Physics-informed neural networks approach for the one-dimensional beams

Felipe Pereira dos Santos[1], Lapo Gori[1]

[1]*Dept. of Structural Engineering, Federal University of Minas Gerais*
Avenida Antônio Carlos 6627, 31270901, Belo Horizonte/MG, Brazil
felipereira.santos@hotmail.com, lapo@dees.ufmg.br

Abstract. The physics-informed neural network (PINN) is a machine learning technique where the physics of the problem are embedded into the loss function. The straightforward approach is to define the loss function using the problem governing differential equations and its boundary/initial conditions in a sort of collocation method. In general, the hyper-parameters of neural networks for each problem in hands are defined via a grid search-like procedure, or simply by trial and error. In this paper, the application of PINNs is illustrated for one-dimensional beam problems, and the influence of the weights initialisation on the training is investigated. The code was built using SciANN, a Python package that uses TensorFlow and Keras for scientific computing and physics-informed deep learning employing artificial neural networks.

Keywords: Physics-informed neural networks, Artificial neural networks, Beam problems

1 Introduction

Physics-informed neural networks (PINNs) are artificial neural networks that can be trained with data while respecting physical constraints. This is achieved by formulating the loss function to include the governing equations of the problem and its initial and/or boundary conditions. While the term *physics-informed neural networks* was introduced by Raissi et al. [1], the concept behind the method dates back three decades, as seen in the work of Lagaris et al. [2]. The method builds on insights from Hornik et al. [3] (see also Cybenko [4]), who demonstrated that neural networks are universal approximators. The contemporary approach by Raissi et al. [1] generalizes previous methods using modern computational tools. This foundational work has led to numerous extensions and improvements in the PINN method over the years. Notably, several papers have applied PINNs to Euler-Bernoulli and Timoshenko beam problems, including [5–8].

In this paper, we investigate the impact of weight initialisation in PINNs[1] on the analysis of four common boundary conditions of a Timoshenko beam under a uniformly distributed load. Analytical solutions were used for error computation, and results from the linear finite element method (FEM) are presented for comparison.

2 Physics-informed neural networks

The structure of an *artificial neural network* and its training process are shown in Fig. 1. Its basic components are represented by the input layer, which is responsible for the entry data, the hidden layers, where most operations are performed during the learning task and the output layer, which provides the model predictions. Each layer comprises several neurons that communicate with each other. The weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ are the trainable parameters of a neural network, which means that these quantities change during the training. In this process, the information is transmitted between two neighbouring layers according to eq. (1) as follows:

$$\begin{aligned} \boldsymbol{z}^{\ell} &= \sigma^{\ell}\left(\boldsymbol{W}^{\ell}\boldsymbol{z}^{\ell-1} + \boldsymbol{b}^{\ell}\right), \quad \ell = 1, 2, 3, \ldots, L \\ \boldsymbol{\Theta} &= \{\boldsymbol{W}, \boldsymbol{b}\} \quad \text{all trainable parameters,} \end{aligned} \tag{1}$$

[1]Using the strong form of the problem, that is to say, the *classic* PINN method.

where $\boldsymbol{W}^{\ell}$ are the weights of each one of the connections between two layers, $\boldsymbol{b}^{\ell}$ are the biases, which are values added to the weighted input from the previous layer, $\boldsymbol{z}^{\ell}$ is the output of the layer ℓ and input of the layer $\ell+1$, hence $\boldsymbol{z}^{0} \equiv \boldsymbol{X}$, the input of the neural network, and $\boldsymbol{z}^{L} \equiv \boldsymbol{y}$, the output of the neural network, as illustrated in Fig. 1. In equation eq. (1), the output of the neurons from layer $(\ell-1)$ are used as input data for the neurons in layer ℓ. The function σ^{ℓ} is the activation function that usually inserts non-linearities into the neural network connections, allowing to capture more complex relationships between the inputs and outputs. In order to model a problem with a neural network, it is necessary to first set the neural network, and then to train it. The loss function $\mathcal{L}$ illustrated in Fig. 1 evaluates how well the model predictions perform on the dataset (inputs).

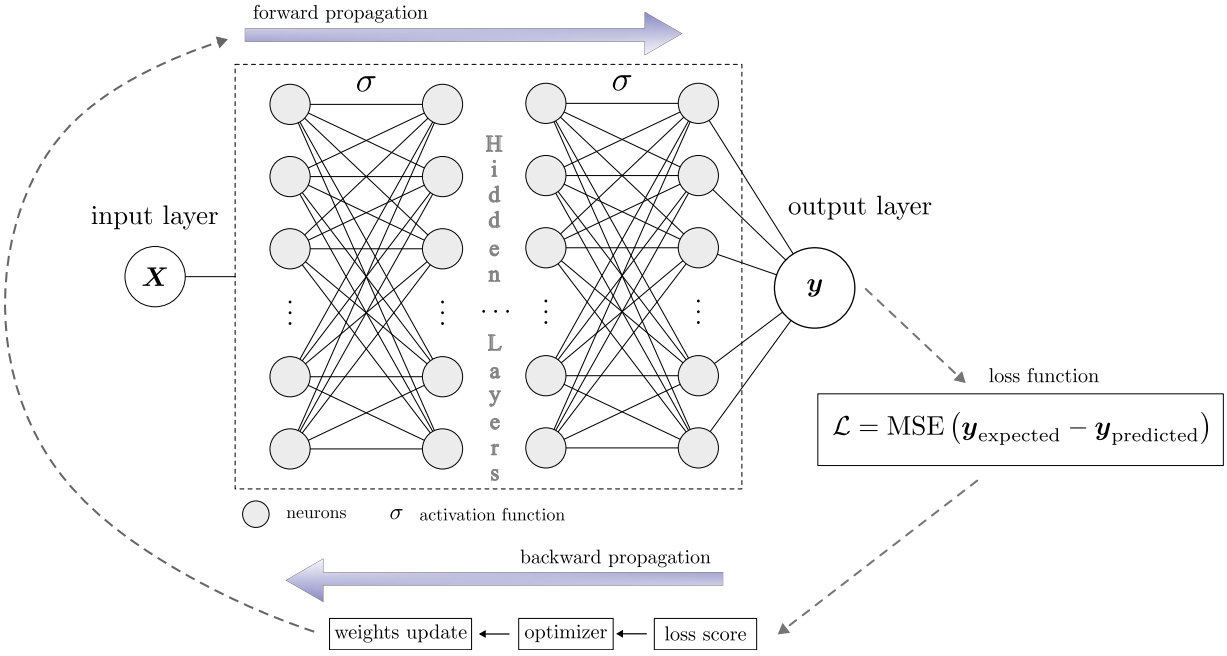


Figure 1. Fully connected neural network

In supervised learning, the input-output pairs form the labelled dataset used to train the neural network. This process involves minimizing the loss function to obtain "optimal" values for the weights and biases. This is typically accomplished using a gradient descent algorithm or similar methods, where the error in the neural network's predictions is evaluated during each iteration. In a *physics-informed neural network* the loss function is enriched to include the physics of the problem.

Suppose our task is to find a solution $v(x)$ for a generic non-linear partial differential equation (PDE). Since neural networks are recognised as universal approximators [3, 9], the output $\boldsymbol{y}$ (see Fig. 1) can be a candidate solution $v(x)$ for the PDE, that is to say, $v(x) \approx \boldsymbol{\mathcal{N}}_v$ is a reasonable assumption to make, where $\boldsymbol{\mathcal{N}}_v$ represents a neural network prediction. The basic idea of PINNs is to approximate the field variables with neural networks, and as already pointed out, construct the loss function to respect the physics of the problem represented by the PDE itself and by its initial and/or boundary conditions. This leads to an enriched loss function of the type:

$$\mathcal{L} = \lambda_{\text{data}}\mathcal{L}_{\text{data}} + \lambda_{\text{PDE}}\mathcal{L}_{\text{PDE}} + \lambda_{\text{BCs}}\mathcal{L}_{\text{BCs}} + \lambda_{\text{ICs}}\mathcal{L}_{\text{ICs}}, \tag{2}$$

where $\mathcal{L}_{\text{data}}$ is the term associated to the labelled dataset as in a regular ANN, that is to say the same loss shown in Fig. 1. In addition to that, $\mathcal{L}_{\text{PDE}}$ is the residual of the partial differential equations, while $\mathcal{L}_{\text{ICs}}$ and $\mathcal{L}_{\text{BCs}}$ are, respectively, the terms associated to the initial and boundary conditions. Each λ is a hyper-parameter that needs to be tuned for training. A common metric used for each term in eq. (2) is the Mean Square Error (MSE), as shown in eq. (3) where each term of the loss function is illustrated.

$$
\begin{aligned}
\mathcal{L}_{\text{data}} &= \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left| u(\boldsymbol{x}^i_{\text{data}}, t^i_{\text{data}}) - y^i_{\text{data}} \right|^2, \\
\mathcal{L}_{\text{BC}} &= \frac{1}{N_{\text{BC}}} \sum_{i=1}^{N_{\text{BC}}} \left| u(\boldsymbol{x}^i_{\text{BC}}, t^i_{\text{BC}}) - y^i_{\text{BC}} \right|^2, \\
\mathcal{L}_{\text{IC}} &= \frac{1}{N_{\text{IC}}} \sum_{i=1}^{N_{\text{IC}}} \left| u(\boldsymbol{x}^i_{\text{IC}}, t^i_{\text{IC}}) - y^i_{\text{IC}} \right|^2, \\
\mathcal{L}_{\text{PDE}} &= \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left| \text{generic PDE} - 0 \right|^2,
\end{aligned}
\tag{3}
$$

where the number of collocation points N is specific for each loss term in eq. (3). In this paper, since all the examined problems are static, with no initial conditions, and data was not used in the training process, the loss function employs only the governing partial differential equations and their boundary conditions. The target problem of this paper is the Timoshenko beam, a problem governed by the following couple differential equations:

$$
\frac{\partial}{\partial x}\left(EI\frac{\partial \theta}{\partial x}\right) + k_s GA\left(\frac{\partial v}{\partial x} - \theta\right) = 0, \tag{4}
$$

$$
\frac{\partial}{\partial x}\left[k_s GA\left(\frac{\partial v}{\partial x} - \theta\right)\right] = -q(x), \tag{5}
$$

where $q(x)$ is the distributed load acting over the beam span, A the area of the cross-section, I the second moment of area of the cross-section, E the Young's modulus, G the shear modulus, and k_s the shear correction factor. Deflections and cross-section rotations are represented by the variables $v(x)$ and $\theta(x)$, respectively. Unlike the Euler-Bernoulli beam, the Timoshenko beam model accounts for shear deformation effects in addition to bending deformation.

3 Modelling aspects

In this section, the settings used to performed the training of the PINN models are introduced. A neural network with 3 hidden layers with 20 neurons each was adopted to approximate each field variable. While it is possible to use only one neural network for the model to represent all field variables, in this paper, the deflection and cross-section rotations are approximated by separated neural networks as shown in eq. (6):

$$
v(x) \approx \boldsymbol{\mathcal{N}}_u(x, \boldsymbol{\Theta}), \quad \theta(x) \approx \boldsymbol{\mathcal{N}}_\theta(x, \boldsymbol{\Theta}) \tag{6}
$$

Due to its success on many problems, the optimization algorithm Adam (Adaptive Moment Estimation) was employed [10] with a batch size of 32. An initial learning rate of 10^{-3} was settled, and it is automatic reduced during training by default, when the learning stagnates. The standard number of epochs was 400. Among many options of activation functions, such as, Sigmoid, ReLU, Gaussian, Softplus, in this work, the hyperbolic tangent $\tanh(x)$ was the choice, since this is a common choice for PINNs simulations in engineering problems, and it possesses continuous derivatives. Finally, the mean square error was chosen as the loss function metric, and 2000 was the standard number of training points. To generate the input data, half of the 2000 points were randomly distributed along the beam domain to be the inputs of the differential equations. The other half was divided equally among the boundary conditions, 500 points for $x = 0$, and 500 points for $x = L$. The target data associated with all these points are simply zeros. It is worth noticing that this was a particular choice of points distribution, and it can influence the performance of the model training. At last, different from the training set, the test set was generated with equally distributed points.

In order to facilitate the construction and understanding of the algorithm to solve the beam models, the open source Python package SciANN [11] was used to ease the code structuring. This library is build with Tensorflow [12] and Keras [13], hence it inherits their functionalities. SciANN covers several engineering applications, such as

model fitting, solution of ordinary and partial differential equations, and model inversion (parameter identification) [11].

The initialisation of the weights and biases was performed using several initialisers to analyse its influence, if any, on the error results of deflections and rotations. The weights of the initialisers **Xavier**, **LeCun**, **random**, and **He** are obtained by a normal distribution, using the Gaussian distribution or a uniform distribution, where every value inside the range has an equal probability of occurring. The **truncated normal** initialiser draws samples from a normal distribution but truncates values with more than two standard deviations from the mean, re-sampling them instead. The **variance scaling** initialiser is designed to adapt to the shape of its input tensors. Finally, the **orthogonal** initialiser is generated using an orthogonal matrix. All initialisers are bounded by a range of values [-limit, limit] typically defined based on the number of inputs (F_{in}) and outputs (F_{out}) of the layer. While it is possible to set different means and standard deviations for the initialisation, in this study, the weights were initialised using TensorFlow with its default settings. In Fig. 2, the probability density of the weights for a layer with 20 inputs and 20 outputs is illustrated, while for clarity, the probability density of the weights is shown for a layer with 1000 inputs and 1000 outputs in Fig. 3.

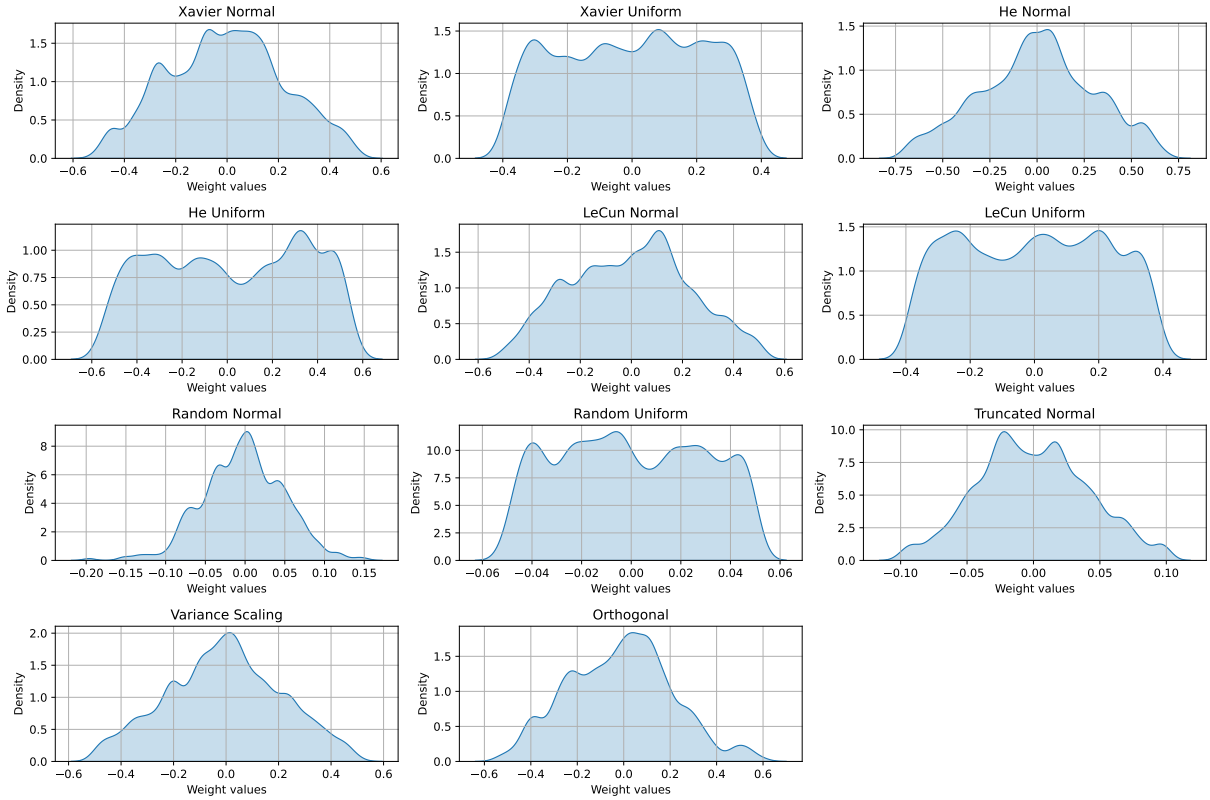


Figure 2. Weights considering a layer with 20 inputs and 20 outputs

4 Numerical simulations

This section illustrates numerical results obtained with the PINN approach applied to the Timoshenko beam. All simulations were performed considering a linear elastic material, with a Young's modulus $E = 100$ N/m^2 and Poisson's ratio $\nu = 0.3$. A rectangular cross-section with $b = 0.2$ m and $h = 0.5$ m was adopted, with a form factor $k_s = 5/6$ and a beam length $L = 2$ m. A number of 2000 collocation points were used for training, while the predictions were performed with 200 uniformly distributed points along the beam span. The above parameters are assumed for all beams, except when specified otherwise. A prismatic beam under an uniformly distributed load $q(x) = -1$ N/m over its span is investigated considering four boundary conditions, as shown in Fig. 4: (i) simply-supported (pinned-pinned), (ii) cantilever (fixed-free), (iii) doubly-fixed (fixed-fixed), and (iv) fixed-pinned.

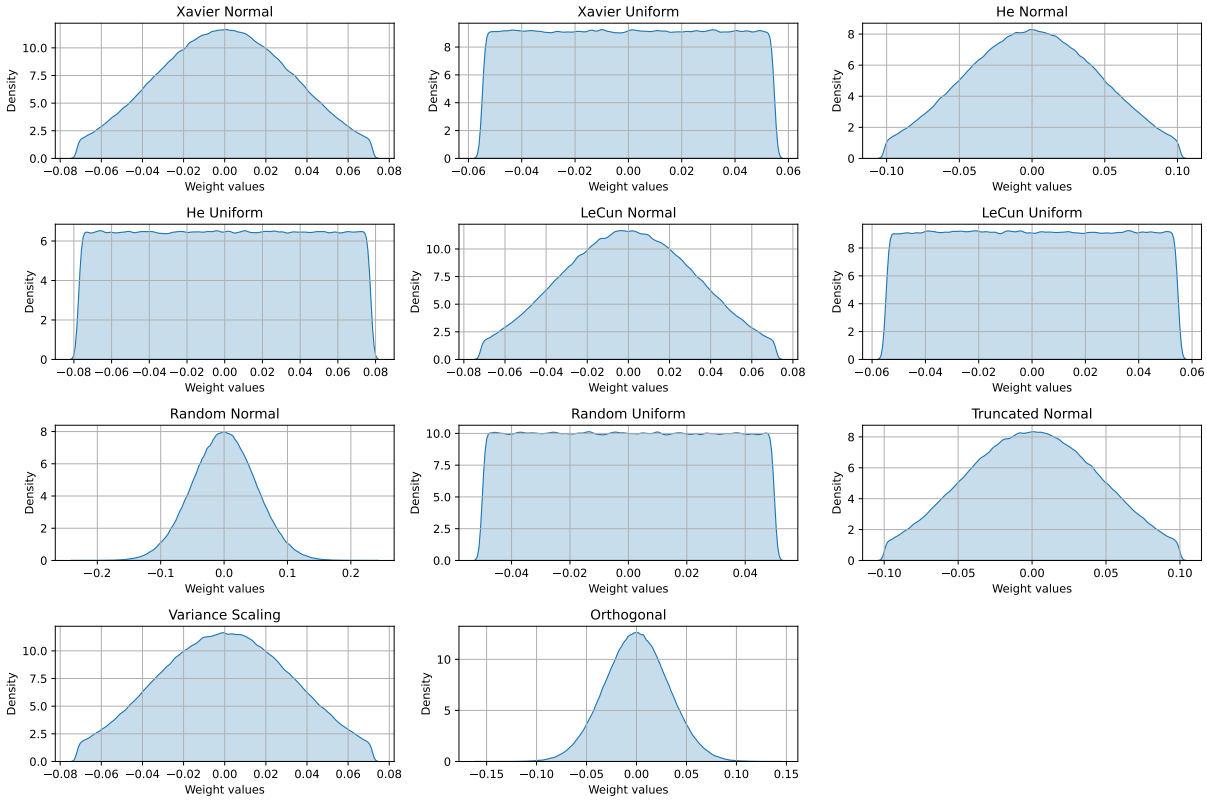


Figure 3. Weights considering a layer with 1000 inputs and 1000 outputs

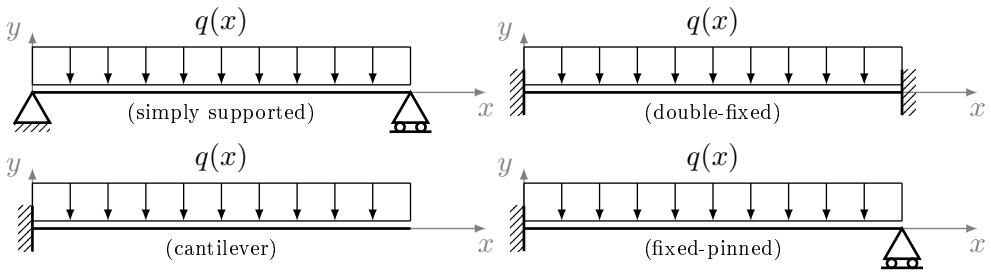


Figure 4. Classic beam boundary conditions

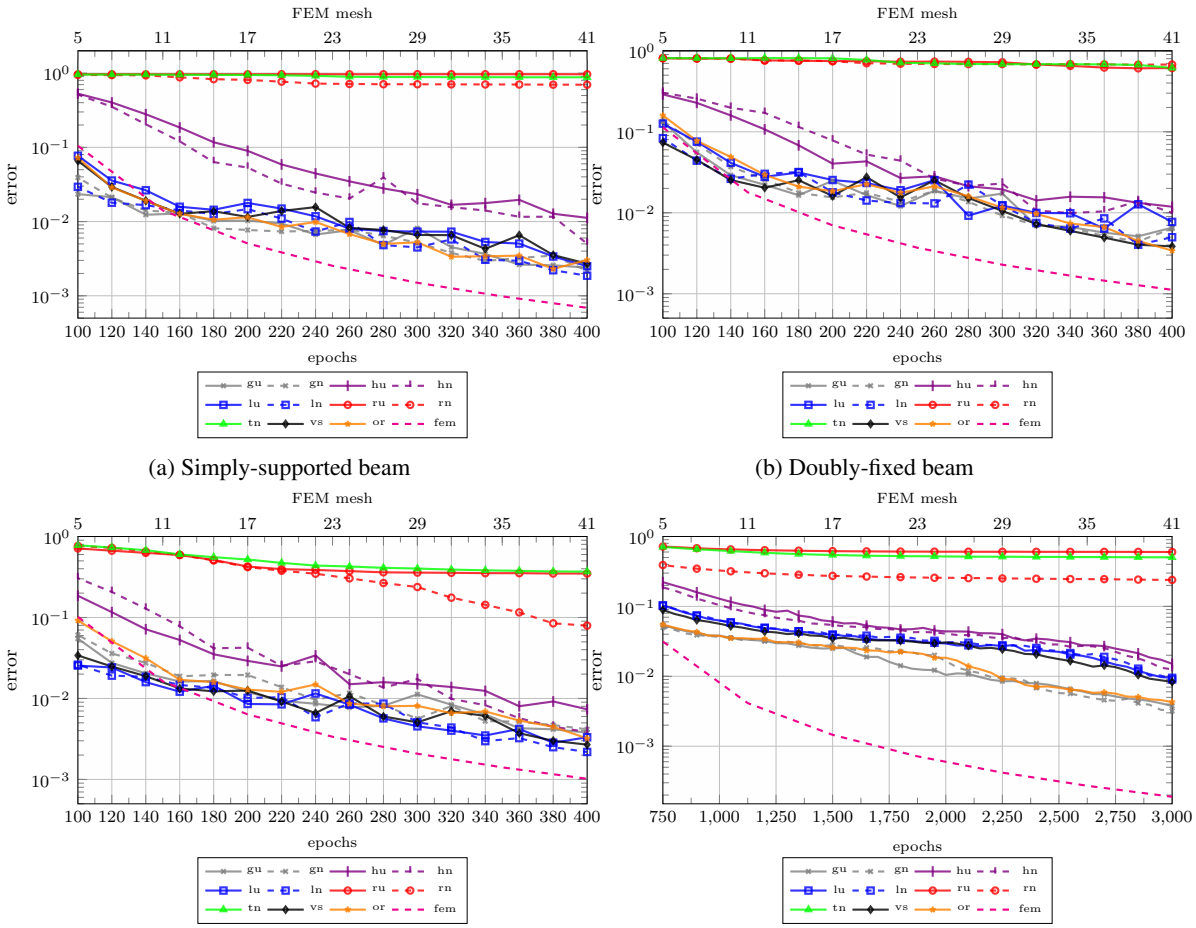
The quality of the predictions obtained from the trained model is evaluated using an error measure based on reference values, which in our case are the analytical solutions. The error measure used is shown in eq. (7) below:

$$e = \left[\frac{\sum_{i=1}^{N_n} (v_i^{ref} - v_i^{num})^2 + \sum_{i=1}^{N_n} (\theta_i^{ref} - \theta_i^{num})^2}{\sum_{i=1}^{N_n} (v_i^{ref})^2 + \sum_{i=1}^{N_n} (\theta_i^{ref})^2} \right]^{\frac{1}{2}} \tag{7}$$

where v_i^{ref} and θ_i^{ref} are the reference solutions in terms of deflection and rotation at a particular collocation point i, respectively, while v_i^{num} and θ_i^{num} are the corresponding numerical solutions, and N_n is the total number of collocation points. Linear FEM model results are also compared with PINN. In this case, the collocation points are replaced by the nodes of the discretisation for the error evaluation.

A PINN model was trained using the eleven weight initialisers described in section 3, and the errors were

evaluated. To mitigate the variability in results due to PINN training, error values were averaged over 10 simulations for each boundary condition using each weight initialiser. The corresponding error values, computed using eq. (7), are shown in Fig. 5. In these plots, the vertical axis represents the error in log scale, the bottom horizontal axis corresponds to the epochs of the PINN simulations, and the top horizontal axis represents the node discretisation of the linear FEM.



(a) Simply-supported beam

(b) Doubly-fixed beam

(c) Fixed-pinned beam

(d) Cantilever beam

Figure 5. Error values considering a uniformly distributed load over the beams span: PINNs x FEM

As it can be observed from Fig. 5, except for the cantilever beam, the results of PINNs simulations after training are in good agreement with the analytical solutions, with an error of the order of 10^{-3}, and overall accordance with the FEM simulations. As shown in Fig. 5d, the PINNs take longer to learn the cantilever beam behaviour, even after increasing the number of epochs to 3000, the outcome was not as good as the other cases. This difficulty might be associated to the fact that the right hand side boundary conditions are dependent of the derivatives of the approximated field variables (shear and bending moment), instead of the deflections and rotations directly. In addition to that, the error results of FEM simulations for the cantilever beam is smaller than the other boundary conditions, which contributes to increase the discrepancy when compared to the PINN results.

Regarding the initialisers, Fig. 5 reveals three distinct regions of interest. The first region includes the **random** (normal and uniform) and **truncated normal** initialisers, where learning was not successful in the mean sense. This failure might be associated with vanishing gradients or local minima issues. The second region is represented by the **He** initialiser, which, although convergent, yielded slightly worse results compared to the other initialisers. Finally, all the other initialisers produced similar outcomes, with no clear evidence to favour any particular one over the others. Although there are existing studies on weight initialisation [14, 15], which are generally developed with a particular activation function in mind, there is no consensus on how to initialise weights for PINN models in general.

5 Conclusions

In this paper, we conducted a preliminary investigation into the effect of weight initialisation on the training results of a Timoshenko beam using the $tanh$ activation function. The numerical simulations indicated that many initialisers are suitable for training the PINN model. Of the 11 initialisers analysed, the **random** (normal and uniform) and **truncated normal** initialisers should be avoided. Although the **He** (normal and uniform) initialiser produced reasonable results, other initialisers were found to be preferable. Future work will involve more robust simulations to provide further insights into the role of weight initialisers for PINN models in the context of the Timoshenko beam with the defined settings.

Acknowledgements. The authors gratefully acknowledge the financial support of the CAPES (in portuguese: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior)

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.
[2] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, vol. 9, n. 5, pp. 987–1000, 1998.
[3] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, vol. 2, n. 5, pp. 359–366, 1989.
[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, vol. 2, n. 4, pp. 303–314, 1989.
[5] D. Katsikis, A. D. Muradova, and G. E. Stavroulakis. A gentle introduction to physics-informed neural networks, with applications in static rod and beam problems. *Journal of Advances in Applied & Computational Mathematics*, vol. 9, pp. 103–128, 2022.
[6] K. A. Luong, T. Le-Duc, and J. Lee. Deep reduced-order least-square method—a parallel neural network structure for solving beam problems. *Thin-Walled Structures*, vol. 191, 2023a.
[7] K. A. Luong, T. Le-Duc, and J. Lee. Automatically imposing boundary conditions for boundary value problems by unified physics-informed neural network. *Engineering with Computers*, pp. 1–23, 2023b.
[8] Z. Chen, S. K. Lai, and Z. Yang. At-pinn: Advanced time-marching physics-informed neural network for structural vibration analysis. *Thin-Walled Structures*, vol. 196, 2024.
[9] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, vol. 3, n. 5, pp. 551–560, 1990.
[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.
[11] E. Haghighat and R. Juanes. Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, vol. 373, pp. 113552, 2021.
[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org, 2015.
[13] F. Chollet and others. Keras, 2015.
[14] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.
[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.