Contents

Contents iii

List of Figures v

List of Tables vi

1 Introduction 1
1.1 The Artificial Neural Network's Architecture 1

2 Literature Review 4

3 Methodology 7
3.1 PINN feedforward 8
3.2 Backpropagation in PINN training 9

4 Results 11
4.1 Rod with Axial Uniformly Distributed load 11
4.2 Simply Supported Beam with UDL 16
4.3 Cantilever Beam with UDL 18
4.4 Propoed Cantiliver Beam with UDL 22
4.5 Cantiliver Beam with Uniformly Varying Load 23
4.6 Simply Supported Beam with Multipoint loading 26
4.7 Cantilever Beam with Multi UDL 28

5 Conclusions and Discussions 32

6 Future Work 34

7 References 36

List of Figures

1.1 Artificial Neural Network 2

3.1 PINN Architecture 9

4.1 Rod with UDl along Axis with fixed at one end 12
4.2 Rod's PINN Architecture 13
4.3 The true and PINN solutions of Rod 16
4.4 Simply Supported Beam with Uniformly Distrubuted Loading 16
4.5 PINN's Architecture for Simply Supported with UDL case 18
4.6 True solution and DNN solution for SSB with UDL 19
4.7 cantiliver Supported Beam with Uniformly Distrubuted Loading . . . 19
4.8 PINN of Cantiliver Beam with UDL 20
4.9 DNN solution vs True Solution 21
4.10 Propped Cantilever beam with Uniform Distributed Load 21
4.11 PINN Architecture of Propped Cantilever Beam 23
4.12 DNN vs True solution for Propped Beam with UDL 24
4.13 Cantilever Beam with Uniformly Varying Load 24
4.14 ANN solution vs True solution for Cantilever with UVL 26
4.15 SSB with Multi Point Loading 26
4.16 DNN vs True solution of simply supported Beam with Multi point loading 28
4.17 Cantiliver Beam with Multi UDL conditions 29
4.18 DNN vs True Solution of Cantilever Multiple Udl conditions 31

List of Tables

4.1 Optimization parameters of Neural Network for the case of Rod . . . 15
4.2 Optimized parameters for SSB UDL case 18
4.3 Optimized parameters of PINN for case of Cantilever with UDL . . . 22
4.4 Optimized Parameters for Propped beam 23
4.5 Optimized Parameters for Cantilever with UVL 25
4.6 Optimized Parameters for SSB with Multi Point Loading condition . 28
4.7 Optimized Parameters of Cantiliver with multiple loading conditions . 30

Chapter 1

Introduction

1.1 The Artificial Neural Network's Architecture

Artificial neural networks, like biological neural networks, are made up of neurons as their fundamental units.These neurons take in data from the network or from other neurons, process it, and then send it to the following group of neurons.

The following expression is computed for each neuron in the neural network's input layer, assuming that $x_1, x_2, \ldots, x_n$ represent the inputs for a specific neuron.

$$a = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b = \sum_{i=1}^{n} w_ix_i + b, \tag{1.1.1}$$

Here $w_1, w_2, \ldots, w_n$, are weights and b,the bias and the number of inputs be n.

We also present $'a'$ computed for the activation function, $f(x)$. In ANNs, the activation function is crucial because it determines whether or not to activate a neuron based on how important its input is to the prediction process. The Sigmoid,tanh and ReLU activation functions are the most frequently used ones. The sigmoid nonlinear function, for instance,

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{1.1.2}$$

and the tanh activation function expression is

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{1.1.3}$$

Due to its smooth behavior, the sigmoid is essential to the logistic regression classification problem. The tanh, which also exhibits a smooth and nonlinear characteristic, is another activation function used in this study.

An Artificial Neural Network (ANN) is made up of the interconnected neurons in the neural network that was just described. Feedforward refers to the method by which data travels through hidden layers from the input layer to the output layer. An illustration of one such neural network is shown in Figure 1.1.

The following weight matrices should be introduced in order to better understand how an ANN performs its feedforward process. The weights between successive hidden layers should be represented by the matrix W_1,W_2...,W_n. $k = 1, 2, \ldots, N_L$, where N_L stands for the total number of layers. The corresponding inputs are denoted as $i = 1, 2, \ldots, n_0$ (where $n_0 = n$), and the input layer is given the number 0.

Figure 1.1 makes the following assumptions: $N_L = 3$, $n_0 = 4$, $n_1 = n_2 = 5$, and $n_3 = 3$. The matrices W_1, W_2, and W_3 are therefore determined to have dimensions of (4×5),$(5 \times 5), and (5 \times 3), respectively.$

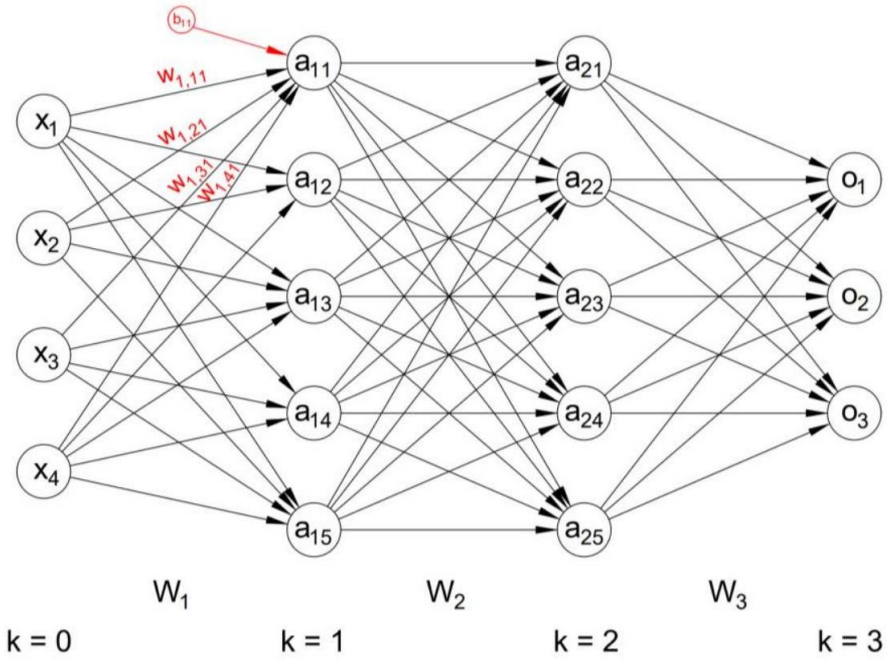

FIGURE 1.1: Artificial Neural Network

The output of each Neuron can be calculated by the equation (1.2.1) and equation (1.2.3). The Neural Network training process is an essential step in Deep Learning

because it allows for the updating of weights and biases by minimising the loss function.Calculating the derivatives of loss with respect to weights and biases makes it possible to perform these updates.These updates can be computed in PyTorch using chainrule and backpropagation.The general steps for training the neural network are as follows:

1. Enter the parameters of the problem, epochs, and the input dataset for the ANN in the forward pass.
2. Calculating the loss function
3. Backpropagation is the process of going backward from the most recent layer and calculating the derivatives of the Loss function with respect to each trainable parameter.
4. In order to minimise the Loss function, adjust the trainable parameters in that direction.
5. For all of the training set's data, repeat above steps

Follow the steps above to repeat or train the network for the entire dataset for each of the epochs mentioned. Each individual pair of inputs and outputs must have its parameters adjusted and the loss function calculated for 1 step(one whole cycle). This requires a huge computational cost that extends the algorithm's convergence time. Because of this,the given data-set is generally divided into discrete groups called batches in applications. The Adam and the GD(Gradient Decent) Algorithms are the two primary optimisation algorithms used in Deep Neural Network training. The network's weights and biases, which change in backpropagation, are taken into account when minimising the loss function by the gradient descent algorithm.The adaptive learning rate and momentum mechanisms of the Adam algorithm, which allow for effective convergence and the handling of sparse gradients, make it superior to other optimisation algorithms used in deep learning.

Chapter 2

Literature Review

"Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations" by Maziar Raissi, Paris Perdikaris, and George Em Karniadakis (2017) This paper introduces the PINN framework as a deep learning approach for solving both forward and inverse problems in nonlinear partial differential equations (PDEs). The authors demonstrate the efficacy of the method in a variety of applications, including fluid dynamics and solid mechanics.

"Solving Ordinary Differential Equations with Neural Networks" by Martin Hanke and Michael Obersteiner (1994) This paper proposes a neural network approach for solving ordinary differential equations (ODEs) by mapping the differential equations onto a neural network architecture. The authors demonstrate the effectiveness of the method on several test problems.

"DeepXDE: A Deep Learning Library for Solving Differential Equations" by Zichao Long, Yiping Lu, and Bin Dong (2020) This paper introduces DeepXDE, a deep learning library for solving differential equations, including both ODEs and PDEs. The authors demonstrate the effectiveness of the library on several benchmark problems, including the Navier-Stokes equations and the Allen-Cahn equation.

"Learning to Solve Partial Differential Equations Using Deep Learning" by Oksana Chkrebtii, Alexander G. Gray, and Kurt Keutzer (2018) This paper proposes a deep learning approach for solving PDEs based on a convolutional neural network architecture. The authors demonstrate the efficacy of the method on several benchmark problems, including the heat equation and the wave equation.

"A Physics-Informed Neural Network for Multiphase Flow in Porous Media" by Linan Zhang, Di Wang, and Liang Zhao (2018) This paper proposes a PINN-based approach for modeling multiphase flow in porous media. The authors demonstrate the effectiveness of the method on several test problems and show that it outperforms traditional numerical methods.

"Learning to Solve Differential Equations in Continuum Mechanics by Example" by Sebastian Hirsch, Michael Potter, and Michael Ortiz (2017) This paper proposes a deep learning approach for solving differential equations in continuum mechanics, based on a neural network architecture. The authors demonstrate the efficacy of the method on several benchmark problems, including the elastodynamics equations and the incompressible Navier-Stokes equations.

"A Hybrid Method of Physics-Informed Neural Networks and Gaussian Processes for Inverse Problems" by Huan Lei, Yulong Liu, and George Em Karniadakis (2020) This paper proposes a hybrid approach combining PINNs and Gaussian processes for solving inverse problems in PDEs. The authors demonstrate the effectiveness of the method on several test problems, including the Poisson equation and the Burgers equation.

"Solving High-Dimensional Partial Differential Equations Using Deep Learning" by Ting-Kam Leonard Wong, Jinchao Xu, and Xiu Yang (2019) This paper proposes a deep learning approach for solving high-dimensional PDEs based on a neural network architecture. The authors demonstrate the effectiveness of the method on several test problems, including the Black-Scholes equation and the Allen-Cahn equation.

"A Machine Learning Framework for Solving High-Dimensional Partial Differential Equations" by Dongkun Zhang, Bing Yu, and Chi-Wang Shu (2020) This paper proposes a machine learning framework for solving high-dimensional PDEs based on a neural network architecture. The authors demonstrate the efficacy of the method on several benchmark problems, including the Burgers equation and the Schrödinger equation.

"Physics-Informed Deep Learning for Fluid Dynamics: A Review" by George Em Karniadakis (2020) This paper provides a comprehensive review of the application of physics-informed deep learning methods for fluid dynamics problems. The author discusses the advantages and limitations of the PINN approach, and provides examples of its application to a wide range of problems, including

turbulence modeling, flow control, and multiphase flow. The paper also highlights some of the challenges that need to be addressed in order to fully realize the potential of PINNs for fluid dynamics applications.

Chapter 3

Methodology

Differential equations are used to formulate many engineering issues. These equations can be solved in a variety of ways. One approach entails locating an analytical solution, which is the precise expression of the given differential equations and initial/boundary conditions-satisfying unknown function $u(x)$. Utilising semi-analytical or numerical techniques, such as the Runge-Kutta of different orders or finite element methods (FEM), is another option. In these techniques, an approximate solution is used to approximate $u(x)$ point by point. The PINN uses an artificial neural network (ANN) to approximate the true solution $u(x)$ such that $u(x) \approx \hat{u}(x) = NN(x)$.

'A network that can be trained using a mathematical model imposed by both physics and mathematics is said to be "physics-informed". It is a type of self-supervised learning in which the differential equations themselves produce pairs of inputs and desired outputs for the training process. For static ordinary differential equation models or models involving partial differential equations, the inputs to the network are either one independent variable or two or more independent variables.These input values come from the domain of the $u(x)$ definition. The function $NN(x)$ is represented by the output that is produced during the feedforward process. This method differs from conventional ANNs in that, during the training process, in addition to computing the derivatives of the loss function for all trainable network parameters, the neural network's derivatives with respect to the input variables are also used. All partial derivatives of $\hat{u}(x)$ with respect to the independent variables are calculated in the case of a partial differential equation. The difference between

the approximation $\hat{u}(x)$ and the actual solution $u(x)$ is then assessed using a loss function.

3.1 PINN feedforward

This section examines the transfer of given data from the first layers to the intermediate hidden layers and ultimately to the neural network's output. We'll use a seconder differential to show how the process works:

$$\frac{d^2u(x)}{dx^2} + a\frac{du(x)}{dx} = b, \quad x \in (x_0, L], \quad L \in \mathbb{R} \tag{3.1.1}$$

$$u(x_0) = u_0, \quad \frac{du(x_0)}{dx} = u_1 \tag{3.1.2}$$

Here, the constant parameters a and b are used. A Physics-Informed Neural Network (PINN) is used to find the solution $u(x)$ to the problem defined by equations (1) and (2), and the output of the neural network, denoted as $NN(x)$, roughly approximates the solution $u(x)$. The computation of the output vector for each layer is described by the formulas below:

$$\begin{aligned}
\mathbf{z}_0 &= \mathbf{x} \\
\mathbf{z}_k &= \sigma\left(W_k^T \cdot \mathbf{z}_{k-1} + \mathbf{b}_k\right), \quad k = 1, 2, \ldots, N_L - 1. \\
\mathbf{z}_{N_L} &= W_{N_L}^T \cdot \mathbf{z}_{N_L-1} + \mathbf{b}_{N_L}
\end{aligned}$$

The final hidden layer has a linear relationship with the PINN's output. It is significant to note that, contrary to Artificial Neural Networks (ANNs) in general (see equation (2.6) with $k = N_L$), the output layer (equation (3)) does not pass through the activation function. In this case, the last hidden layer's outputs are represented by the vector z_{N_L-1}, the weights of the last layer is represented by the matrix $W_{N_L}^T$, and the bias is represented by the vector $\mathbf{b}_{N_L}$. As a result, the general solution can be written as:

$$\hat{u}(\mathbf{x}) = NN(\mathbf{x}) = W_{N_L}^T \cdot \mathbf{z}N_L - 1 + \mathbf{b}N_L \tag{3.1.3}$$

The collocation points for equation (1) are represented by the components of the input vector $\mathbf{x}$. It is important to note that each collocation point in the neural network is given a displacement value, denoted as $NN(x_i)$, where $i = 1, 2, \ldots, n$.

3.2 Backpropagation in PINN training

The goal of the training network is to get the function $NN(x) = \hat{u}(x)$ to satisfy the given mathematical differential equation and initial conditions with the least amount of error possible. Trainable parameters include weights and biases. The loss function, more specifically the mean square error, is minimised in order to adapt the weights and biases.

$$MSR = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{d^2\hat{u}\,(x_i)}{dx^2} + a\frac{d\hat{u}\,(x_i)}{dx} - b\right)^2 + \left(\hat{u}\,(x_0) - u_0\right)^2 + \left(\frac{d\hat{u}\,(x_0)}{dx} - u_1\right)^2 \tag{3.2.1}$$

Let the collocation points, $i = 1, 2, \ldots, n$, be represented by x_i. The error resulting from the differential equation as well as the errors resulting from the initial and boundary conditions must all be taken into account when constructing the loss function. The architecture of the Physics-Informed Neural Network (PINN) used in this example is shown in Figure 3.1. While the second block represents the

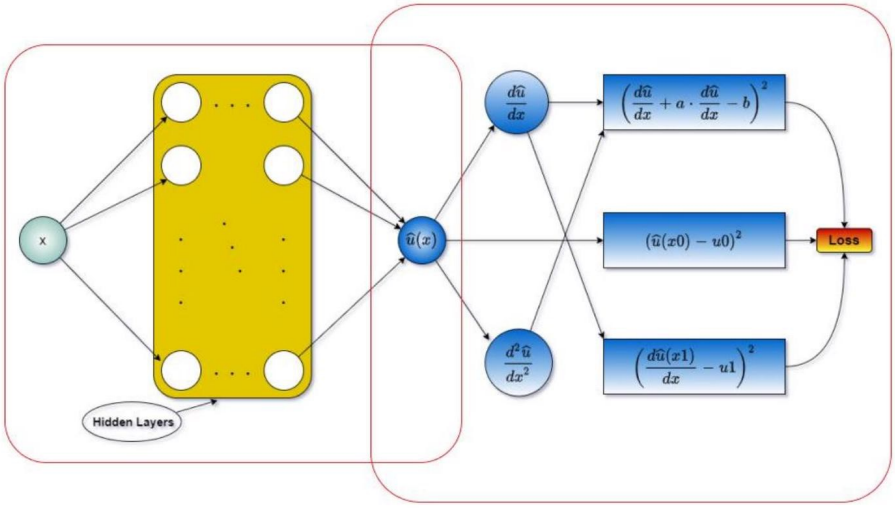

FIGURE 3.1: PINN Architecture

extension for learning using physics, the first block represents the back propagation

of neural network. The Gradient Descent and Adam method are examples of well-known optimisation algorithms that can be used with the training algorithm. The general procedure for implementing a Physics-Informed Neural Network (PINN) is as follows:

1. Provide the neural network with the necessary data, such as the collocation points, the initial conditions ,boundary conditions, weights, biases,epochs, and the mechanical problem's parameters.
2. Feedforward: Using the activation function, weights, and biases, determine $\hat{u}(x)$ as a function of the values of x_i.
3. Calculate the all derivatives of $\hat{u}(x)$ with respect to x subjected to initial and boundary values.
4. construct loss function like shown in block diagram.
5. Backpropagation: Calculate the weights in back propagation.
6. Utilising an optimisation strategy, minimise the loss function.
7. Update the NN's biases and weights parameters.
8. Repeat above procedure until we get considerable accuracy.

Chapter 4

Results

This section tests the PINN for a few engineering practical problems, especially for the solid deformable rod and beam models with different boundary conditions,intial conditions,support conditions and loading conditions. Ordinary differential equations, initial and boundary conditions, and other issues are taken into consideration.we currently focused on problems involving the L2 and L4 ordinary differential equations.we have solved following problems.

1. Rod with Axial Uniformly Distributed load
2. Simply Supported Beam with UDL
3. Cantilever Beam with UDL
4. Proped Cantilever Beam with UDL
5. Cantilever Beam with Uniformly Varying Load ,maximum intensity at the fixed end
6. Simply Supported Beam with Multipoint Loading
7. Cantilever Beam with Multi UDL

4.1 Rod with Axial Uniformly Distributed load

If a rod is subjected to distributed tensile forces denoted as cx, the load at an elementary point x can be represented by the integral $\int_x^L cxdx = \frac{c}{2}(L^2 - x^2)$, where

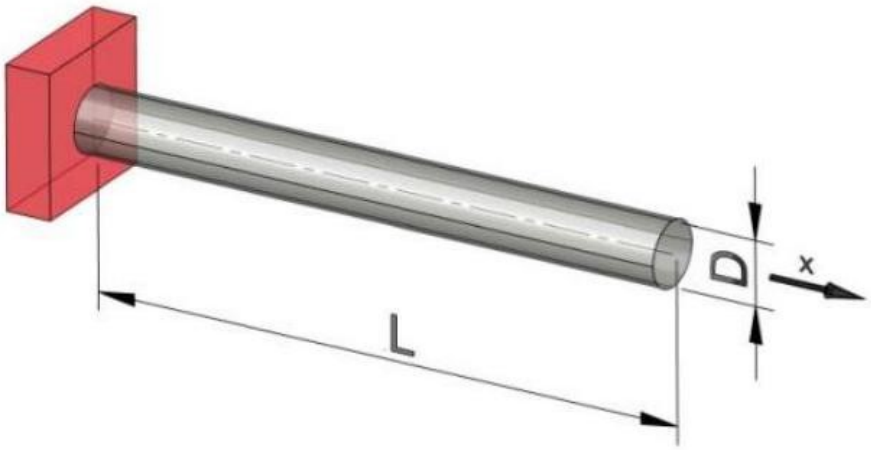

FIGURE 4.1: Rod with UDl along Axis with fixed at one end

L represents total rod length and c is a constant coefficient. The rod's stresses can be calculated as follows.

$$\sigma = \frac{\text{Load}}{\text{Area}} = \frac{c\,(L^2 - x^2)}{2A}$$

A denotes the cross-sectional area. The strain at particular point is given by:

$$\varepsilon = \frac{du}{dx}$$

By Hooke's law:

$$E\frac{du}{dx} = \frac{c\,(L^2 - x^2)}{2A}$$

or

$$AE\frac{du}{dx} = \frac{c}{2}\left(L^2 - x^2\right)$$

Hence,

$$AE\frac{d^2u}{dx^2} = -cx, \quad x \in (0, L)$$

The fixed rod's boundary conditions are

$$u(0) = 0, \left.\frac{du}{dx}\right|_{x=\mathrm{L}} = 0$$

By solving above equations we can get exact solution as:

$$u(x) = \frac{c}{6AE}\left(-x^3 + 3L^3x\right)$$

The block Diagram of the PINN architecture for the given Rod problem was shown below.

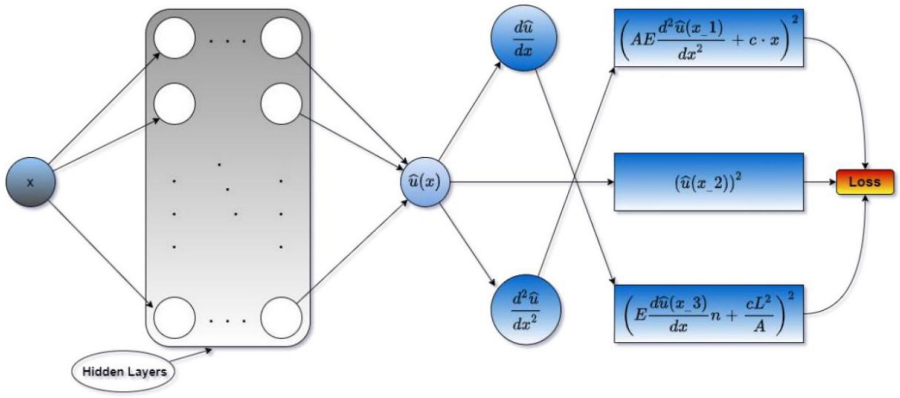

FIGURE 4.2: Rod's PINN Architecture

We have already covered the theory portion.In this section, we will show you some code samples that demonstrate how the PyTorch API actually functions to successfully train neural networks.

1. Import necessary modules and start timer

```
start = timer()
import torch.nn as pinn
import torch
import numpy as np
torch.manual_seed(42)
```

2. We set the input, output and hidden layer sizes for our neural network.

```
input_size = 1
hidden_size1 = 15
hidden_size2 = 15
output_size = 1
```

3. We create a neural network with 2 hidden layers using the Sequential function from PyTorch. We use the Tanh activation function for both hidden layers.

```
N2 = pinn.Sequential(
```

```
pinn.Linear(i_s, h_s1),
pinnn.Tanh(),
pinn.Linear(h_s1, h_s2),
pinn.Tanh(),
nn.Linear(h_s2, o_s)
)
```

4. We check if GPU is available and move the neural network to GPU if it is.

```
dvce = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
N2 = N2.to(dvce)
```

5. We define a function f(x) which we will use to compute the target function.

```
def f(x):
return -2*x
```

6. We define the loss function for our neural network. We compute the first four derivatives of the output with respect to the input using the torch.autograd.grad function. We then calculate the mean squared error of these derivatives and the function f(x) along with the output at x=0 and the derivative at x=1.

```
def loss(x):
x.requires_grad = True
y = N2(x)
dy_1 = torch.autograd.grad(y.sum(), x, create_graph=True)[0]
dy_2 = torch.autograd.grad(dy_1.sum(), x, create_graph=True)[0]
dy_3 = torch.autograd.grad(dy_2.sum(), x, create_graph=True)[0]
dy_4 = torch.autograd.grad(dy_3.sum(), x, create_graph=True)[0]
ans = torch.mean( (dy_2-f(x))**2 + (y[0, 0] - 0)**2 +(dy_1[-1, 0] - 0)**2 )
return ans
```

7. We use the Adam with learning rate of 0.01

```
optimizer = torch.optim.Adam(N2.parameters(), lr=0.01)
```

8. We create the input data for our neural network.

```
x = torch.linspace(0, 1, 100)[:, None]
```

9. We define a closure function which computes the loss and backpropagates the gradients.

```
def closure():
optimizer.zero_grad()
l = loss(x)
l.backward()
return l
```

10. We train our neural network for 500 epochs and print the cost function every 10 epochs.

```
epochs = 500
for i in range(epochs):
optimizer.step(closure)
if(i%10==0):
print('cost function:',loss(x).detach().numpy())
```

11. We create input data for plotting the output of our neural network and the true function.

    After applying the Adam algorithm, the numerical results are shown in Table 4.1

No.	Layers	epochs	Loss Mse	Time(sec)
1	$[5, 5]$	600	7.1E-5	12
2	$[10, 10]$	500	6.67E-5	11
3	$[5, 5, 5]$	500	3.997E-5	12
4	$[15, 15]$	500	2.941E-5	13

TABLE 4.1: Optimization parameters of Neural Network for the case of Rod

The Adam algorithm performs significantly better and more consistently. It is critical to ensure an equivalent number of nodes and collocation points used in finite element analysis, as in the case of PINN, for a fair comparison with classical computational methods.

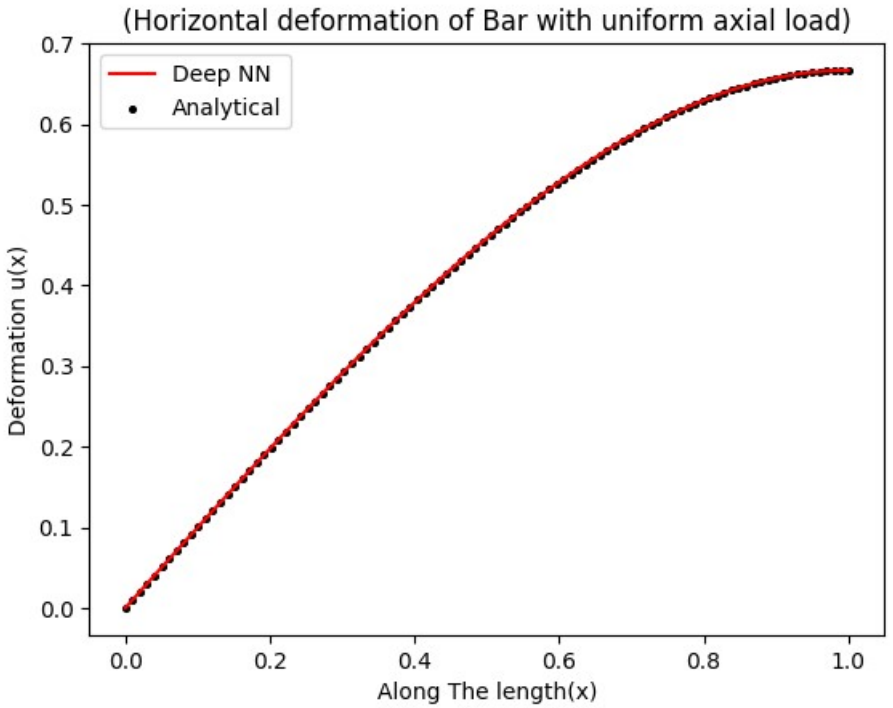

FIGURE 4.3: The true and PINN solutions of Rod

4.2 Simply Supported Beam with UDL

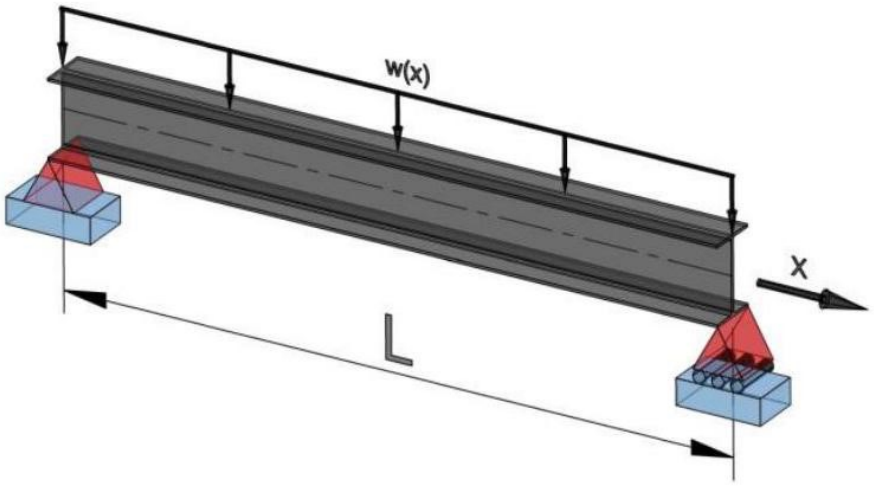

FIGURE 4.4: Simply Supported Beam with Uniformly Distrubuted Loading

The static theory gives us the following differential equation that describes the bending of the beam if a beam is subjected to a bending moment M, where

$M = -Px$ and P denotes the concentrated load.

$$\frac{d^2y}{dx^2} = \frac{M(x)}{EI}$$

we can convert above order 2 differential equation into fourth order differential equation by introducing the realtions of load,shear force,bending moment.

$$\frac{dM}{dx} = V, \quad \frac{dV}{dx} = -w$$

Therefore, as a result of above

$$\frac{d^4y}{dx^4} = -\frac{w(x)}{EI}$$

The following boundary conditions hold for a fully clamped beam of length L: $y(0) = 0$, $M(0) = 0$, $y(L) = 0$, and $M(L) = 0$. According to these conditions, the moment (M) and deflection (y) must both be zero at the beam's ends.

$$y(0) = 0, \quad y(L) = 0, \quad \frac{d^2y(0)}{dx^2} = 0, \quad \frac{d^2y(L)}{dx^2} = 0$$

We take into account a W-Beam with the following specifications to ensure realistic values: E = 200GPa (200×10^9Pa), I = 0.000038929334m^4, L = 2.7m, and w = 60KN/m.

The true solution of the problem is

$$y(x) = \frac{w}{24EI}\left(-x^4 + 2Lx^3 - L^3x\right)$$

The programming code used in this case is similar to the code in Problem 1, but some simplifications were made to make it easier to read, and changes had to be made to the loss function and boundary conditions.Below is how the modified loss function appears.

```
def f(x):
    return -w/(E*I)


def loss(x):
    x.requires_grad = True
    y = N2(x)
```

```
dy_1 = torch.autograd.grad(y.sum(), x, create_graph=True)[0]
dy_2 = torch.autograd.grad(dy_1.sum(), x, create_graph=True)[0]
dy_3 = torch.autograd.grad(dy_2.sum(), x, create_graph=True)[0]
dy_4 = torch.autograd.grad(dy_3.sum(), x, create_graph=True)[0]
return torch.mean( (dy_4-f(x))**2  + (y[0, 0] - 0)**2 +(y[-1, 0] - 0)**2
                          +(dy_2[0, 0] - 0)**2+(dy_2[-1, 0] - 0)**2 )
```

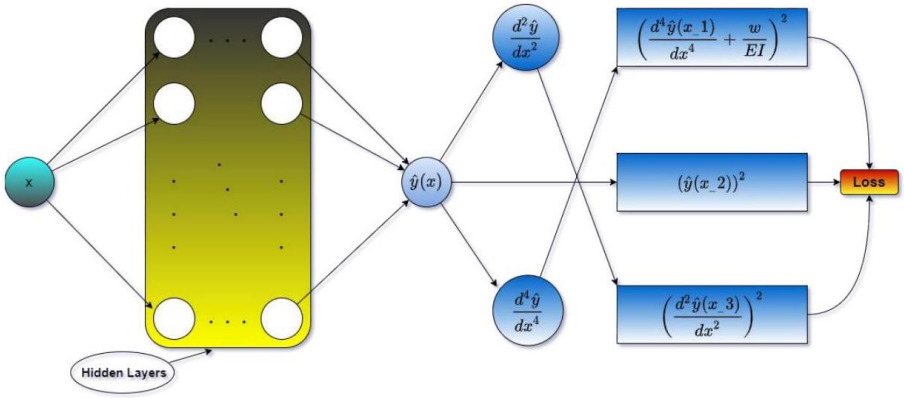

FIGURE 4.5: PINN's Architecture for Simply Supported with UDL case

The PINN's structure is shown in Figure 4.5, and the numerical outcomes are shown in below table 4.2.

No.	Layers	Epochs	Loss Mse	Time(Sec)
1	$[35, 35]$	1000	$3.13\text{E}-06$	21
2	$[50, 20]$	1500	$4.28\text{E}-7$	28
3	$[50, 20]$	2000	$3.4\text{E}-7$	31
4	$[20, 20, 20]$	1000	$3.87\text{E}-6$	29
5	$[30, 20, 20]$	1000	$2.53\text{E}-6$	30
6	$[40, 30, 30]$	1500	$2.2\text{E}-7$	33
7	$[40, 30, 30]$	2000	$8.7\text{E}-8$	44

TABLE 4.2: Optimized parameters for SSB UDL case

Below figure shows the comparison of True and DNN solutions of the SSB with UDL case

4.3 Cantilever Beam with UDL

In this situation, we look at the differential equation that describes how a cantilever beam bends.

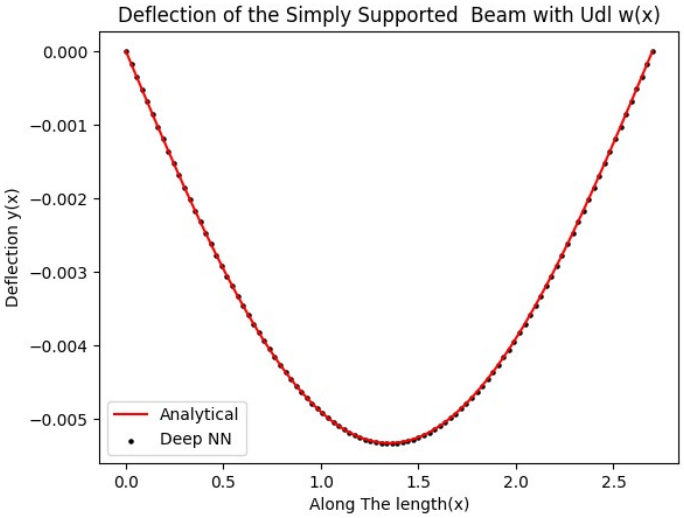

FIGURE 4.6: True solution and DNN solution for SSB with UDL

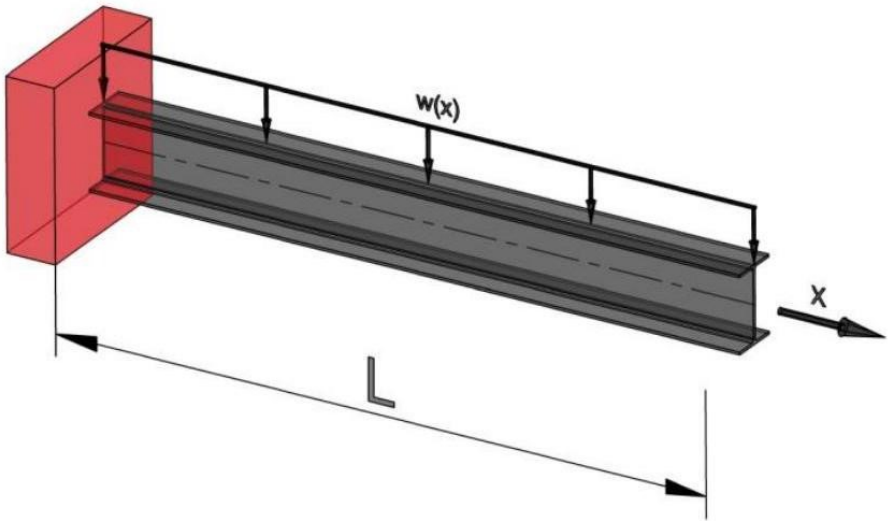

FIGURE 4.7: cantiliver Supported Beam with Uniformly Distrubuted Loading

The fixed edge's boundary conditions at $x = 0$ are :

$$y(0) = 0, \quad \frac{dy(0)}{dx} = 0$$

For the free end $x = L$, $M(L) = 0, V(L) = 0$.So,

$$\frac{d^2y(L)}{dx^2} = 0, \frac{d^3y(L)}{dx^3} = 0$$

The true solution $y(x)$ of the problem (4.12), (4.15) and (4.16) is given by

$$y(x) = \frac{w}{24EI}\left(-x^4 + 4Lx^3 - 6L^2x^2\right)$$

The code snippets for the above problems are the same, with the exception that the loss function and boundary conditions must change, as shown below.

```
def loss(x):
x.requires_grad = True
y = N2(x)
dy_1 = torch.autograd.grad(y.sum(), x, create_graph=True)[0]
dy_2 = torch.autograd.grad(dy_1.sum(), x, create_graph=True)[0]
dy_3 = torch.autograd.grad(dy_2.sum(), x, create_graph=True)[0]
dy_4 = torch.autograd.grad(dy_3.sum(), x, create_graph=True)[0]
ans=torch.mean( (dy_4-f(x))**2  + (y[0, 0] - 0)**2
+(dy_1[0, 0] - 0)**2+(dy_2[-1, 0] - 0)**2+(dy_3[-1, 0] - 0)**2 )
return ans
```

The PINN's structure for Cantiliver is shown in Figure 4.8, and the numerical outcomes are shown in below table 4.3.

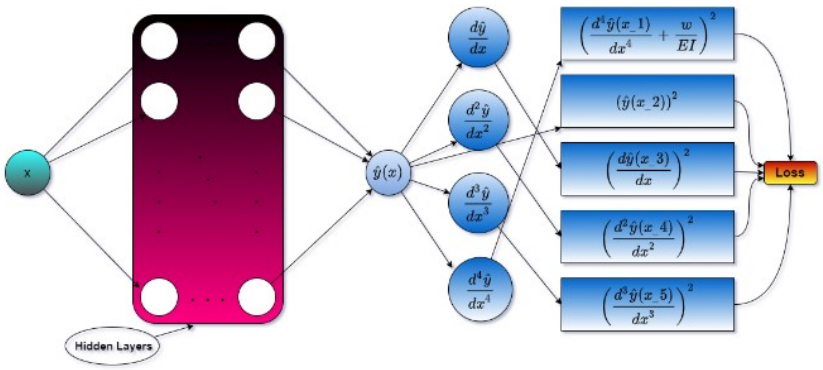

FIGURE 4.8: PINN of Cantiliver Beam with UDL

Below figure shows the comparison of True and DNN solutions of the Cantiliver with UDL case

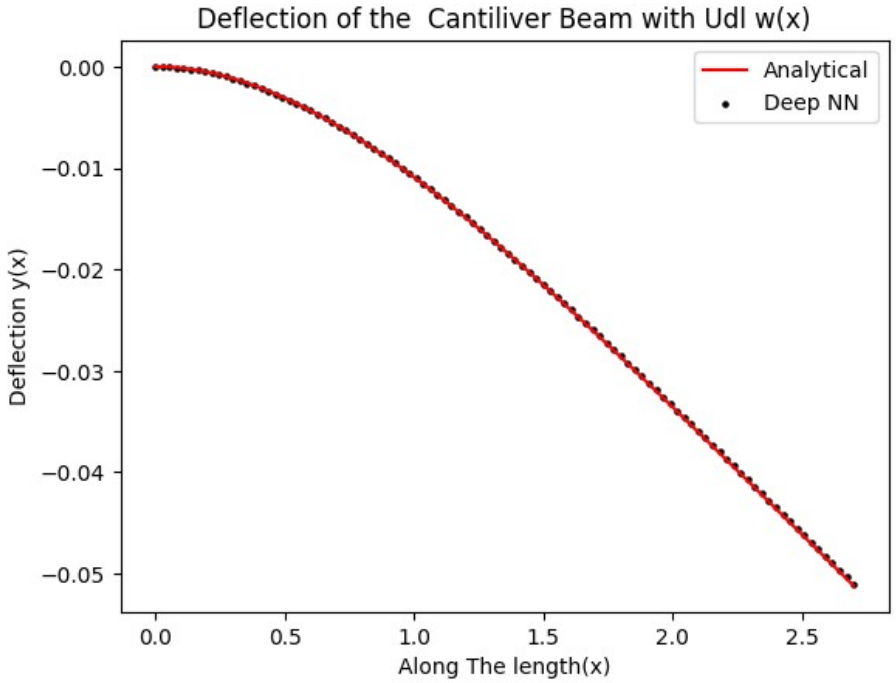

FIGURE 4.9: DNN solution vs True Solution

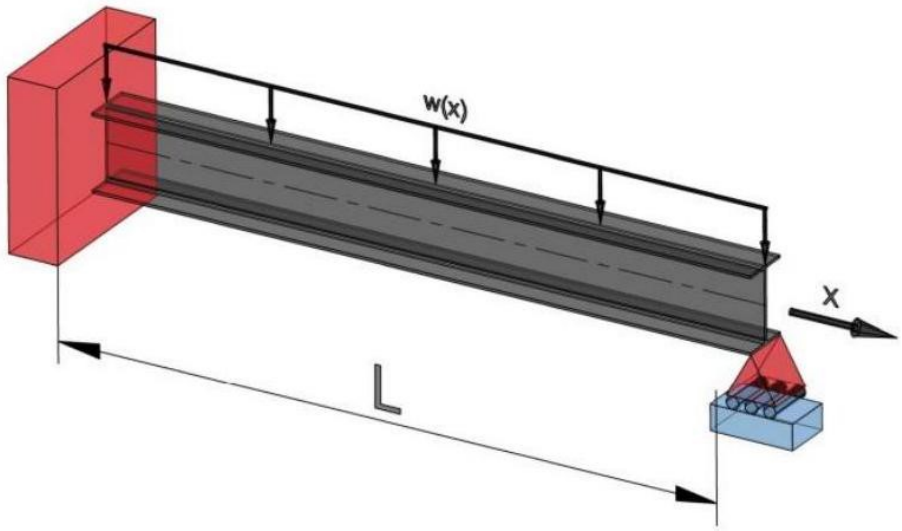

FIGURE 4.10: Propped Cantilever beam with Uniform Distributed Load

No of Layers	Epochs	Mean Loss	Time(sec)
[25,25]	1500	2.15E-07	24
[30,30]	2500	3.53E-07	17
[40,30]	2500	2.04E-08	17
[40,30,40]	2000	9.50E-08	22
[40,30,40]	2500	6.40E-08	32
[24,24,12]	2000	2.70E-08	28

TABLE 4.3: Optimized parameters of PINN for case of Cantilever with UDL

4.4 Propoed Cantiliver Beam with UDL

In this section we deals differential equation which describes how propped cantilever beam bends.Specifically, the boundary conditions are $y(0) = 0$ and $dy(0)/dx = 0$ at the left end, and $y(L) = 0$ and $M(L) = 0$ at the right end. The fourth order Differential equation which governs the problem, is the same as the differential equation discussed in Problems 2 and 3.

$$y(0) = 0, \quad \frac{dy(0)}{dx} = 0, y(L) = 0, \frac{d^2y(L)}{dx^2} = 0$$

By solving above the exact solution we can get is:

$$y(x) = \frac{w}{24EI}\left(-x^4 + \frac{5}{2}Lx^3 - \frac{3}{2}x^2\right)$$

Similar to Problem 3, the Python code used to implement the PINN method needs to be modified.The scripts that correspond to each are listed below.

```
def loss(x):
x.requires_grad = True
y = N2(x)
dy_1 = torch.autograd.grad(y.sum(), x, create_graph=True)[0]
dy_2 = torch.autograd.grad(dy_1.sum(), x, create_graph=True)[0]
dy_3 = torch.autograd.grad(dy_2.sum(), x, create_graph=True)[0]
dy_4 = torch.autograd.grad(dy_3.sum(), x, create_graph=True)[0]
ans=torch.mean( (dy_4-f(x))**2  + (y[0, 0] - 0)**2 +(dy_1[0, 0] - 0)**2
+(dy_2[-1, 0] - 0)**2+(y[-1, 0] - 0)**2 )
return ans
```

The PINN's structure for Cantiliver is shown in Figure 4.11, and the numerical outcomes are shown in below table 4.2.

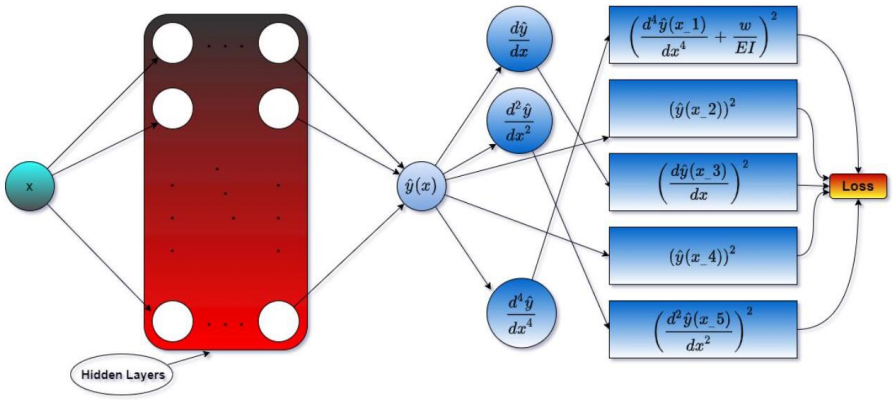

FIGURE 4.11: PINN Architecture of Propped Cantilever Beam

No of Layers	Epochs	Mean Loss	Time(sec)
[2,20]	1000	1.64E-06	27
[25,25]	2000	7.34E-07	54
[25,25,25]	2000	1.06E-06	26
[30,25,25]	2000	8.39E-07	26
[30,30,25]	2500	6.17E-07	34
[40,30,20]	3000	3.40E-08	35
[50,40,30]	3000	3.27E-08	41

TABLE 4.4: Optimized Parameters for Propped beam

Below figure 4.12 shows the comparison of True and DNN solutions of the Cantiliver with UDL case

4.5 Cantiliver Beam with Uniformly Varying Load

In this experiment, we employ a feedforward neural network to estimate the deflection of a cantilever beam under the influence of a uniformly varying load. The beam possesses characteristics such as length l, moment of inertia I, elastic modulus E, and is subject to a load $w(x)$. The deflection of the beam, denoted as $y(x)$, obeys a fourth-order differential equation:

$$\frac{d^4y}{dx^4} = -\frac{w}{EI}(1 - \frac{x}{l}) \tag{4.5.1}$$

with boundary conditions $y(0) = y'(0) = y''(l) = y'''(l) = 0$.

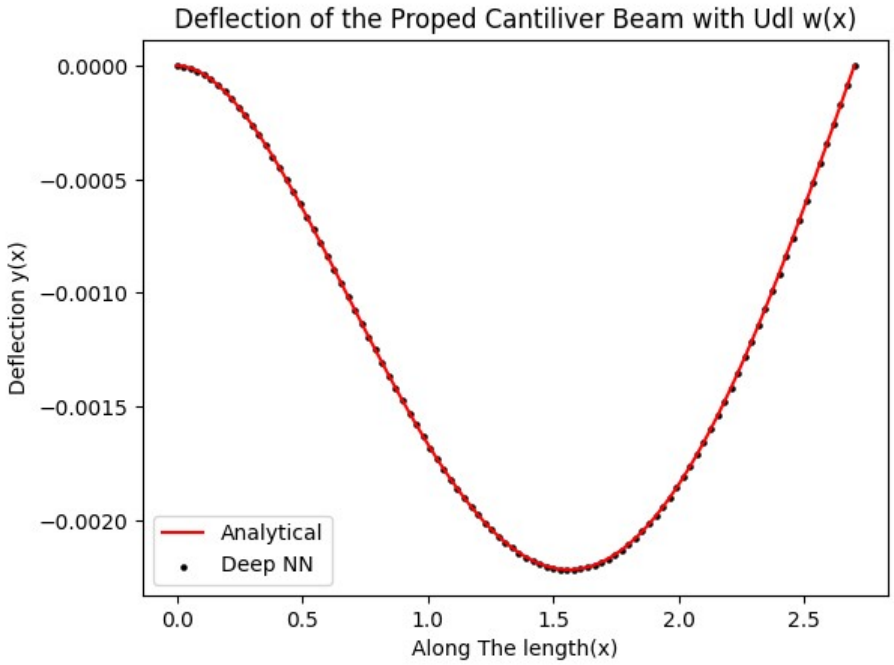

FIGURE 4.12: DNN vs True solution for Propped Beam with UDL

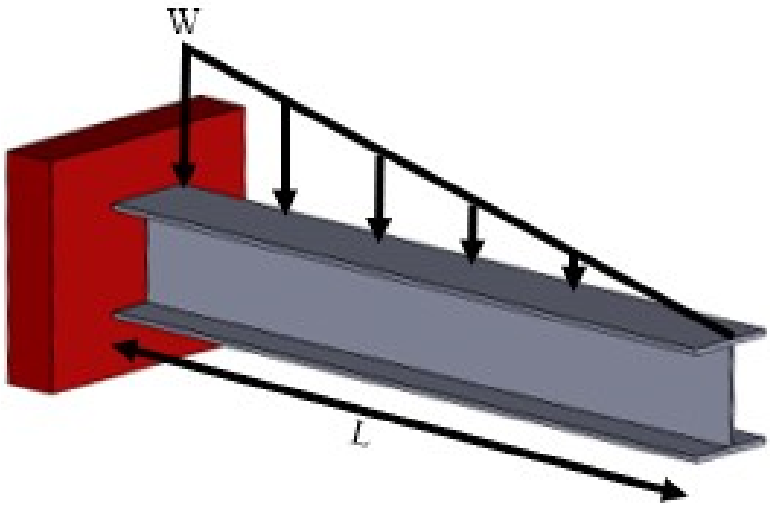

FIGURE 4.13: Cantilever Beam with Uniformly Varying Load

The goal of this experiment is to train a neural network to approximate $y(x)$ and compare the network's predictions to the analytical solution of the differential equation.The input to the network is the coordinate x along the beam, and the output is the deflection $y(x)$. The network is trained using the mean squared error loss function:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(\left(\frac{d^4y(x_i)}{dx_i^4} - f(x_i)\right)^2 + y(0)^2 + y'(0)^2 + y''(l)^2 + y'''(l)^2\right), \tag{4.5.2}$$

The analytical solution of the differential equation at point x_i is denoted as $y_t(x_i)$. The neural network's prediction at the same point is represented by $y(x_i)$, and n refers to the number of points utilized for training the network.

For optimization of the neural network, we employ the Adam optimizer with a learning rate of 0.01, and the training process consists of minimizing the loss function over given epochs.

Once the training is complete, we assess the model's performance by comparing its predictions to the analytical solution for the cantilever beam problem. The analytical solution provides the deflection of a cantilever beam subjected to a uniformly distributed load which was given by:

$$y_t(x) = -\frac{w}{120EI}(10l^3 - 10lx^2 + 5l^2x - x^3) \tag{4.5.3}$$

No of Layers	**Epochs**	**Mean Loss**	**Time(sec)**
[15,15]	1500	1.16E-09	9
[15,15]	2000	6.83E-10	12
[15,10,5]	2000	4.98E-09	18
[15,10,5]	3000	2.12E-09	27
[15,15,5]	3000	1.79E-09	28
[16,10,4]	3000	6.00E-08	25
[50,50]	3000	2.00E-10	23
[50,50]	2000	2.70E-11	14

TABLE 4.5: Optimized Parameters for Cantilever with UVL

We generate a set of input data points $x \in [0, l]$ and compare the neural network's predictions with the analytical solution.

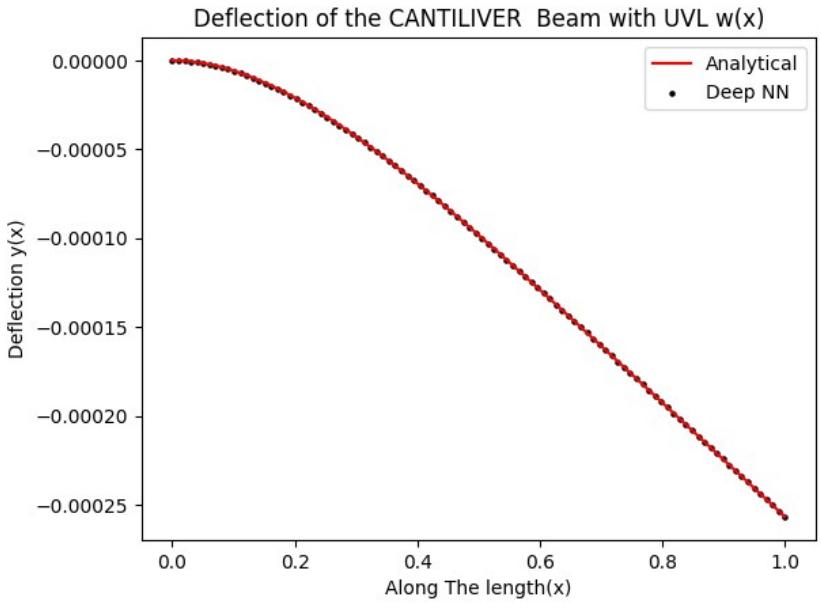

FIGURE 4.14: ANN solution vs True solution for Cantilever with UVL

4.6 Simply Supported Beam with Multipoint loading

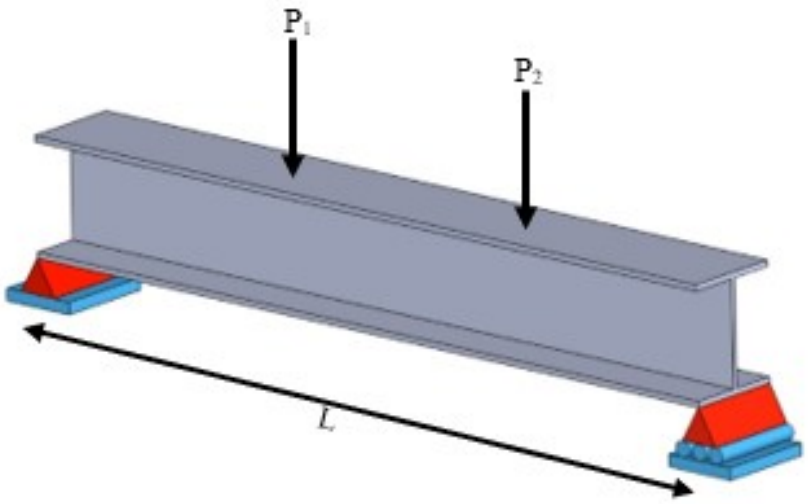

FIGURE 4.15: SSB with Multi Point Loading

In this problem, we look at the differential equation describing how a cantilever beam will behave when loaded from multiple points. For the fixed edge $x = 0$, the boundary conditions are

$$y(0) = 0, \quad y(l) = 0$$

We take into account a W-Beam with the following specifications to ensure realistic values: E = 200GPa (200×10^9Pa), I = 0.000085m^4, L = 6m, and P_1 = 60KN,P_2 = 120KN,a = 1m,b = 3m.When a load is subjected to multipoint loading, the governing differential equations remain the same as in the aforementioned problem. However, this situation presents an added complexity due to the existence of two moments that correspond to two differential equations, rendering the problem more challenging to solve. This is demonstrated below

$$\frac{d^2y}{dx^2} = stepFunction(x, a, b) \tag{4.6.1}$$

where

```
def stepFunction(x, a, b):
    M1=(60000*x)/(E*I)
    M2=(60000*x-48000*(x-1))/(E*I)
    M3=(60000*x-48000*(x-1)-40000*(x-3))/(E*I)
    # Initialize output tensor with zeros
    condition1 =  x < a
    condition2 = (x >= a) & (x < b)
    condition3 = (x >= b)
    region1 = torch.ones_like(x)*M1
    region2 = torch.ones_like(x)*M2
    region3 = torch.ones_like(x)*M3
    output = torch.where(condition1, region1,
    torch.where(condition2, region2, region3))
    return output
```

And the network is trained using the mean squared error loss function:

$$\mathcal{L} = (\frac{d^2y}{dx^2} - stepFunction(x, a, b))^2 + (y_0 - 0)^2 + (y_l - 0)^2 \tag{4.6.2}$$

After training, we evaluate the model by comparing its predictions with the analytical solution to the cantilever beam problem. The analytical solution for the deflection of a cantilever beam under a uniformly distributed load is given

by:

$$y_t(x) = y_1 * (1 - stepFunction(x, a, b)) + y_2 * stepFunction(x, a, b) \quad (4.6.3)$$

where

$y_1 = \frac{10^9}{E \cdot I}(-2.5x^4 + 30x^3 - 105x^2)$

$y_2 = \frac{10^9}{E \cdot I}(-5x^4 + 40x^3 - 120x^2 + 10x - 2.5)$

No of Layers	Epochs	Mean Loss	Time(sec)
[150,100,50,25]	3000	1.00E-08	67
[150,100,50,25]	3000	1.80E-08	60
[60,40,50,25]	3000	1.48E-08	42

TABLE 4.6: Optimized Parameters for SSB with Multi Point Loading condition

We generate a set of input data points $x \in [0, l]$ and compare the neural network's predictions with the analytical solution.

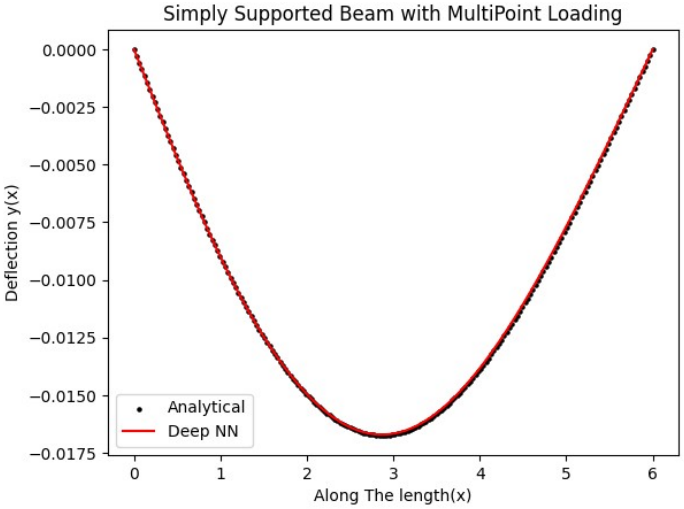

FIGURE 4.16: DNN vs True solution of simply supported Beam with Multi point loading

4.7 Cantilever Beam with Multi UDL

Solving a differential equation that describes how a cantilever beam responds to numerous uniformly distributed loads is the task at hand. The challenge comes from the right side of the equation's discontinuity, which is brought on by variations in the moments between various sections along the length of the

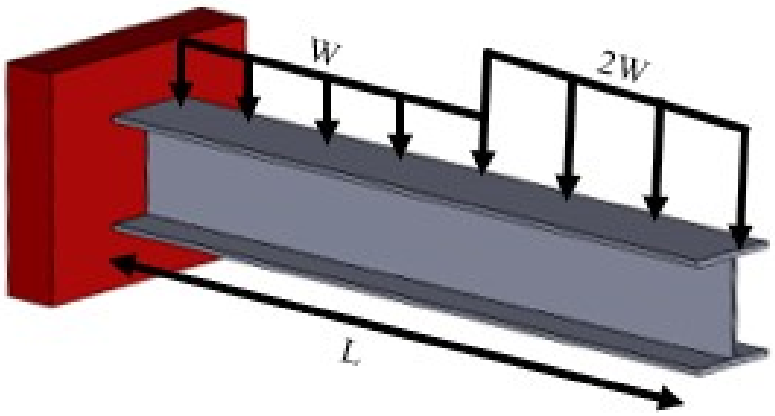

FIGURE 4.17: Cantiliver Beam with Multi UDL conditions

beam. While there are similarities between this problem and the scenario we previously discussed, it also involves a fourth-order differential equation and changes to the boundary conditions. Following is the Governing Differential:

$$\frac{d^4y}{dx^4} = stepFunction(x, a, b) \tag{4.7.1}$$

The fixed edge's boundary conditions are: $x = 0$

$$y(0) = 0, \quad \frac{dy(0)}{dx} = 0$$

It is accurate to say that $M(L) = 0, V(L) = 0$ for the free edge $x = L$. Therefore,

$$\frac{d^2y(L)}{dx^2} = 0, \frac{d^3y(L)}{dx^3} = 0$$

We take into account a W-Beam with the following specifications to ensure realistic values: E = 200GPa (200×10^9Pa), I = 0.000085m^4, L = 2m, and P_1 = 60KN/m,P_2 = 120KN/m,a = 1m,b = 1m The coordinate x along the beam serves as the network's input, and the deflection $y(x)$ serves as its output. To enhance the network's performance during training, we use the mean squared error loss function.

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(\left(\frac{d^4y(x_i)}{dx_i^4} - stepFunction(x, a, b)\right)^2 + y(0)^2 + y'(0)^2 + y''(l)^2 + y'''(l)^2\right), \tag{4.7.2}$$

where the neural network's prediction at point x_i is $y_t(x_i)$, the analytical solution to the differential equation at point x_i is $y(x_i)$, and the number of points used to train the network was n.

where $stepFunction(x, a, b)$ follows as:

```
def step_function(x, a, b):
M1=-(w1)/(E*I)
M2=-(w2)/(E*I)
# Initialize output tensor with zeros
condition1 =  (x < a)
condition2 = (x >= a)
region1 = torch.ones_like(x)*M1
region2 = torch.ones_like(x)*M2
output = torch.where(condition1, region1,region2)
return output
```

After training, we assess the model by contrasting its predictions with the cantilever beam problem's analytical solution. The following equation provides the analytical solution for a cantilever beam's deflection under a load that is distributed uniformly:

$$y_t(x) = y_1 * (1 - stepFunction(x, a, b)) + y_2 * stepFunction(x, a, b) \quad (4.7.3)$$

where

$y_1 = -\frac{10^9}{EI}(-2.5x^4 + 30x^3 - 105x^2)$

$y_2 = -\frac{10^9}{EI}(-5x^4 + 40x^3 - 120x^2 + 10x - 2.5)$

No of Layers	Epochs	Mean Loss	Time(sec)
[80]	4000	2.13E-07	26
[60,50]	4000	2.00E-07	36
[60,50,50]	3000	1.80E-07	45
[50,30,20]	3000	4.30E-07	50
[40,30,20,10]	4000	3.00E-07	67
[100,75,50,25]	4000	2.80E-07	90

TABLE 4.7: Optimized Parameters of Cantiliver with multiple loading conditions

We generate a set of input data points $x \in [0, l]$ and compare the neural network's predictions with the analytical solution.

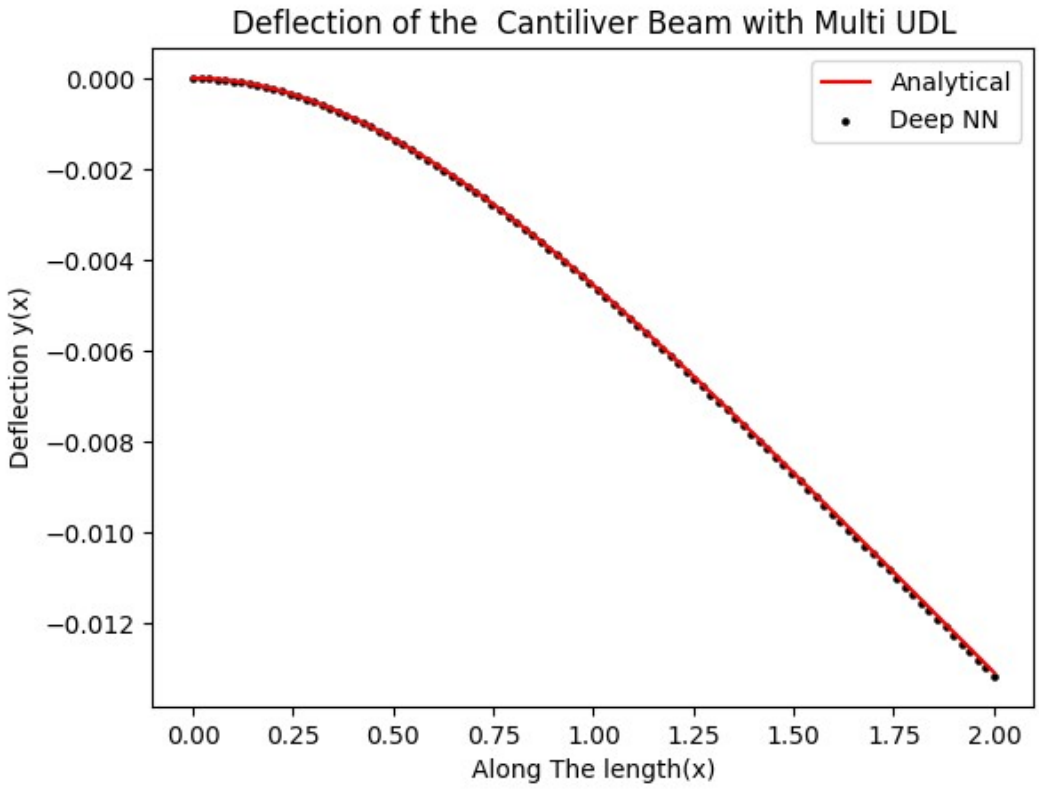

FIGURE 4.18: DNN vs True Solution of Cantilever Multiple Udl conditions

Chapter 5

Conclusions and Discussions

Using the Adam optimisation algorithm and the tanh activation function, this discussion focuses on using Physics Informed Neural Networks (PINN) to solve structural problems, specifically beam problems. The study examines the use of PINN to address a variety of loading conditions, such as multiple uniformly distributed load (UDL) scenarios and multiple concentric point loading.

Step functions were used to precisely depict the loading conditions within the PINN framework to handle the complexity of multi concentric point loading. Similar to how L2 differential equations were used to model multi-centric loading conditions, L4 differential equations were used to model UDL conditions. Step functions were essential for successfully integrating these conditions into the PINN framework.

For accurate results, choosing the right numerical accuracies was essential. To find the most appropriate numerical accuracies, a combination of trial and error and knowledge from prior literature was used, striking a balance between computational efficiency and accuracy. The maximum number of training epochs was set at 4000 to avoid overfitting and potential mistakes when applied to new problems.

The approach described in this study has an intriguing feature: solving the aforementioned problems only requires a small amount of computation time. The proposed method's effectiveness in comparison to traditional numerical solutions and Finite Element Analysis (FEA) techniques is demonstrated by the maximum solving time of one minute. Due to its effectiveness, PINN stands out as a strong contender for tasks requiring structural problem-solving.

Additionally, the study specifically chose issues with analytical solutions to aid in thorough visualisation and comprehension. The accuracy and dependability of the PINN-based approach were successfully validated by comparing the results with the analytical solutions. This validation expands the proposed methodology's applicability beyond issues with precise analytical solutions, making it applicable for a wider range of structural engineering difficulties.

Chapter 6

Future Work

Although this thesis has made significant strides in using PINN to solve structural problems, there are still a number of areas that can be explored and improved. Here, we highlight potential research directions for the future:

(a) **Enhancing accuracy and convergence**: Researching various optimisation methods and activation functions may result in further increases in convergence rates and accuracy. The PINN framework's performance might be improved by investigating adaptive learning rates and cutting-edge optimisation methods.

(b) **Handling complex geometries**: The methodology's applicability would be increased if it were extended to address structural issues involving complex geometries, like curved beams or asymmetrical shapes. This might entail creating methods for dealing with boundary conditions and non-linear geometries.

(c) **Uncertainty quantification**: A more thorough understanding of the dependability and robustness of the solutions would be possible with the incorporation of uncertainty quantification techniques into the PINN framework. Uncertainty quantification in the context of PINN could be facilitated by investigating Bayesian neural networks or probabilistic formulations.

(d) **Integration with experimental data**: The PINN framework could be extended to incorporate experimental data, enabling data-driven structural design and analysis. The precision and dependability of the predictions could be increased by combining numerical simulations with

experimental findings using methods like data assimilation or model calibration.

(e) **Parallelization and scalability**: The effectiveness and performance of the PINN framework might be further improved by looking into parallel computing methods and scalable architectures. Large-scale structural problems may be computed faster using GPU acceleration or distributed computing resources.

Chapter 7

References

(a) Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations.

(b) Hanke, M., & Obersteiner, M. (1994). Solving Ordinary Differential Equations with Neural Networks.

(c) Long, Z., Lu, Y., & Dong, B. (2020). DeepXDE: A Deep Learning Library for Solving Differential Equations.

(d) Chkrebtii, O., Gray, A. G., & Keutzer, K. (2018). Learning to Solve Partial Differential Equations Using Deep Learning.

(e) Zhang, L., Wang, D., & Zhao, L. (2018). A Physics-Informed Neural Network for Multiphase Flow in Porous Media.

(f) Hirsch, S., Potter, M., & Ortiz, M. (2017). Learning to Solve Differential Equations in Continuum Mechanics by Example.

(g) Lei, H., Liu, Y., & Karniadakis, G. E. (2020). A Hybrid Method of Physics-Informed Neural Networks and Gaussian Processes for Inverse Problems.

(h) Wong, T.-K. L., Xu, J., & Yang, X. (2019). Solving High-Dimensional Partial Differential Equations Using Deep Learning.

(i) Zhang, D., Yu, B., & Shu, C.-W. (2020). A Machine Learning Framework for Solving High-Dimensional Partial Differential Equations.

(j) Karniadakis, G. E. (2020). Physics-Informed Deep Learning for Fluid Dynamics: A Review.