite element method

A numerical approximation of the continuous solution field u of any partial differential equation (PDE) given by Eq. (1) on a given domain Ω can be performed by various methods. Some of the widely used techniques include finite element method [38], finite volume method [39], particle methods [40], and finite cell method [41]. In this contribution, we restrict the discussion to Galerkin-based finite element methods.

$$\mathcal{L}(u) = 0 \quad \text{on } \Omega \tag{1}$$

$$u = u_d \quad \text{on } \Gamma_D \tag{2}$$

$$\frac{\partial u}{\partial x} = g \quad \text{on } \Gamma_N \tag{3}$$

Consider the PDE in Eq. (1) defined on a domain Ω together with the boundary conditions given by Eqs. 2 and 3. Here, u_d and g are the Dirichlet and Neumann boundary conditions on the respective boundaries. A finite element formulation of Eq. (1) on a discretization of the domain with m elements and n nodes, together with boundary conditions, will result in the system of equations shown by Eq. (4). We assume all the necessary conditions on the test and trial spaces [38] are fulfilled.

$$\underbrace{\begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & \cdots & k_{n,n} \end{pmatrix}}_{K(u^h)} \underbrace{\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}}_{u^h} = \underbrace{\begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{pmatrix}}_{F} \tag{4}$$

In Eq. (4), $K(u^h)$ is the non-linear left hand side matrix, also called the stiffness matrix. u^h is the discrete solution field, and F is the right hand side vector. The residual of the system of equations in Eq. (4) can be written as

$$r(u^h) = K(u^h)u^h - F \tag{5}$$

To obtain the solution u^h, a Newton–Raphson iteration technique can be employed using the linearization of $r(u^h)$ and its tangent matrix. This requires the solution of a linear system of equations in every iteration. These iterations are carried out until the residual norm $\|r\|_n$ meets the tolerance requirements. For a detailed discussion of the methodology, readers are referred to [42]. For this residual-based formulation, in case of a linear operator $\mathcal{L}$, it takes only one iteration to converge. For a large number of elements and nodes, among different steps of the finite element methodology, the most computationally demanding step is the solution of the linear system of equations. In an application where computational efficiency is critical, like real time simulations [43] and digital twins [44], it is imperative that this step be avoided. Techniques suitable for such applications, like model order reduction [45,46], construct a surrogate model of Eq. (4) to reduce this cost significantly. Techniques involving neural-networks can completely avoid this cost, but will require a significant amount of training and test data, which is typically generated by simulating the underlying finite element problem. In "Finite element method-enhanced neural network for forward problems" section, we discuss an algorithm that combines residual information from a numerical method to train a neural network for linear PDEs.

In this case the residual $r(u^h)$ becomes

$$r(u^h) = Ku^h - F \tag{6}$$

Neural network and custom loss

Consider a neural network mapping from input variables $\mathbf{x} = [x_1, x_2, ...x_m]$ of domain X to output variables $\mathbf{y} = [y_1, y_2, ...y_n]$ of domain Y. Input variables are given to the input layer of the neural network so as to obtain output variables from the output layer. The input layer is connected to the output layer by L number of layers in between, called hidden layers. Each hidden layer receives input from the previous layer and outputs $o^l = [o_1^l, o_2^l, \ldots, o_k^l]$. The $w_l \in \mathbb{R}^{n_l \times n_l + 1}$ matrix represents the weights between layer l and $l+1$ and the vector b_l of size n_l represents the bias vector from layer l. Here, n_l represents the number of neurons in each layer of the neural network.

The output from any layer is transformed as below before sending it to the next layer.

$$z^l = w^l o^{l-1} + b^l \tag{7}$$

The nonlinear activation function $\sigma(.)$ is applied to each component of the transformed vector z^l before sending it as an input to the next layer.

$$o^l = \sigma(z^l) \tag{8}$$

Following this sequence for all the layers, the output of the entire neural network can be written as

$$y(\theta) = \sigma^L(z^L.....\sigma^2(z^2.\sigma^1(z^1(x)))) \tag{9}$$

where y is the output vector for the given input vector x, and $\theta = \{w^l, b^l\}_{l=1}^L$ is the set of trainable parameters of the network.

The network is trained by treating it as a minimization problem. The objective function for minimization is a function of y and y^t, called the loss function in the neural network community. Here, y is the predicted value from the neural network and y^t is the actual value which is either measured, or simulated.

The calculated loss δ is reduced by updating the trainable parameters in the process called backpropagation [17]. In backpropagation, the gradient of δ with respect to trainable parameters is calculated, and the trainable parameters are updated as follows,

$$w^l = w^l - \eta \frac{\partial \delta}{\partial w^l} \tag{10}$$

The derivatives of δ with respect to the trainable parameters are calculated using the chain rule. The parameter η is called learning rate and is chosen by the user.

$$\begin{aligned} \frac{\partial \delta}{\partial w^l} &= \frac{\partial \delta}{\partial y} \frac{\partial y}{\partial w^l} \\ \frac{\partial \delta}{\partial b^l} &= \frac{\partial \delta}{\partial y} \frac{\partial y}{\partial b^l} \end{aligned} \tag{11}$$

In Eq. (11), the derivative of output y with respect to trainable parameters $\frac{\partial y}{\partial w^l}$ and $\frac{\partial y}{\partial b^l}$ is calculated by considering the network's architecture and activation functions. Implementations for the calculation of the aforementioned derivatives are readily available in machine learning libraries like, TensorFlow [47], PyTorch [48], etc. But the first part of Eq. (11) depends on the chosen loss function. One of the common loss functions for minimization among neural network communities is the mean square error (MSE) between

the true value for y^t and the predicted value for y. The true value is taken from the training data and the prediction is the network output. The loss function $\delta(y, y^t)$ is given by

$$\delta_{MSE} = \frac{1}{m}\sum_{i=i}^{m}(y_i - y_i^t)^2 \tag{12}$$

For δ_{MSE}, the first part of Eq. (11) is the derivative of δ_{MSE} with respect to the output of the neural network y given by,

$$\frac{\partial \delta}{\partial y} = 2\frac{1}{m}\sum_{i=i}^{m}(y_i - y_i^t) \tag{13}$$

But there are large variety of neural networks that use different other loss functions. Some statistical applications uses loss functions such as the mean absolute percentage error (MAPE) for training.

$$\delta_{MAPE} = \frac{1}{m}\sum_{i=i}^{m}\left|\frac{(y_i - y_i^t)}{y_i^t}\right| \tag{14}$$

In this case the first part of the Eq. (11), $\frac{\partial \delta}{\partial y}$, will have a different function when compared with that used for MSE in Eq. (13).

Finite element method-enhanced neural network for forward problems

The proposed algorithm combines FEM and a neural network to produce a surrogate model, which we call the FEM enhanced neural network hybrid model. The method is explained with the help of the flowchart given in Fig. 1. The surrogate can be used to predict simulation results for parametric PDEs of the form $\mathcal{L}(u, \lambda) = 0$. The parameter λ for the PDE can be initial conditions, boundary conditions, material properties or geometrical properties. The model undergoes a training process before it is used as a surrogate model for simulation. During the training process, the variables for simulation are taken as input parameters. This include the parameters λ of the PDE, neural network parameters θ and constants for running the simulation C. The input parameters are processed by both the neural network and the FEM library. The neural network outputs u after the forward pass through the chosen network architecture. The output of the network is the discrete solution field vector u given as,

$$U = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \tag{15}$$

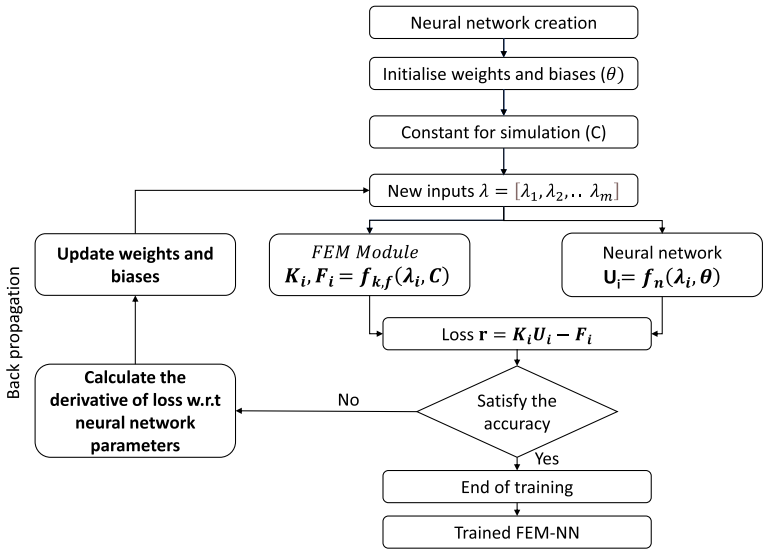
In the meantime, the FEM library takes the input variables and calculates the stiffness matrix K and the force vector F as explained in "Finite element method" Section.

The residual r is calculated using the prediction U from the neural network and K and F from FEM. Loss for the neural network prediction is defined as Euclidean norm of the residual vector r.

$$\delta = ||r||_2 \tag{16}$$

where r is given by

$$r = KU - F$$



(a) FEM-neural network training

Fig. 1 FEM-neural network training

$$
\begin{aligned}
&= \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & \cdots & k_{n,n} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} - \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{pmatrix} \\
&= \begin{pmatrix} k_{1,1}u_1 + k_{1,2}u_2 + .. + k_{1,n}u_n - F_1 \\ k_{2,1}u_1 + k_{2,2}u_2 + .. + k_{2,n}u_n - F_2 \\ \vdots \\ k_{n,1}u_1 + k_{n,2}u_2 + .. + k_{n,n}u_n - F_n \end{pmatrix}
\end{aligned} \tag{17}
$$

This gives loss δ as,

$$
\begin{aligned}
\delta &= \|r\|_2 \\
&= \sqrt{(k_{1,1}u_1 + k_{1,2}u_2 .. + k_{1,n}u_n - F_1)^2 .. + (k_{n,1}u_1 + k_{n,2}u_2 .. + k_{n,n}u_n - F_n)^2} \\
&= \sqrt{\sum_{j=1}^{n} \sum_{i=1}^{n} (K_{j,i}u_i - F_j)^2}
\end{aligned} \tag{18}
$$

As explained in "Neural network and custom loss" section, the learnable parameters are updated using backpropagation to minimize the loss. Backpropagation calculates the gradients of the loss with respect to the learnable parameters. Since we use a loss function specific to FEM, we need to calculate the second part of the Eq. (11), $\frac{\partial \delta}{\partial y}$, for the custom loss used here. In the case of a hybrid model, the output y of the neural network is the predicted discrete solution field u.

$$
y = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}
$$

The gradient has to be calculated with respect to each member of the output. $\frac{\partial \delta}{\partial y}$ becomes,

$$\begin{pmatrix} \frac{\partial \delta}{\partial u_1} \\ \frac{\partial \delta}{\partial u_2} \\ \vdots \\ \frac{\partial \delta}{\partial u_n} \end{pmatrix} = \frac{1}{2\delta} 2 \begin{pmatrix} k_{1,1}u_1 + k_{1,2}u_2 + .. + k_{1,n}u_n - F_1 \\ \vdots \\ k_{n,1}u_1 + k_{n,2}u_2 + .. + k_{n,n}u_n - F_n \end{pmatrix}^T \begin{pmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & \cdots & k_{n,n} \end{pmatrix}$$

which gives

$$\frac{\partial \delta}{\partial y} = \frac{r^T K}{\delta} \tag{19}$$

where r^T is the transpose of the residual vector r and K is the stiffness matrix. With the implementation of the above in the machine learning frameworks, we can train the network against the residual of the differential equation. The second part of the Eq. (11) is readily available in all the neural network frameworks. The parameters of the neural network are updated using the Eq. (11). The process is repeated until the residual r satisfies the desired accuracy level. The procedure for training and prediction is detailed in Algorithm 1.

Algorithm 1 FEM-NN training and prediction for forward problems

procedure TRAIN
  Read simulation parameters
  Initialize neural network
  Initialize weights and biases
  Let L be the number of layers in the neural network
  Initialize FEM Package
  Compute the system matrices K and F
  while not Stop Criterion **do**
    $U \leftarrow$ Neural network prediction
    Compute the residual $r = KU - F$
    Compute the loss as the Euclidean norm of the residual vector $\delta = ||r||_2$
    Compute the derivative $\frac{\partial \delta}{\partial u} = \frac{r^T \times K}{\delta}$
    for all $l \in \{1, \ldots, L\}$ **do**
      Compute the derivative of neural network parameters using chain rule $\frac{\partial \delta}{\partial w_l} = \frac{\partial \delta}{\partial u} \frac{\partial u}{\partial w_l}$
      Update trainable parameters (weights and biases) $w_l = w_l - \eta \frac{\partial \delta}{\partial w_l}$
    end for
  end while
end procedure

procedure PREDICT
  Initialize FEM Package
  Compute the system matrices K and F
  Predict U using the trained neural network
  Compute the Residual $r = KU - F$
  return U and r
end procedure

Once trained, the neural network can be deployed using a similar hybrid approach. During the deployment of the network (Fig. 2, the input variables pass through both the

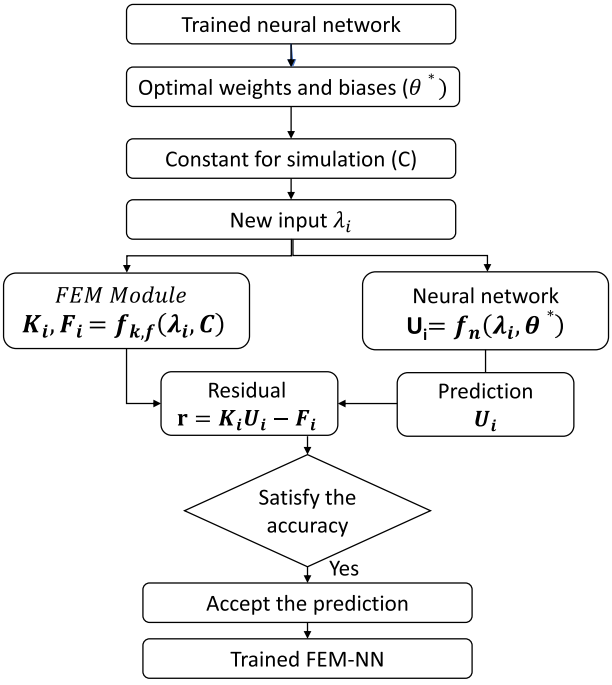


Fig. 2 FEM-neural network deployment

neural network and the FEM framework. The trained neural network predicts the output U. In a conventional way of deployment of networks, the prediction accuracy is not measurable. But here the output is used along with the stiffness matrix K and force vector F from FEM to calculate the residual r in Eq. (6). The residual r is a measure on how much the output deviates from the actual solution of the governing equation. Hence, the accuracy of prediction of FEM-neural network is measurable.

Finite element method-enhanced neural network for inverse problems

Inverse problems appear in many different applications of science and industry. Forward problems estimate the results for a defined cause, whereas inverse problems typically estimate the cause for the observed results. It is possible to use forward-solving software for inverse analysis as well. In such cases, the inverse problem is formulated as a parameter identification problem, where the unknown parameters of the forward problem are calculated by minimizing a suitable cost function. The estimation of unknown parameters results in correcting or updating the mathematical model used. Such a corrected or updated model can be used for different applications like simulation and prediction, optimization, system monitoring, fault detection, etc. This makes inverse problems vital in engineering. The algorithm introduced in "Finite element method-enhanced neural network for forward problems" section is extended for inverse problems as well.

In the case of inverse problems for Eq. (17), the primary variable u is known whereas forces F or stiffness matrix K can have unknown parts. As there are different kinds of inverse problems, we consider one category where a stiffness matrix has unknown parts for the rest of the paper. Such a problem can be described using the following equation

$$\begin{pmatrix} K_k & K_{ku} \\ K_{uk} & K_u \end{pmatrix} \begin{pmatrix} U_k \\ U_u \end{pmatrix} = \begin{pmatrix} F_k \\ F_u \end{pmatrix} \tag{20}$$

where K_k is the known part and K_u the unknown part of the system matrix. K_{ku} and K_{uk} are the cross coupling terms of the known and unknown parts of the system matrix. Similarly, U_u are the responses corresponding to unknown and U_k are the responses corresponding to the known parts of the system matrix. The loss for the neural network prediction is defined as the Euclidean norm of the residual vector r.

$$\delta = ||r||_2 \tag{21}$$

where r is given by

$$\begin{aligned} r &= KU - f \\ &= \begin{pmatrix} K_k & K_{ku} \\ K_{uk} & K_u \end{pmatrix} \begin{pmatrix} U_k \\ U_u \end{pmatrix} - \begin{pmatrix} F_k \\ F_u \end{pmatrix} \\ &= \begin{pmatrix} K_k U_k + K_{ku} U_u - F_k \\ K_{uk} U_k + K_u U_u - F_u \end{pmatrix} \end{aligned} \tag{22}$$

This gives loss δ as,

$$\delta = ||r||_2 = \sqrt{(K_k U_k + K_{ku} U_u - F_k)^2 + (K_{uk} U_k + K_u U_u - F_u)^2} \tag{23}$$

Similar to the calculation performed for forward problems, we need to calculate the derivative of the residual with respect to neural network prediction to perform the back-propagation. In the case of Eq. (23) it is $\frac{\partial \delta}{\partial K_u}$

$$\frac{\partial \delta}{\partial K_u} = \frac{1}{\delta}(K_{uk} U_k + K_u U_u - F_u) U_u \tag{24}$$

Equation 24 and the second part of the Eq. (11) are used to update the neural network parameters for training the neural network to identify the unknown part of the matrix.

The procedure for training and predicting for an inverse problem is detailed in Algorithm 2. One example for such an inverse problems is the stiffness identification of fluid bearings in a rotor-dynamic system demonstrated in "Fluid bearing stiffness identification" section.

Comparison between FEM-enhanced neural network and conventional neural networks for simulation

In this section, the discussed novel FEM-NN hybrid model for simulation and the conventional way of making neural network-based surrogates for simulation are compared.

Conventionally, a large number of simulations are run to produce the input and output data required for training the neural network. Hence, the training is a supervised training requiring a large amount of data. There are two main problems associated with this approach when it comes to the numerical simulation of physical phenomena. First, such models are constructed without considering the physics or the differential equation behind the problem. During the training phase, the network treats the input and output of the simulation as pure data. The equations governing the simulation or the physics behind the problem are not considered. The second problem is that a large number of samples are required to train the network. Since neural network training requires a large number of input–output pairs, we need to run a large number of simulations. Running such a large number of simulations is computationally expensive, especially when it comes to multiphysics simulations. In a nutshell, the shortcomings associated with traditional neural network surrogates are as follows.

Algorithm 2 FEM-NN training for inverse problems

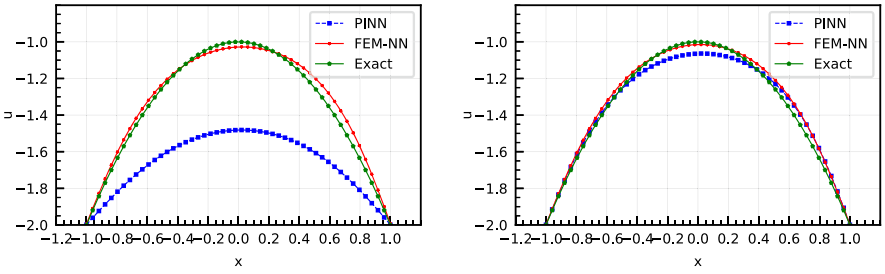
procedure TRAIN
  Read simulation parameters
  Initialize neural network
  Initialize weights and biases
  Initialize FEM Package
  Let L be the number of layers in the neural network
  while not Stop Criterion **do**
    Compute the system matrices $K_k, K_{ku}, K_{uk}, F_k, F_u, U_k$ and U_u
    $K_u \leftarrow$ Neural network prediction
    Compute the residual $r = KU - F$
    Compute the loss as the Euclidean norm of the residual vector $\delta = ||r||_2$
    Compute the derivative $\frac{\partial \delta}{\partial K_u} = \frac{1}{\delta}(K_{uk}U_k + K_u U_u - F_u)U_u$
    for all $l \in \{1, \ldots, L\}$ **do**
      Compute the derivative using chain rule $\frac{\partial \delta}{\partial w_l} = \frac{\partial \delta}{\partial K_u}\frac{\partial K_u}{\partial w_l}$
      Update trainable parameters (weights and biases) $w_l = w_l - \eta \frac{\partial \delta}{\partial w_l}$
    end for
  end while
end procedure

procedure PREDICT
  Initialize FEM Package
  Predict K_u
  Use the predicted K_u for forward simulation or other analysis
end procedure

- Neural network treat inputs and outputs as pure numbers and not as physical quantities.
- Training is done against the output data, not against the governing PDE describing the physics.
- Prediction can go wrong in an untrained scenario and the error of prediction is not measurable.
- A large number of computationally expensive simulations are required to create data.

The proposed FEM-NN solves the problems associated with conventional surrogates. The proposed algorithm uses the custom loss function defined in Eq. (18). The custom loss is based on the discretized version of the PDE, hence it follows the physics. Since the loss is based on the predictions from the neural network and matrices from FEM, it does not need target values or precalculated simulation results. Hence, the computationally expensive simulations need not be run. Though there are methods like PINNs, they fail to converge to the actual solution for multiple reasons. One of the main reasons is the multiobjective treatment of boundary conditions and PDE. This problem is overcome by using the introduced FEM-NN method.

- The network is trained against the equation, hence the physics is preserved.
- The computational cost of simulation is less as the training does not require target values
- The prediction comes with the residual of the linear system in Eq. (6). This makes the prediction accuracy measurable.



(a) Prediction after 1900 epochs **(b)** Prediction after 2500 epochs

Fig. 3 Comparison of FEM-NN and PINN along the training epochs

- Single loss term taking care of boundary conditions and conservation laws

Comparison between FEM-enhanced neural network and PINN

This section compares the introduced FEM-NN and the state-of-the-art PINNs to demonstrate the advantages of the introduced method.

As mentioned in "Introduction" section, one major limitation of the PINNs is that it makes the problem more ill-conditioned. Whereas the introduced algorithm directly uses the matrices after the application of boundary conditions and hence preserves the well-condition. Consider a one dimensional Poisson equation given below

$$\begin{aligned} &\frac{\partial^2 u}{\partial x^2} = 2 \\ &u(x=-1) = -2, \quad T(x=1) = -2 \end{aligned} \tag{25}$$

Two networks, one with PINN and other with FEM-NN, were trained to solve the Equation system 25. First network is deep copied to make the second one so that both are initialized with same random weights and biases. The Fig. 3 shows the prediction from both models for two cases. In the first case, the prediction after training for 1900 epochs is considered. In can be observed that the FEM-NN has already converged to the actual solution, but PINN still needs to. This shows that the FEM-NN able to learn the exact physics along with boundary conditions quicker than PINNs. However, it should be noted that PINN also converge to the exact solution after 2500 epochs.

Results and discussion

Steady-state convection diffusion problem

This section discusses the application of FEM-enhanced neural network on the one dimensional, convection-diffusion equation. The equation describes physical phenomena where particles, energy, or other physical quantities are transferred inside a physical system due to two processes; convection and diffusion. In this example, we consider the temperature as the physical quantity.

A one-dimensional convection diffusion equation for temperature takes the form

$$\begin{aligned} &\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = k\frac{\partial^2 T}{\partial x^2} + S\,, \\ &T(x=0) = T_1, \quad T(x=1) = T_2 \end{aligned} \tag{26}$$

where T is the temperature, u the convection velocity, k the thermal diffusion coefficient and $S(x)$ the heat source. The term $\frac{\partial T}{\partial t} = 0$ for steady-state problems. The 2^{nd} order

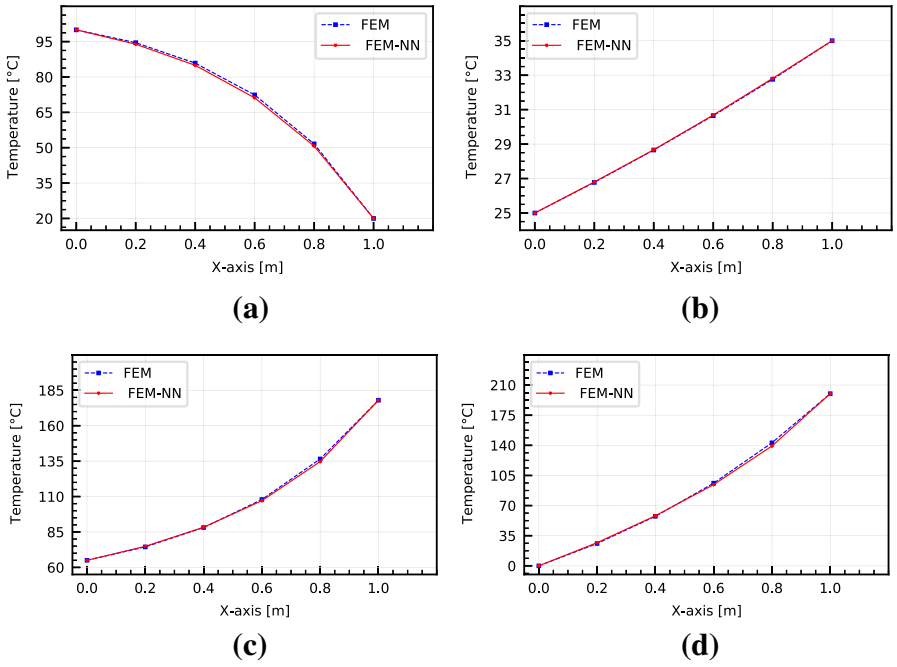


Fig. 4 Distribution of temperature along x-axis for the steady state convection diffusion example for cases **a** $T1 = 100, T2 = 20, k = 10, u = 20, S = 100$, **b** $T1 = 25, T2 = 35, k = 10, u = 3, S = 1$, **c** $T1 = 65, T2 = 178, k = 6, u = 11$ and **d** $T1 = 0, T2 = 200, k = 10, u = 30$

ODE is supported by two boundary conditions (BCs) provided at the two ends of the 1D domain, given as T_1 and T_2. We are concerned with the temperature distribution along the x-axis for the given boundary conditions, velocity, diffusion coefficient and source.

The model is trained using boundary conditions T_1 and T_2, source S, thermal diffusivity k and convection velocity u as the input parameters to obtain the nodal temperature values as output. The parameters for the neural network $T(x)$ are learned by minimizing the FEM-based loss. A standard Adam optimizer [49] is applied to minimize the following loss function,

$$\delta = \frac{1}{N}\sum_{i=i}^{N} ||K_i T_i - F_i||_2^2 \tag{27}$$

where K is the stiffness matrix for the given input, F the force vector and T the output from the neural network and N is the total number of samples used for training. Equation 27 guarantees that the model learns and preserves the underlying physics rather than mere data.

The output of the model for different values of u, k and S under the boundary conditions T_1 and T_2 is shown in Fig. 4. The figure also includes the standard FEM result for comparison. It can be observed that the model learns the physics, and predictions match the results obtained using FEM. Figure 4 has four sub-figures, which show the prediction for different combinations of k, u and S. In every combination of k, u, S, $T1$ and $T2$, the network was able to predict a temperature distribution that closely matches the FEM result. The average absolute error between FEM results and FEM-NN results are 0.616, 0.022, 0.352, and 0.642 degrees for Figs. 4a, b, c and d respectively.

Figure 4a and b have a constant heat source on every node. Figure 4c and d have different values for the heat source on each node. They also use random values for other parameters

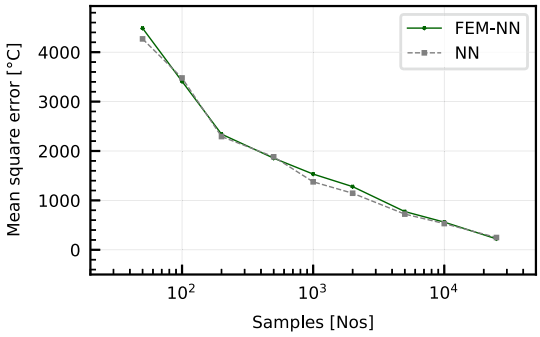


Fig. 5 Comparison of error of prediction between FEM-NN and conventional neural network

too. The parameters used are, $T_1 = 65$, $T_2 = 178$, $S = [5, 2, 3, 4, 5, 1]$, $u = 11$, $k = 6$ for Fig. 4c. Similarly, the parameters for the Fig. 4d are $T_1 = 0$, $T_2 = 200$, $S = [5, 2, 3, 4, 5, 1]$, $k = 1$, and $u = 1$. It can be observed that the model generalizes quite well in predicting the distribution of temperature.

This algorithm described in "Finite element method-enhanced neural network for forward problems" section does not use target values since it uses the matrix and vector from FEM for loss calculation. Hence, it does not fall under supervised learning and can be regarded as a semi-supervised learning approach. A comparison of the accuracy of prediction for FEM-NN and conventional neural networks is presented in Fig. 5. The accuracy is demonstrated using the mean squared error between the prediction and the actual solution of 50,000 samples. The actual solution of 50, 000 samples is created using a standard FEM code. It can be observed that the error is less than or similar to that of the conventional neural network, whereas the time taken for conventional neural network training, including data creation, is 3.4 times more than the introduced FEM-NN.

23-member truss

In this example, we consider a 23-member, simply-supported truss structure example taken from [50]. A truss is governed by the equation

$$\frac{d}{dx}\left[AE\frac{du(x)}{dx}\right] = 0 \tag{28}$$

Where, A, E are the cross sectional area and modulus of elasticity of the material and u the displacement. The equilibirum of forces is given by

$$AE\frac{du(x)}{dx} = T \tag{29}$$

Here T is the axial force on the truss. A system like Fig. 6 is solved by assembling the discretized form of each member to form the complete system of equations. The geometrical dimensions of the truss structure are given in Fig. 6. Young's modulus of the horizontal bars is given by E_1 and for vertical bars by E2. The horizontal bars have a cross-sectional area of A_1 and vertical bars have A_2. The truss is loaded by vertical forces $P1 - P6$. All of these 10 variables are taken as an input for the neural network to predict the 39 nodal displacements for the 13 nodes of the truss. The mean and standard deviation of the training sample for each variable used as input is given in Table 1.

The vertical displacement of the nodes using an FEM-NN surrogate model is plotted in Fig. 7. The example shown took the inputs $E_1 = 2.1 \times 10^{11}$, $E_2 = 2.32 \times 10^{11}$, $A_1 =$

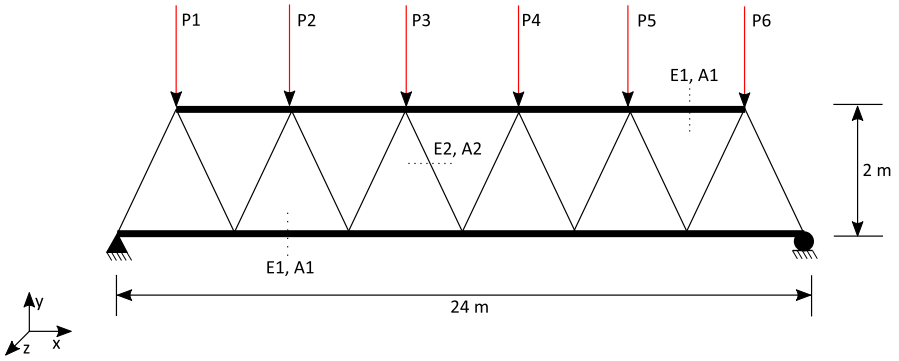


Fig. 6 23-member truss with load

Table 1 Input parameters for the 23-bar truss problem

Input variable	Distribution	Mean	Standard deviation
Horizontal cross-section area $A_h(m^2)$	Normal	1.0×10^{-3}	1.0×10^{-4}
Vertical cross-section area $A_v(m^2)$	Normal	2.0×10^{-3}	2.0×10^{-4}
Horizontal Young's modulus $E_h(Pa)$	Normal	2.1×10^{11}	2.1×10^{10}
Vertical Young's modulus $E_v(Pa)$	Normal	2.1×10^{11}	2.1×10^{10}
Vertical forces $P1 - P6(N)$	Normal	-5.0×10^5	5.0×10^4

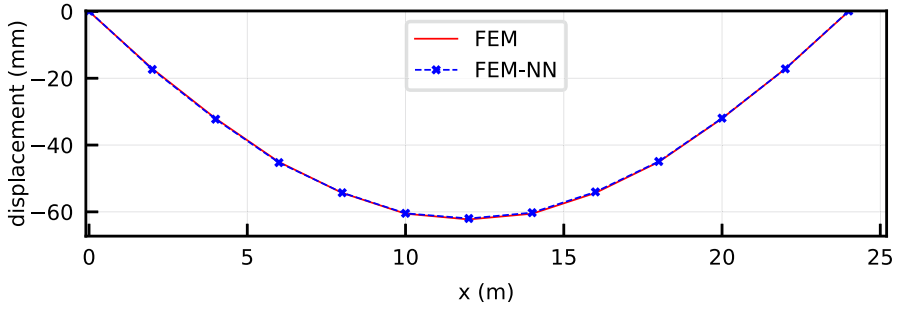


Fig. 7 Vertical deflection of the truss at lower horizontal section

9.2×10^{-4}, $A_2 = 1.89 \times 10^{-3}$, $P_1 = -5.2 \times 10^4$, $P_2 = -5.2 \times 10^4$, $P_3 = -5.4 \times 10^4$, $P_4 = -3.6 \times 10^4$, $P_5 = -6.5 \times 10^4$ and $P_6 = -4.4 \times 10^4$. The displacement is compared with a reference solution calculated using standard FEM. It can be observed that the surrogate model predicts the displacements accurately. The mean error in prediction is as low as $1e^{-4}$.

Applications

Uncertainty quantification of vibration due to wind load on high-rise building

It is important to identify, characterize and calculate the uncertainties for the results of analyses of complex systems. Uncertainty analyses are mandatory in many regulatory standards and guidelines. Oberkampf et al. [51] state that "realistic, modeling and simulation of complex systems must include the non-deterministic features of the system and environment".

A sampling-based uncertainty analysis with Monte Carlo approaches is widely used in the characterization and quantification of uncertainty [52]. Monte Carlo-based uncertainty analyses can now be found in virtually all engineering fields [53]. These analyses lead

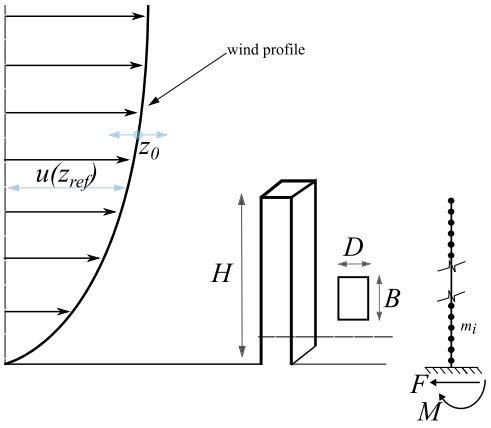


Fig. 8 Wind load on a high-rise building

to final results expressed as a complementary cumulative distribution function (CDF). A large number of simulations, each with different input values sampled from their respective distributions, are run for a Monte Carlo-based uncertainty analysis. The results from simulations are used to obtain probability distributions of targeted outcomes. Hence, the Monte Carlo is inherently computationally expensive. Two methods are mainly used to reduce the number of simulation runs. The first is improving the efficiency of the sampling strategy while attaining the desired accuracy with a minimum number of simulation runs. The second is stopping the analysis once suitably accurate results have been achieved. But it is also possible to use surrogate models with less computational time instead of the original simulation model. A trained neural network can be used as a surrogate model. In this way, the overall cost of a Monte Carlo uncertainty analysis can be reduced. One major drawback of using surrogate models is their reduced accuracy. Since conventional neural networks behave like a black box, we can not quantify the accuracy of the prediction.

FEM-NN can be used as an alternative to overcome such challenges. The integral nature of using FEM and a neural network with residual-based training enables us to obtain the residual of each prediction. In this way, some of the predictions which are not accurate enough can be solved further using any of the conventional iterative methods, using the prediction from FEM NN as the initial value. Wind load on a high-rise building is used to demonstrate the use of FEM-NN for uncertainty quantification analysis.

The wind load on high-rise buildings is studied during the design of buildings as it can cause structural damage to the buildings. The uncertainty in wind climate and uncertain terrain also affect the long term performance of the structure. Hence, the uncertainty in these parameters and its effects has to be studied. The effect of uncertainty in the wind load on the horizontal displacement at the top of the building is studied in this example. The CAARC (Commonwealth Advisory Aeronautical Council) [54] building geometry is used for the study. It is one of the main models used for calibrating experimental techniques in wind tunnels for studying building aerodynamics. Since its introduction by Wardlaw et al. in [55], the CAARC model has been extensively used in wind tunnel experiments. The popularity resulted in large number of papers using numerical simulation based wind analysis on CAARC model [54,56,57].

Table 2 Details of the building—geometry and structural

Parameters	Values
Height (H)	180 m
Width (W)	45 m
Length (D)	30 m
Frequency (f)	0.2 Hz
Density (ρ)	160 kg/m^3
Damping ratio (ζ)	0.01

Table 3 Details of uncertain wind parameters

Uncertain parameters	Distribution	Values
Mean wind velocity $u(z_{ref})$	Weibull	Mean = 40 m/s
		Shape parameter = 2
Roughness length, z_0	Uniform	[0.1, 0.7]

The parameters of the building height (H), width (B) and length (D), air density ρ, natural frequency of the building (f) and damping coefficient (ζ) are shown in Fig. 8 and given in Table 2. The high-rise building is modeled using the Euler-Bernoulli beam model as described by the following equations

$$\frac{\partial^2}{\partial x^2}\left(EI\frac{\partial^2 u_z(x)}{\partial x^2}\right) = f(z) \tag{30}$$

$$\frac{\partial^2}{\partial x^2}\left(EI\frac{\partial^2 u_y(x)}{\partial x^2}\right) = f(y) \tag{31}$$

where u_y and u_z represents the deflection of the building and f represents the applied load. E represents the Young's modulus and I is the second moment of area. The corresponding surrogate model is created using FEM-NN. The wind profile and surface roughness are used as an input to the model. The mean and standard deviation of the training sample for each variable used as an input is given in Table 3. A wind-load on the building is calculated using the mean wind velocity $u(z_{ref})$ and roughness length z_0. The static wind load at each height of the building is calculated as

$$F_d(x) = \frac{\rho V(x)^2 A C_d}{2} \tag{32}$$

where, ρ is the air density, $V(x)$ is the velocity at height x, A is the reference area and cd is the coefficient of drag for the cross section.

The trained model is used for quantifying the uncertainty associated with the effects of wind load on a high-rise building. The trained model is used to run a large number of simulations to perform the uncertainty quantification using the Monte Carlo method.

Uncertainty in calculating the top displacement of the building is plotted for different cases in Fig. 9. Figure 9a shows the probability distribution of the top displacement of the building. Figure 9b shows the cumulative distribution function. Both figures also contain the results obtained using FEM-based simulations. It can be observed that the FEM-NN is able to reproduce the results obtained using FEM with reasonable accuracy. In this case, the input parameters were taken from the same distribution as that used for training. Figure 9c and d are the results of an analysis with different distribution of input parameters than that used for training. The statistical quantities of the analysis for both the cases are given in Table 4. The FEM-NN was able to produce accurate results in this

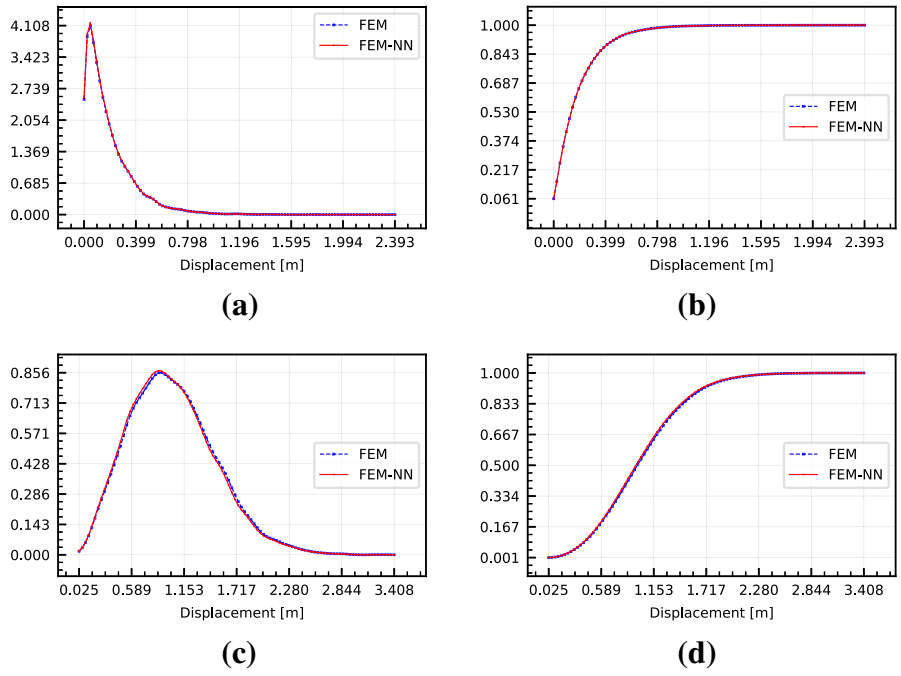


Fig. 9 PDF and CDF in trained region in **a** and **b** and untrained region in **c** and **d**

Table 4 Statistical quantities of Monte-Carlo analysis

	Trained region				Untrained region			
	Mean	Standard deviation	Skewness	Kurtosis	Mean	Standard deviation	Skewness	Kurtosis
FEM	0.183	0.199	2.679	12.278	1.016	0.470	0.667	0.717
FEM-NN	0.181	0.197	2.706	12.555	1.004	0.468	0.693	0.790

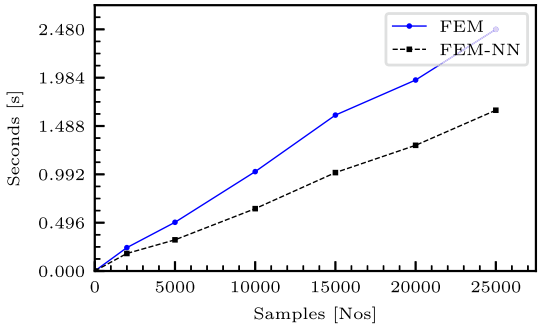


Fig. 10 Cost for Monte Carlo simulations in seconds

case too. Hence, the FEM-NN can replace the conventionally expensive FEM model for analyses such as Monte Carlo uncertainty quantification.

The cost of running a Monte Carlo based uncertainty quantification using FEM and FEM-NN is plotted against different numbers of samples in Fig. 10. It can be seen that the FEM-NN based analysis reduces the cost significantly. However, it should be noted that the training time or hyperparameter tuning time of the neural network is not taken into account for this.

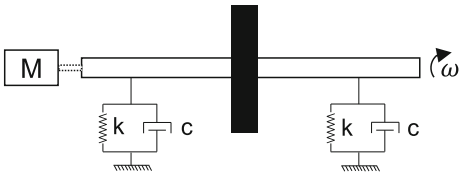


Fig. 11 Schematic diagram of a flexible rotor-bearings system supported on bearings

Fluid bearing stiffness identification

Rotor dynamic systems are of engineering interest because of their use in a wide range of applications such as power plants, engines, etc. The accurate prediction of their dynamic behavior is vital for uninterrupted operation and safety. The estimation of bearing coefficients has been a primary barrier for predicting or simulating the dynamic behavior of such systems. For example, the dynamic coefficients of fluid bearings vary with the rotating speed of the machine. Fluid bearings are bearings in which the load is supported by a thin layer of fluid. In hydrostatic bearings, the fluid is pressurized using an external pump, whereas, in hydrodynamic bearings, the fluid is pressurized by the speed of the rotating shaft/journal. A detailed review of different parameter identification methods for rotor dynamic systems can be found in [58].

The introduced FEM-NN can be used as an alternative to the different methods that exist to identify bearing parameters. In this example, the stiffness coefficients of fluid bearings for the rotor dynamic system are modeled as a neural network taking rotational speed as input.

$$K_b = f(\omega) \tag{33}$$

where K_b represents the dynamic stiffness coefficients of the fluid bearing and ω represents the rotational speed of the shaft. The function f represents the neural network mapping rotational speed to stiffness coefficients. The complete system of equations describing the motion of a rotor-dynamic system in frequency domain can be described using the Eq. (34)

$$\left[-\omega^2 M + j\omega(G\omega + C) + K\right] q = F \tag{34}$$

where M, G, C and K are mass, gyroscopic, damping and stiffness matrices for the complete system. F and q are the force and response of the system in the frequency domain. When the stiffness of a fluid bearing is modeled as Eq. (33), the system stiffness matrix K can be written as

$$K = K_r + K_b \tag{35}$$

where K_r is the stiffness matrix of the system excluding bearings.

Inverse problem algorithms, similar to those for forward problems, also use the residual of the equation as the loss function. In this case the residual r is

$$r = \left[-\omega^2 M + j\omega(G\omega + C) + K\right] q - F \tag{36}$$

Minimizing this residual by optimizing the neural network parameters results in a neural network which predicts the bearing stiffness for a given speed.

Figure 12 compares the predicted bearing stiffness against the actual stiffness. The actual stiffness values are calculated by modelling a fluid bearing with the help of an open source

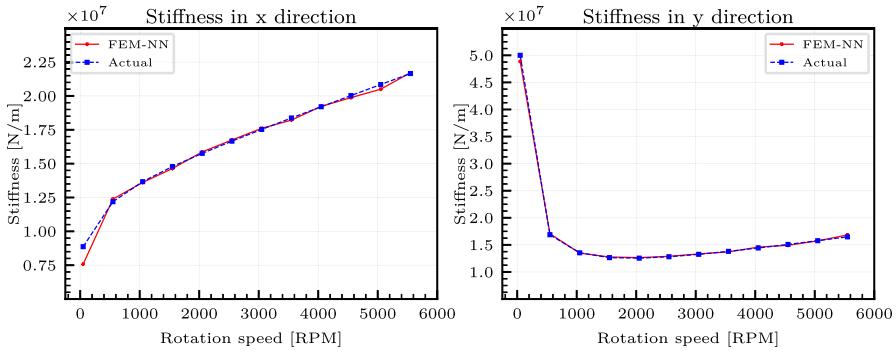


Fig. 12 Bearing stiffness vs Speed for x-direction (left) and y-direction (right)

software called ross rotordnamic [59]. It can be observed that the stiffness variation against speed of rotation is captured precisely by the neural network. The stiffness in the $x-$ and $y-$ direction, which vary differently, are learned by a single training process using FEM-NN algorithm for inverse problems proposed in section Finite element method-enhanced neural network for inverse problems.

Conclusion and outlook

We introduced the Finite Element Method-enhanced neural networks (FEM-NN) for forward and inverse PDE problems. The method combines the discretized form of the PDE and loss function of the neural network to result in physics-informed surrogate models. State-of-the-art algorithms in informed machine learning follow the idea of two loss terms, using the strong form of the PDE or its variants as one loss term and weakly enforcing boundary conditions with the help of a mean squared error as the other loss term. In contrast to the trend in informed machine learning, we use a single loss term that encapsulates the PDE along with boundary conditions in the discretized form. This way, the well-posedness of the problem is not compromised. Compared to conventional supervised learning, the computational cost is reduced drastically by using inexpensive FEM simulation (system matrices) data for the training. In contrast, traditional methods use the final solution, which is obtained after solving the matrix system using linear solvers. The method also opens a way to use conventional FEM frameworks and neural networks to make inexpensive surrogate models. Additionally, developments in the FEM and AI communities could further develop the hybrid approach.

Although the algorithm is simple and has the potential to change simulation methods drastically, it demands more research. The selection of suitable neural network architecture is empirical in nature. This makes the application of the method to bigger problems time-consuming in terms of architecture search. The present method developed for linear problems needs to be investigated for non-linear cases. The constant parameterized input nature of neural networks restricts using a trained neural network on other geometries. Even though concepts like transfer learning offer flexibility, they require further investigation before they can be combined with the proposed algorithm. The same problem can be solved from the numerical community point of view by employing a polycube representation of the geometry. However, such methods are yet to be explored. It is also possible to combine the introduced method with state-of-the-art PINNs. If the predictions using PINNs are performed on a numerical grid, we can use the introduced residual-based loss

function as a new loss term for PINNs. This way, the ill-posedness experienced by PINNs can be reduced.

Acknowledgements
The authors are grateful to Siemens for sponsoring this project

Author contributions
RE, BO, KUB and RW conceptualized the main algorithm presented in this work. RE, AG and MK developed the methodology for forward and inverse problems. AK implemented the uncertainty quantification application. RE and MK implemented the parameter identification of fluid bearing. All authors approved the final submitted draft.

Funding Information
Open Access funding enabled and organized by Projekt DEAL.

Availability of data and materials
The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request. They are generated using open-source softwares Kratos [60] and Ross [59]. A sample case is available at https://github.com/rishithellathmeethal/fem_nn.git.

Declarations

Consent for publication
All authors approved the final submitted draft

Competing interests
R.E., B.O., and M.K. are employed at company Siemens AG.

Received: 23 June 2022 Accepted: 8 March 2023
Published online: 13 May 2023

References

1. Russell S, Norvig P. Artificial intelligence—a modern approach. Prentice Hall series in artificial intelligence. 2nd ed. Prentice Hall; 2003.
2. Szeliski R. Computer vision: algorithms and applications. Berlin: Springer; 2010.
3. Jurafsky D. Speech & language processing. India: Pearson Education India; 2000.
4. Burbidge R, Trotter M, Buxton B, Holden S. Drug design by machine learning: support vector machines for pharmaceutical data analysis. Comput Chem. 2001;26(1):5–14.
5. Lavecchia A. Machine-learning approaches in drug discovery: methods and applications. Drug Discov Today. 2015;20(3):318–31.
6. Libbrecht MW, Noble WS. Machine learning applications in genetics and genomics. Nat Rev Genet. 2015;16(6):321–32.
7. Funge JD. Artificial intelligence for computer games: an introduction. CRC Press; 2004.
8. Grzeszczuk R, Terzopoulos D, Hinton G. Neuroanimator: fast neural network emulation and control of physics-based models. In: Proceedings of the 25th annual conference on computer graphics and interactive techniques; 1998. p. 9–20.
9. Kawato M, Uno Y, Isobe M, Suzuki R. Hierarchical neural network model for voluntary movement with application to robotics. IEEE Control Syst Mag. 1988;8(2):8–15.
10. Lu D, Weng Q. A survey of image classification methods and techniques for improving classification performance. Int J Remote Sens. 2007;28(5):823–70.
11. Papageorgiou CP, Oren M, Poggio T. A general framework for object detection. In: Sixth international conference on computer vision (IEEE Cat. No. 98CH36271), IEEE; 1998. p. 555–562.
12. Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. p. 3431–3440.
13. Nareyek A. Ai in computer games. Queue. 2004;1(10):58–65.
14. Fairclough C, Fagan M, Mac Namee B, Cunningham P. Research directions for AI in computer games. Technical report, Trinity College Dublin, Department of Computer Science; 2001.
15. Yannakakis GN. AI in computer games : generating interesting interactive opponents by the use of evolutionary computation. PhD thesis, University of Edinburgh, UK; 2005.
16. Tompson J, Schlachter K, Sprechmann P, Perlin K. Accelerating eulerian fluid simulation with convolutional networks. CoRR **abs/1607.03597**; 2016. arXiv:1607.03597
17. Hecht-Nielsen R. Theory of the backpropagation neural network. In: Neural networks for perception. Elsevier; 1992. p. 65–93.
18. Adeli H, Yeh C. Perceptron learning in engineering design. Comput-Aided Civil Infrastruct Eng. 1989;4(4):247–56.
19. Thuerey N, Weißenow K, Prantl L, Hu X. Deep learning methods for reynolds-averaged navier-stokes simulations of airfoil flows. AIAA J. 2020;58(1):25–36.
20. Pfaff T, Fortunato M, Sanchez-Gonzalez A, Battaglia PW. Learning mesh-based simulation with graph networks; 2020. arXiv preprint http://arxiv.org/abs/2010.03409
21. Guo X, Li W, Iorio F. Convolutional neural networks for steady flow approximation. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining; 2016. p. 481–490
22. Zhang R, Liu Y, Sun H. Physics-informed multi-lstm networks for metamodeling of nonlinear structures. Comput Methods Appl Mech Eng. 2020;369:113226.

23. von Rueden L, Mayer S, Beckh K, Georgiev B, Giesselbach S, Heese R, Kirsch B, Pfrommer J, Pick A, Ramamurthy R, et al. Informed machine learning–a taxonomy and survey of integrating knowledge into learning systems; 2019. arXiv:1903.12394
24. Chang M-W, Ratinov L, Roth D. Guiding semi-supervision with constraint-driven learning. In: Proceedings of the 45th annual meeting of the association of computational linguistics; 2007. p. 280–287
25. Hu Z, Yang Z, Salakhutdinov R, Xing E. Deep neural networks with massive learned knowledge. In: Proceedings of the 2016 conference on empirical methods in natural language processing; 2016. p. 1670–1679.
26. Stewart R, Ermon S. Label-free supervision of neural networks with physics and domain knowledge. In: Thirty-first AAAI conference on artificial intelligence; 2017.
27. Raissi M, Perdikaris P, Karniadakis GE. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations; 2017. arXiv preprint http://arxiv.org/abs/1711.10561
28. Griewank A, et al. On automatic differentiation. Math Program Recent Dev Appl. 1989;6(6):83–107.
29. Cai S, Mao Z, Wang Z, Yin M, Karniadakis GE. Physics-informed neural networks (pinns) for fluid mechanics: a review. Acta Mech Sinica. 2022;1–12.
30. Cai S, Wang Z, Wang S, Perdikaris P, Karniadakis GE. Physics-informed neural networks for heat transfer problems. J Heat Transfer. 2021;143(6).
31. Misyris GS, Venzke A, Chatzivasileiadis S. Physics-informed neural networks for power systems. In: 2020 IEEE power & energy society general meeting (PESGM), IEEE; 2020. p. 1–5.
32. Lu L, Meng X, Mao Z, Karniadakis GE. Deepxde: a deep learning library for solving differential equations. SIAM Review. 2021;63(1):208–28.
33. Pang G, Lu L, Karniadakis GE. fpinns: fractional physics-informed neural networks. SIAM J Sci Comput. 2019;41(4):2603–26.
34. Zhang D, Lu L, Guo L, Karniadakis GE. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. J Comput Phys. 2019;397:108850.
35. Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. Nature Reviews. Physics. 2021;3(6):422–40.
36. Wang S, Yu X, Perdikaris P. When and why pinns fail to train: a neural tangent kernel perspective. J Comput Phys. 2022;449:110768.
37. Krishnapriyan A, Gholami A, Zhe S, Kirby R, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. Adv Neural Inf Process Syst. 2021;34:26548–60.
38. Zienkiewicz OC, Taylor RL. The finite element method, the basis. The finite element method. Wiley; 2000. https://books.google.de/books?id=Huc5tAEACAAJ
39. Versteeg HK, Malalasekera W. An introduction to computational fluid dynamics: the finite volume method. Pearson education; 2007.
40. Oñate E, Owen R. Particle-based methods: fundamentals and applications. Computational methods in applied sciences. Netherlands: Springer; 2011.
41. Kollmannsberger S. The finite cell method: towards engineering applications. Technische Universität München, Munich; 2019. https://books.google.de/books?id=oJtdzQEACAAJ
42. Ypma TJ. Historical development of the Newton-Raphson method. SIAM Rev. 1995;37(4):531–51.
43. Niroomandi S, Alfaro I, González D, Cueto E, Chinesta F. Real-time simulation of surgery by reduced-order modeling and x-fem techniques. Int J Numer Methods Biomed Eng. 2012;28(5):574–88. https://doi.org/10.1002/cnm.1491.
44. Keiper W, Milde A, Volkwein S. Reduced-order modeling (ROM) for simulation and optimization: powerful algorithms as key enablers for scientific computing. Berlin: Springer; 2018.
45. Schilders WH, van der Vorst HA, Rommes J. Model order reduction: theory, research aspects and applications. Mathematics in industry. Heidelberg: Springer; 2008.
46. Antoulas AC. Approximation of large-scale dynamical systems. Society for Industrial and Applied Mathematics; 2005. https://doi.org/10.1137/1.9780898718713. https://epubs.siam.org/doi/pdf/10.1137/1.9780898718713. https://epubs.siam.org/doi/abs/10.1137/1.9780898718713
47. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker PA, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X. Tensorflow: a system for large-scale machine learning. In: OSDI. USENIX Association; 2016. p. 265–283.
48. Ketkar N. Introduction to pytorch. In: Deep learning with Python. Springer; 2017. p. 195–208.
49. Kingma DP, Ba J. Adam: a method for stochastic optimization; 2014. arXiv preprint http://arxiv.org/abs/1412.6980
50. Sudret B. Uncertainty propagation and sensitivity analysis in mechanical models-contributions to structural reliability and stochastic spectral methods. Habilitationa diriger des recherches, Université Blaise Pascal, Clermont-Ferrand, France. 2007;147:53.
51. Oberkampf WL, DeLand SM, Rutherford BM, Diegert KV, Alvin KF. Error and uncertainty in modeling and simulation. Reliab Eng Syst Saf. 2002;75(3):333–57.
52. Helton JC. Treatment of uncertainty in performance assessments for complex systems. Risk Anal. 1994;14(4):483–511.
53. Janssen H. Monte–Carlo based uncertainty analysis: sampling efficiency and sampling convergence. Reliab Eng Syst Saf. 2013;109:123–32.
54. Braun AL, Awruch AM. Aerodynamic and aeroelastic analyses on the CAARC standard tall building model using numerical simulation. Comput Struct. 2009;87(9–10):564–81.
55. Wardlaw R, Moss G. A standard tall building model for the comparison of simulated natural winds in wind tunnels. CAARC, CC 662m Tech. 1970. p. 25.
56. Huang S, Li QS, Xu S. Numerical evaluation of wind effects on a tall steel building by CFD. J Constr Steel Res. 2007;63(5):612–27.
57. Tosi R, Núñez M, Pons-Prats J, Principe J, Rossi R. On the use of ensemble averaging techniques to accelerate the uncertainty quantification of cfd predictions in wind engineering. J Wind Eng Ind Aerodyn. 2022;228:105105.
58. Lees A. Identification of dynamic bearing parameters: a review. Shock Vib Digest. 2004;36(2):99–124.

59. Timbó R, Martins R, Bachmann G, Rangel F, Mota J, Valério J, Ritto TG. Ross—rotordynamic open source software. J Open Source Softw. 2020;5(48):2120. https://doi.org/10.21105/joss.02120.
60. Dadvand P, Rossi R, Oñate E. An object-oriented environment for developing finite element codes for multi-disciplinary applications. Arch Comput Methods Eng. 2010;17(3):253–97.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.