Applying Physics-Informed Extreme Learning Machines to Solve the Euler-Bernoulli Beam Problem

Shi Pan[1,3], **Haolin Li**[2,4,*]

[1]Shanghai World Foreign Academy, 400 Baihua Street, Xu Hui District, Shanghai
[2]Imperial College London, South Kensington, London

[3]spans061104@gmail.com
[4]Feliz19981004@gmail.com
*corresponding author

Abstract. Physics-Informed Neural Networks (PINNs) have been recently utilised to solve forward and backward partial differential equation problems. In this paper, we explore an alternative approach by using Physics-Informed Extreme Learning Machines (PIELM) to address the Euler-Bernoulli beam problem. We experiment with different types of functions combined with various numbers of hidden layers to identify the most effective techniques and conditions for solving partial differential equations with greater efficiency and accuracy. Both the traditional PINN and PIELM methods are applied, and their results are compared. Our findings demonstrate that PIELM offers more efficient computation while maintaining comparable accuracy in the results.

Keywords: Beam theory, partial differential equations, physics-informed neural networks, extreme learning machines

1. Introduction

Partial differential equations (PDEs) is always a complex but fundamental task in science and engineering. Recent solutions of PDE are usually based on analytical and numerical ways. For simple problems, the analytical method can be employed to get a strong form solution of certain PDEs. However, the strong form simulation embedded in analytical methods also restricts the solvable domain of PDEs [1]. To solve generalized PDEs, weak form formulations are usually adopted to solve PDE problems numerically. The Weighted Residual Method is one of the most useful method for the weak form formulations which its representatives include the Collocation Method [2], Subdomain method [3] and also Galerkin Method [4] where the Galerkin method outcomes the best accuracy. Weak formulations has been utilized in many numerical implementations such as the Finite Difference Method (FDM)[5], Finite Volume Method [6] and the Finite Element method [7]. For techniques of separation variables, its quite straightforward but its limited to specific problems and not applicable to non-linears PDEs.

In alternative to traditional numerical method, the Physics-Informed Neural Networks (PINNs) has recently been proposed and served as a new method to solve PDEs[8]. Different from traditional numerical solutions which usually requires discretization of certain solving domains, PINNs solve PDEs by the neural network representations which gives a continuous solution to certain functional problems [9]. Its application involves solving both backward and forward PDE problems as implemented by researchers[10].

Their results show the superiority of PINNs in some aspects,e.g.: (a) theoretically cheaper computations in certain problems and much more friendly memory required by the solution representatives than the traditional numerical methods; and (b) feasibility in solving inverse problems which vague boundary conditions that are difficult to be solved by numerical methods[11].

Beam theory is a fundamental aspect of structural engineering and mechanics that deals with the behavior of beams under various loads. Key features including small deformations, neglecting shear deformation and homogeneous material [12] can favour the uses of beam theory in mechanical engineering [13] such as designing and analysising bridge structures and also in aerospace engineering [14] such as designing of aircraft wings, fuselages and other structural components to ensure adequate strength and stiffness during flight.

In the context of beam theory, Physics-Informed Neural Networks (PINNs) can be also effective for modelling and solving problems related to the behavior of beams under various loading. beam theory is a mathematical simulation of beam model, which is faster and more concise than the traditional fully discrete model. Timoshenko beam theory [15] give us the description of beam theory PDE which gives us the method and theoretical support for solving the beam problem.The finite element methods are usually employed to solve complex beam problems where analytical methods fall short. Alternatively, there are also works focusing on solving the beam problem by PINNs [16]. PINNs offer a powerful approach to solve beam problems, which can be adapted to solve both static and dynamic beam problems. In addition to the traditional PINNs, it has proposed the Physics-Informed Extreme Learning (PIELM) Machines which can approximate complex functions and have been applied to various regression and classification problems based on the combination of the extreme learning machine architecture with the physical information [17]. From using PINN model combine with ELM, we can exchange different variables, such as types of functions and size of the layer, using hidden layer which is an inverse matrix to increase the efficiency as well as the accuracy of the solutions. PIELM has higher accuracy and faster speed compared to the techniques above [17].

In this paper, we utilised the PIELM (Physics-Informed Extreme Learning Machines) to solve beam problems. We exchange different independent variables including types of function, number of hidden layers and measure the prediction time and training time error respectively , using diagrams to represent the trend which find that the sigmoid function requires relatively less time and give higher precision and accuracy. Moreover, the extreme learning machine can hugely increase the efficiency.

2. Euler-Bernoulli beam theory and FE Formulation

Beam theory, particularly the Euler-Bernoulli beam theory, is a cornerstone in the field of structural engineering and mechanics. This classical theory provides a simplified approach to analysing the behaviour of slender, prismatic beams under various loading conditions. Developed in the 18th century by Leonhard Euler and Daniel Bernoulli, the theory reduces the complex three-dimensional stress state in a beam to a one-dimensional problem, facilitating easier analysis.

The Euler-Bernoulli beam theory is governed by a fourth-order partial differential equation (PDE) that relates the beam's deflection $w(x)$ to the applied load $q(x)$. The governing equation is given by:

$$EI\frac{d^4w(x)}{dx^4} = q(x), \tag{1}$$

where E is the Young's modulus of the beam material, I is the second moment of area of the beam's cross-section, and x is the longitudinal coordinate along the beam's length. This equation assumes that plane sections remain plane and perpendicular to the neutral axis, and that the deflections are small.

While the Euler-Bernoulli beam theory provides a robust analytical tool for simple beam configurations and loading conditions, real-world engineering problems often involve complex geometries, material properties, and loading scenarios that are difficult to solve analytically. This is where the Finite Element Method (FEM) becomes invaluable.

The Finite Element Method (FEM) is a powerful numerical technique used in the field of structural engineering for solving complex problems that are difficult to address analytically. Its application to beam problems involves several key steps, starting from the classical beam theory and progressing through to the discretization of the beam into finite elements.

The classical Euler-Bernoulli beam theory is encapsulated by a fourth-order partial differential equation, which describes the relationship between the beam's deflection and the applied loading:

$$EI\frac{d^4w(x)}{dx^4} = q(x), \tag{2}$$

where EI represents the flexural rigidity of the beam, combining the Young's modulus E and the second moment of area I, $w(x)$ denotes the deflection of the beam at position x, $q(x)$ is the distributed load per unit length acting on the beam.

The weak form is obtained by integrating the strong form of the equation by parts. This method reduces the continuity requirements for the solution, making it more amenable to numerical approximation. The process involves integrating the product over the domain of the beam:

$$\int_0^L EI\frac{d^4w}{dx^4}v(x)\,dx = \int_0^L q(x)v(x)\,dx, \tag{3}$$

where L is the length of the beam, and applying integration by parts to the left-hand side of the equation, reducing the order of derivatives on $w(x)$. This results in:

$$\int_0^L \frac{d^2w}{dx^2}\frac{d^2v}{dx^2}EI\,dx = \int_0^L q(x)v(x)\,dx + \text{boundary terms}, \tag{4}$$

where the boundary terms depend on the specific boundary conditions applied to the beam.

By the introduced weak formulation of the Euler-Bernoulli beam equation, in the Finite Element Method, the continuous domain of the beam is discretised into a finite number of elements. Each element is treated as an individual entity with its own local approximation of the displacement field, typically using polynomial basis functions. The global system of equations is assembled from the contributions of all elements, and the resulting system of algebraic equations is solved to obtain the approximate deflections across the beam. Within each element, the displacement $w(x)$ and the rotation $\theta(x)$ are approximated using shape functions $N_i(x)$. Typically, cubic polynomials are used for $w(x)$ because they can sufficiently model the curvature due to bending:

$$w(x) \approx \sum_{i=1}^{n} w_i N_i(x), \tag{5}$$

$$\theta(x) \approx \sum_{i=1}^{n} \theta_i N_i(x), \tag{6}$$

where w_i and θ_i are the nodal values of displacement and rotation, respectively.

3. Physics-informed Extreme Learning Machine

3.1. Physics-informed neural networks

Physics Informed Neural Networks (PINNs) integrate deep learning with physical laws described by differential equations. This method is particularly useful in scenarios where traditional numerical methods are computationally expensive or infeasible. PINNs can be effectively applied to solve the Euler-Bernoulli beam theory, which is governed by a fourth-order differential equation.

The Euler-Bernoulli equation for beam deflection is expressed in Eq.1. To apply PINNs to this problem, one constructs a neural network $\mathcal{N}$ whose output approximates the deflection $w(x)$. The network is trained to minimize a loss function that typically includes several terms to ensure both accuracy and adherence to physical laws:

$$\mathcal{L}(\theta) = \mathcal{L}_{data}(\theta) + \mathcal{L}_{PDE}(\theta) + \mathcal{L}_{BC}(\theta) \tag{7}$$

where θ denotes the neural network parameters. In Eq.7, $\mathcal{L}_{data}(\theta)$ represents the data loss:

$$\mathcal{L}_{data}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \left| w_{predicted}(x_i) - w_{actual}(x_i) \right|^2, \tag{8}$$

$\mathcal{L}_{PDE}(\theta)$ denotes the PDE loss which in this case represents the residual of the Euler-Bernoulli equation computed at several points along the beam:

$$\mathcal{L}_{PDE}(\theta) = \frac{1}{M}\sum_{i=1}^{M} \left| EI\frac{d^4 w_{\mathcal{N}}}{dx^4}(x_i;\theta) - q(x_i) \right|^2, \tag{9}$$

$\mathcal{L}_{BC}(\theta)$ is the boundary condition loss.

3.2. Physics-informed Extreme Learning Machine (PIELM)

The Extreme Learning Machine (ELM) is a fast learning algorithm for single-hidden layer feedforward neural networks (SLFNs). This method uniquely assigns input weights and biases randomly and determines the output weights analytically, circumventing common issues found in gradient-based learning algorithms like slow convergence and the local minima problem.

ELM is designed to train single-hidden layer feedforward neural networks efficiently. It initializes the weights and biases of the hidden layer randomly and fixes them throughout the learning process. The training then involves calculating the output weights that link the hidden layer to the output layer, significantly simplifying and speeding up the learning process.

For a given training set $\{(x_i, t_i)\}_{i=1}^{N}$, where x_i is an input vector and t_i is the corresponding target output, the operation of the hidden layer in ELM can be represented as follows:

$$H\beta = T \tag{10}$$

where H is the hidden layer output matrix, β represents the output weights, and T is the matrix of target values. The output weights β are typically determined by finding the Moore-Penrose generalized inverse of H, providing an optimal least-squares solution to this equation:

$$\beta = H^{-1}T \tag{11}$$

The Physics Informed Extreme Learning Machine (PI-ELM) extends the ELM by incorporating physical laws directly into the learning algorithm, enhancing its application in scientific computing and engineering disciplines where compliance with physical laws is essential.

In actual implementations, the loss function given in Eq.7 is usually modified to the following expression,

$$\mathcal{L}(\theta) = \lambda_1\mathcal{L}_{data}(\theta) + \lambda_2\mathcal{L}_{PDE}(\theta) + \lambda_3\mathcal{L}_{BC}(\theta), \tag{12}$$

where λs denote the weight of each loss to be considered. Therefore, a basic step to use PIELM to solve a beam theory is stated as follows:

(i) Assign the input layer weights randomly;
(ii) Get the input layer outputs based on the data, PDE and boundary conditions defined in the problem;
(iii) Assemble the three sets of equations in the form of $H\beta = T$;
(iv) Output layer weight vector is given by pseudo-inverse of H.

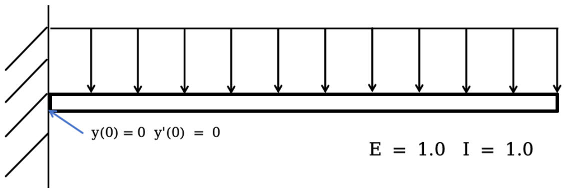

Figure 1. Beam Problem

4. Results

We focus on the problem that a beam model, one end is fixed, apply a uniform load on the beam, to solve for its displacement, with the partial equations:

$$EI\frac{d^2y}{dx^2} = P(L-y), \tag{13}$$

which E represents Young Modulus, I represent Second moment of the cross section, P represent Applied load and L represent the length of the beam. And the boundary conditions $y(0) = 0$, $y'(0) = 0$ is also given. Fig.1 shows the beam problem to be solved. The code for running the PIELM to solve the studies beam problem is given in the appendix.

For the prediction time shown in Fig.2, for the Extreme Learning Machine (ELM), the three activation functions exhibit similar trends, with approximately 2×10^{-4} seconds. However, an anomalous point occurs when the hidden size is 70. Thus, choosing a smaller hidden size may make the sigmoid function a better option. On the other hand, when it comes to prediction time in the Physics-Informed Neural Network (PINN) model, the time needed is consistently higher compared to the ELM, indicating that the ELM is more efficient for solving problems. Additionally, the hyperbolic tangent (tanh) function consistently takes the longest time to compute. Despite an outlier at a hidden size of 20, both the Rectified Linear Unit (ReLU) and sigmoid functions are viable options. However, the error margin for the ReLU function is broader compared to the sigmoid function, indicating more fluctuation. For larger hidden sizes, the sigmoid function might be preferable. In summary, the Extreme Learning Machine (ELM) is more efficient than the Physics-Informed Neural Network (PINN) model, and the sigmoid function is effective for both techniques .

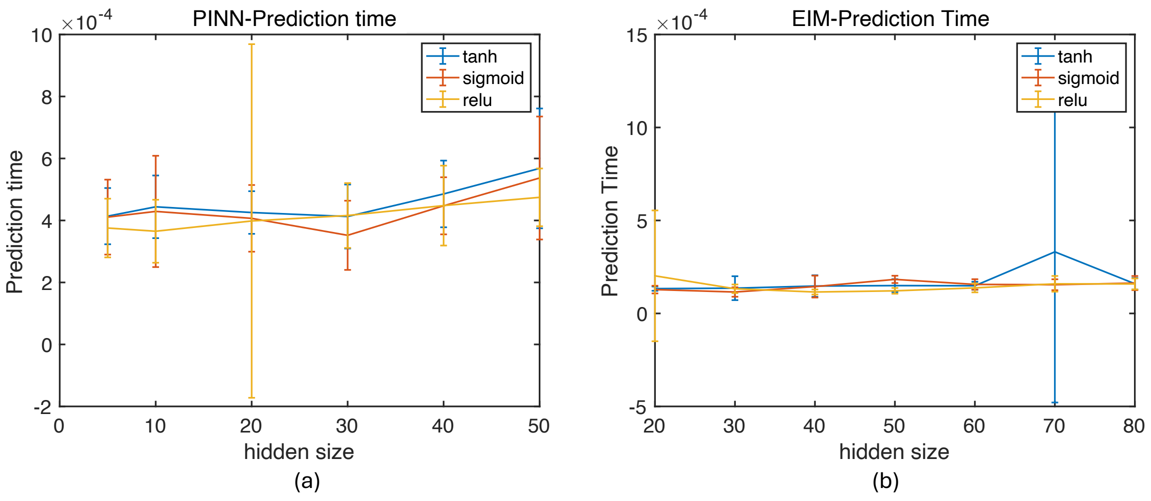

Figure 2. Prediction time for (a) PINN and (b) PIELM.

With respect to the data in Fig.3 about the training time of the PIELM, all three activation functions show an increasing trend. The sigmoid function always requires the most time, the tanh function is second, and the ReLU function is the most efficient, taking around 0.2 seconds for a hidden size of 80. In the Physics-Informed Neural Network (PINN) model, the trend and ranking of the activation functions are similar to those in the ELM, but the training times are significantly longer. For the ReLU function with a hidden size of 80, the PINN model takes around 25 seconds for training.

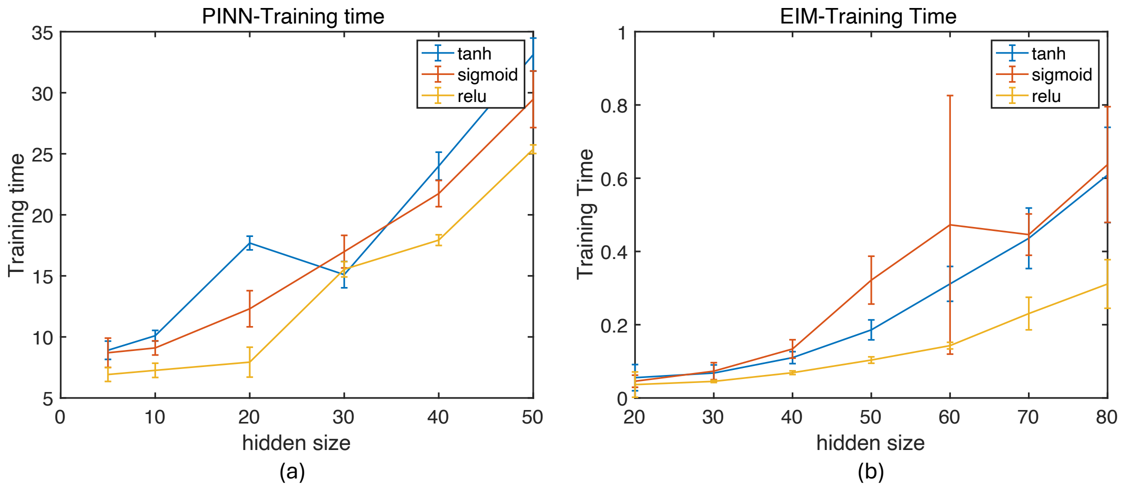

Figure 3. Training time for (a) PINN and (b) PIELM.

Fig.4 gives the errors of the two models compared to the analytical solutions. In the case of the Extreme Learning Machine (ELM), the tanh function shows a very large error that cannot be displayed on the graph. On the other hand, the ReLU function exhibits fluctuating errors, varying from 1×10^{-9} to 4×10^{-9}, while the sigmoid function has negligible error. In the Physics-Informed Neural Network (PINN) model, the ReLU function has a very large error, the tanh function has negligible error, and the sigmoid function has a small error when dealing with lower hidden sizes, around 0.2.

As demonstrated by Fig.4, the sigmoid function shows the least error in both methods, and the error in ELM is less than in PINN, making ELM the more accurate approach.

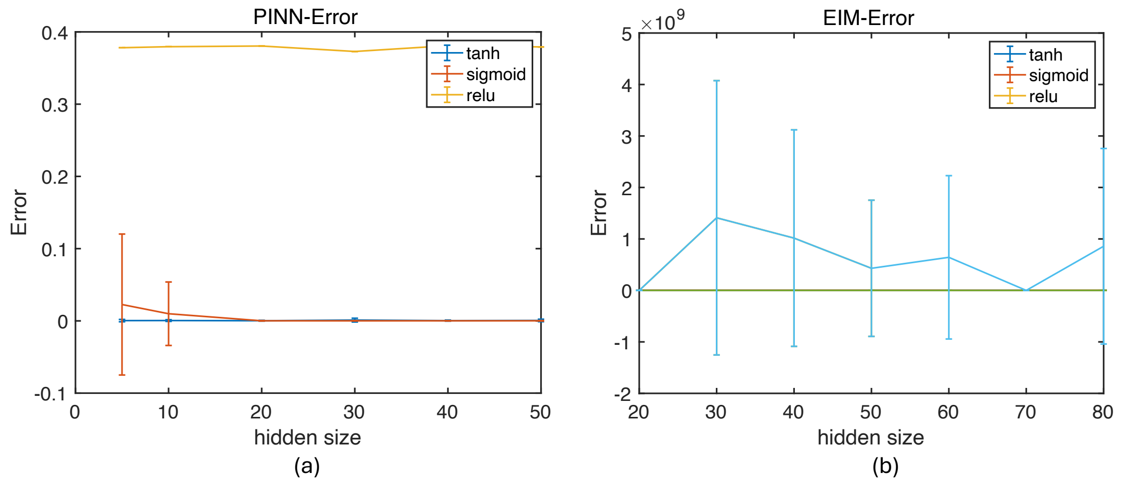

Figure 4. Prediction errors for (a) PINN and (b) PIELM.

Considering training time, prediction time, and error, the sigmoid function requires relatively less time and produces more accurate results. Additionally, the Extreme Learning Machine significantly increases efficiency.

5. Conclusion

In this study, we examined the efficiency and accuracy of the Physics-Informed Extreme Learning Machines (PIELM) in solving the Euler-Bernoulli beam problem, comparing its performance against the traditional Physics-Informed Neural Networks (PINNs). Our results indicate that PIELM consistently outperforms PINNs in terms of computational efficiency, requiring significantly less time for both training and prediction tasks.

Among the activation functions tested, the sigmoid function emerged as the most effective, demonstrating the least error and maintaining stability across different hidden layer sizes. Although the ReLU function was more efficient in terms of training time, it exhibited larger fluctuations in error, making the sigmoid function a more reliable choice.

Overall, the PIELM method not only achieves higher computational efficiency but also maintains accuracy, making it a superior alternative to PINNs for solving partial differential equations, particularly in the context of beam deflection problems. The findings suggest that PIELM, coupled with the sigmoid activation function, offers a promising approach for future applications requiring both speed and precision.

References

[1] Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Society, 2022.

[2] RD Russell and Lawerence F Shampine. A collocation method for boundary value problems. *Numerische Mathematik*, 19:1–28, 1972.

[3] Waad Subber and Abhijit Sarkar. A domain decomposition method of stochastic pdes: An iterative solution techniques using a two-level scalable preconditioner. *Journal of Computational Physics*, 257:298–317, 2014.

[4] Bernardo Cockburn, George E Karniadakis, and Chi-Wang Shu. The development of discontinuous galerkin methods. In *Discontinuous Galerkin methods: theory, computation and applications*, pages 3–50. Springer, 2000.

[5] Randall J LeVeque. Finite difference methods for differential equations. *Draft version for use in AMath*, 585(6):112, 1998.

[6] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

[7] Carlos A Felippa. Introduction to finite element methods. *University of Colorado*, 885, 2004.

[8] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

[9] Yanan Guo, Xiaoqun Cao, Bainian Liu, and Mei Gao. Solving partial differential equations using deep learning and physical constraints. *Applied Sciences*, 10(17):5917, 2020.

[10] Lei Yuan, Yi-Qing Ni, Xiang-Yun Deng, and Shuo Hao. A-pinn: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. *Journal of Computational Physics*, 462:111260, 2022.

[11] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

[12] Vittorio Giavotto, Marco Borri, Paolo Mantegazza, Gianluca Ghiringhelli, V Carmaschi, GC Maffioli, and F Mussi. Anisotropic beam theory and applications. *Computers & Structures*, 16(1-4):403–413, 1983.

[13] Zdenek P Baiant and Raymond J Krizek. Journal of the engineering mechanics division. *a a*, 4:8, 1980.

[14] Erasmo Carrera, Gaetano Giunta, and Marco Petrolo. *Beam structures: classical and advanced theories*. John Wiley & Sons, 2011.

[15] MARK Levinson. A new rectangular beam theory. *Journal of Sound and vibration*, 74(1):81–87, 1981.

[16] David Yang Gao. Nonlinear elastic beam theory with application in contact problems and variational approaches. *Mechanics Research Communications*, 23(1):11–17, 1996.

[17] Vikas Dwivedi and Balaji Srinivasan. Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations. *Neurocomputing*, 391:96–118, 2020.

Appendix

Code of the PIELM method:

```
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
import torch.autograd as autograd
import torch.nn as nn
from torch.autograd import Variable
from timeit import default_timer

input_data = torch.linspace(0,1,100)
target_data = input_data**2

def compute_elm_output_weights(input_data, target_data, \
hidden_layer_weights, hidden_layer_bias):
    # Forward pass through the hidden layer
    # Note: Assuming a sigmoid activation function here
    hidden_layer_output = F.sigmoid(torch.matmul(input_data, \
    hidden_layer_weights) + hidden_layer_bias)

    # Calculate pseudoinverse of hidden layer output
    pseudo_inverse = torch.pinverse(hidden_layer_output)

    # Compute output layer weights
    output_weights = torch.matmul(pseudo_inverse, target_data)

    return output_weights

# Parameters
input_size = 1
hidden_size = 300
output_size = 1

# Randomly initialize weights and biases for the hidden layer
hidden_layer_weights = torch.randn(input_size, hidden_size)
hidden_layer_bias = torch.randn(hidden_size)

# Assume input_data and target_data are given as torch.
input_data = torch.linspace(0,1,300)[:,None]
target_data = input_data**2

# Compute the output layer weights
output_layer_weights = compute_elm_output_weights(input_data, \
target_data, \
hidden_layer_weights, hidden_layer_bias)

# PDE parameters
E = 1.0 # Elastic modulus
I = 1.0 # Second moment of the cross section
P = -1.0 # Applied load
L = 2.0 # length of the beam

# Parameters
```

```
input_size = 1
hidden_size = 40
output_size = 1

# Randomly initialize weights and biases for the hidden layer
hidden_layer = nn.Linear(input_size, hidden_size)

pde_points = 100
PDE_input = torch.linspace(0,1,pde_points)
PDE_input = Variable(PDE_input[:,None].float(), \
requires_grad=True)
BC_input = Variable(torch.zeros(1)[:,None].float(), \
requires_grad=True)

# Training
t1 = default_timer()
# BC training
out_bc = torch.tanh(hidden_layer(BC_input))
# PDE training
out_pde = torch.tanh(hidden_layer(PDE_input))
# Differentiation of BC points
dout_bc_dx = torch.zeros(1,hidden_size)
for i in range(hidden_size):
  dout_bc_dx[:,i] = (autograd.grad(out_bc[:,i],BC_input, \
  torch.ones_like(out_bc[:,i]), \
  retain_graph=True,create_graph=True)[0]).squeeze()
# Differentiation of PDE points
dout_pde_dx = torch.zeros(pde_points,hidden_size)
dout_pde_dxx = torch.zeros(pde_points,hidden_size)
for i in range(hidden_size):
  dout_pde_dx[:,i] = \
  (autograd.grad(out_pde[:,i],PDE_input,torch.ones_like( \
  out_pde[:,i]), \
  retain_graph=True,create_graph=True)[0]).squeeze()
  dout_pde_dxx[:,i] = (autograd.grad(dout_pde_dx[:,i], \
  PDE_input, \
  torch.ones_like(dout_pde_dx[:,i]), \
  retain_graph=True,create_graph=True)[0]).squeeze()

H_bc1 = out_bc * 50000
K_bc1 = torch.zeros(1,1)

H_bc2 = dout_bc_dx * 1000
K_bc2 = torch.zeros(1,1)

H_pde = E * I * dout_pde_dxx
K_pde = P*(L-PDE_input)

H = torch.cat((H_bc1,H_bc2,H_pde),dim=0)
K = torch.cat((K_bc1,K_bc2,K_pde),dim=0)
```

```
# Calculate pseudoinverse of hidden layer output
pseudo_inverse = torch.pinverse(H)

# Compute output layer weights
output_weights = torch.matmul(pseudo_inverse, K)
t2 = default_timer()

# Prediction
# Groud truth
groudtruth = P*PDE_input**2/(6*E*I)*(3*L-PDE_input)
# Forward pass through the hidden layer
input_pred = torch.linspace(0,1,pde_points)[:,None]

t1p = default_timer()
prediction = torch.matmul(torch.sigmoid( \
hidden_layer(input_pred)) \
,output_weights)
t2p = default_timer()

error = torch.mean(abs(prediction - groudtruth))/ \
torch.max(abs(groudtruth))
print('Training Time:',t2-t1,'Prediction Time:',t2p- \
t1p,'Error:',error.detach().item(), \
'Number of Parameters:',hidden_size*2)
```