|  |  |
| --- | --- |
| ***E:\IIETA logo.jpg*** | **International Journal of Heat and Technology**  Vol., No., Month, Year, pp. \*\*-\*\*  Journal homepage: http://iieta.org/journals/ijht |

|  |  |  |
| --- | --- | --- |
| **Vibration Analysis of Tapered Truncated cantilever beams using Deep Learning** | | |
| Naman Yadav\*, Asim Tewari | | |
| Department of Mechanical Engineering, IIT Bombay  Corresponding Author Email: 160100012@iitb.ac.in | | |
| https://doi.org/10.18280/ijht.xxxxxx |  | **ABSTRACT** |
|  |  |  |
| **Received:**  **Accepted:** |  | This work relates to the use of deep learning in vibration analysis of tapered truncated cantilever beams. In this paper, an attempt was made to train a deep learning neural network to predict the natural frequency parameters and node locations of first three mode shapes for tapered truncated cantilever beams. Beam geometries that were considered to generate the data used to train the neural network belong to a quite general class. After tuning some hyperparameters, maximum percent deviation of all the outputs was brought down to less than 0.1 percent. Reported results were obtained by evaluating trained neural networks on a separate test dataset different from both training and validation sets. Finally, cross-evaluation of trained models was done, to see its extrapolating capabilities over unseen data ranges. |
| ***Keywords:***  *Cantilever Beams, Deep Learning, free vibrations, mode shapes, natural frequencies.* |  |

# Introduction

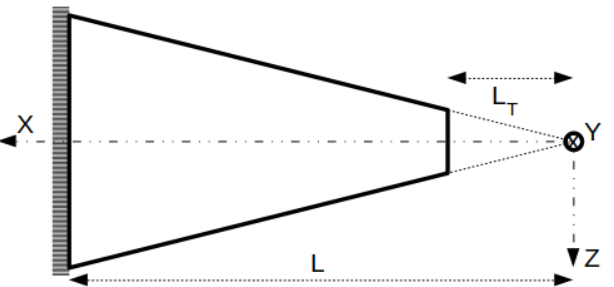
Cantilever beam structure has numerous applications in all fields of engineering; thus estimation of natural frequencies and node locations is crucial [1-2]. Among several other applications, cantilever beams are used in AFM (Atomic Force Microscopy) [3], microfabricated gas sensors [4], etc. Various papers have already been published regarding the vibration analysis of a certain class of truncated tapered beams using Bessel functions [5-9]. Some work has already been done to predict natural frequencies of cantilever beams with uniform cross-section using artificial neural networks [10], but these neural networks predict natural frequencies of cantilever beams of specific material. We have attempted to generalize this approach to a more general class of cantilever beams.

In this paper artificial neural networks were used to predict the natural frequency parameters and node locations for first three modes of truncated cantilever beams with varying cross-section properties. The main objective of the following work is to use the theoretical solutions to generate training data and then train a deep learning model using that data to predict the natural frequency parameters and node locations of first three mode shapes over seen and unseen data ranges. Analysis of cantilever beams was done with larger end clamped and smaller end free.

This work deals with the class of beams satisfying the variation in sectional properties as following:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

where, AC and IC are the cross-sectional area at the clamped end and moment of inertia about flexural axis at the clamped end, respectively, and δ=x/L ϵ [δ0, 1], δ0=LT/L, where LT is the length of truncated portion, L is the total length of beam without truncation (Figure 1), η ϵ [0, 10] is a real number specifying the variation in the area of cross-section of the beam.



**Figure 1.** Cantilever beam structure

1. THEORY

For the cantilever beam subjected to free vibrations the equation of motion is given by the Euler-Bernoulli equation,

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

where, y(x, t) is deflection at position x in yz-plane perpendicular to flexural axis at time t, E is the Young’s modulus of the material, I(x) is the moment of inertia about flexural axis at position x, is density of the material and A(x) is the area of cross section at position x.

Since variable separation is applicable here, therefore, we can assume solution of Eq. (1) in the following form,

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

General solution for in Eq. (3) can be given as,

|  |  |  |
| --- | --- | --- |
|  |  | (4) |
|  |  |  |

where, Jη and Yη are ηth order Bessel Functions of first and second kind, respectively. Iη and Kη are modified ηth order Bessel Function of first and second kind, respectively.

Following are the Boundary conditions for cantilever beam shown in Figure 1,

|  |  |  |
| --- | --- | --- |
|  |  | (5) |
|  |  | (6) |

After applying boundary conditions (Eq. (5-6)) on Eq. (4), following set of linear equations was obtained,

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | (7) |
|  |  | | (8) |
|  |  | | (9) |
|  |  | | (10) |
|  | |  |  |

Now, for this linear system of equations (Eq. (7-10)) to have non-trivial solution following condition should satisfy,

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

After solving Eq. (11), we can get values of β = βr, where, r is the mode number. From which, we can compute and using following relations,

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

Finally, mode shapes can be obtained using Eq. (12) as follows,

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

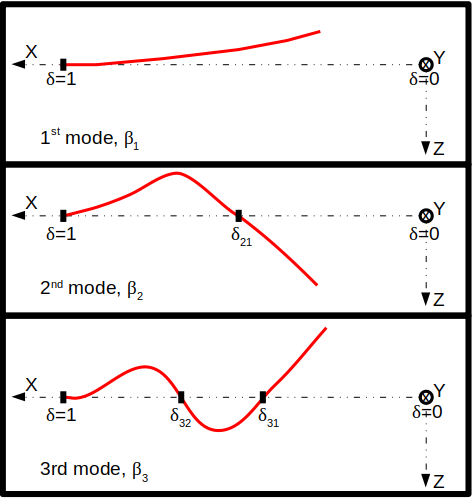
1. INPUT AND OUTPUT VARIABLES

**4.1 Input variables to the neural network**

**η,** **δ0** - where, **η** is exponent defining the variation of Area of cross-section of the cantilever along x-axis and **δ0** is fraction of truncated portion of cantilever.

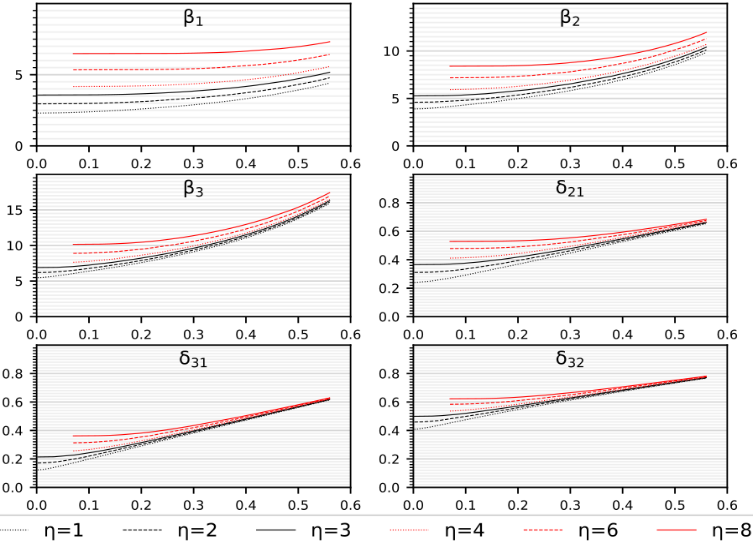
**4.2 Output variables of the neural network**

**β1, β2, β3**, **δ21, δ31, δ32 –** where **β1**, **β2**, **β3** are parameters defining first three natural frequencies and **δ21**, **δ31**, **δ32** are non-dimensionalized numbers defining node locations of first 3 mode shapes (Figure 2).

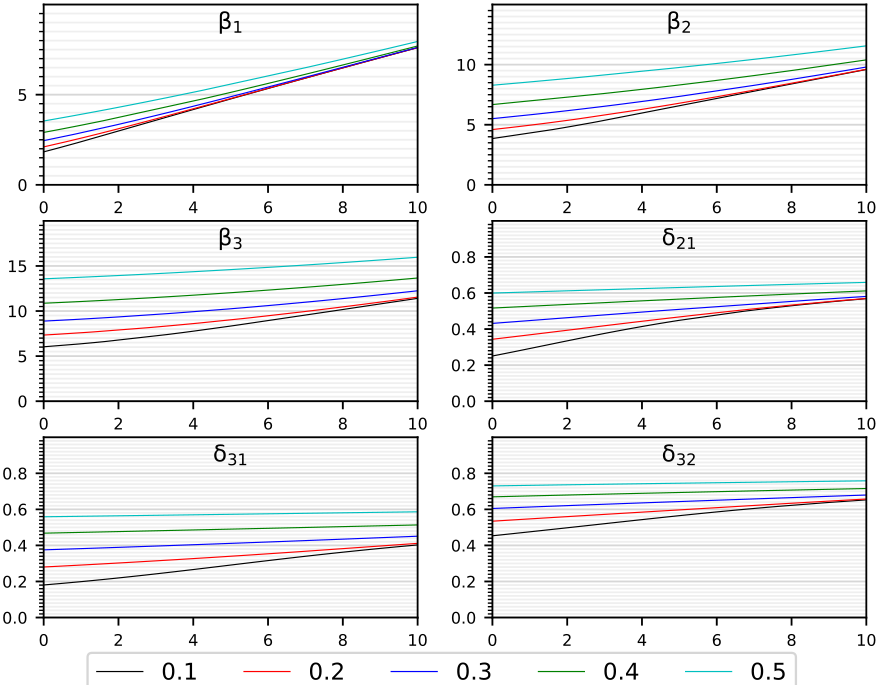


**Figure 2.** Pictorial representation of output variables.

Based on the theory, following plots (Figures 3 and 4) were generated using MATLAB [11] showing theoretical variation in output variables with respect to input variables. For higher values of **η** and very low values **δ0** our solver [12] was not able to solve the equations with enough accuracy, therefore, those values were skipped as can be seen in Figure 3.

****

(a)Output variables vs .

****

(b) Output variables vs .

**Figure 3.** Actual variation of output variables.

# DATA GENERATION

Data was generated for 1,001 values of **η** є [0, 10] and for each **η**, 15000 data points were collected by varying **δ0** between [2e-3, 0.56] for **η** є [0, 3] and [7e-2, 0.56] for **η** є (3, 10], reason for this split being insufficient accuracy in the frequency parameters calculated by the algorithm for small values of **δ0** and large values of **η**.

**4.1 Construction of training and validation datasets**

Since generated data was huge, therefore it was shuffled and few datapoints were picked and these were divided into two sets, exact dimension of these sets is explained later in the following sections. As mentioned earlier, since our algorithm was not able to calculate the frequency parameters for some corner values of **η** and **δ0** accurately, therefore overall distribution of data points was as following:

(a) For **η** є [0, 3], **δ0** є [2e-3, 0.56], 301\*15000 datapoints.

(b) For **η** є [3, 10], **δ0** є [7e-2, 0.56], 701\*15000 datapoints.

(c) For **η** є [0, 10], data for above two cases was merged as described below:

+ **η** є [0, 3], **δ0** є [2e-3, 0.56], 301\*15000=4,515,000.

+ **η** є (3, 10], **δ0** є [7e-2, 0.56], 700\*15000=10,500,000.

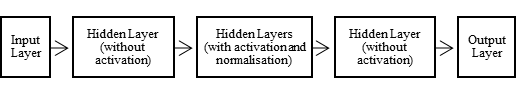
**4.2 Construction of test dataset**

Apart from the above datapoints a separate Test dataset of 50\*100=5,000 examples was created for each interval (**η** є [0,3] and **η** є [3, 10]), for the final evaluation of the models that performed best on the validation set. These datapoints were generated by sampling 50 random values of **η** from each interval and for each value of **η**, 100 values of **δ0** were randomly chosen. For data generation MATLAB was used.

# DEEP LEARNING MODEL

# 5.1 Neural network architecture

Structure of the neural network is defined as following: [Input Layer, 2 units] 🡪 [Fully connected Hidden layer without activation function, with 6\*N2 units] 🡪 L1\*[(Fully connected Hidden layer 🡪 Layer normalisation 🡪 activation function), all three layers having N1 units each] 🡪 [Fully connected Hidden layer without activation function, with 6\*N2 units] 🡪 [Output layer, 6 units]. Figure 4 gives pictorial representation of the neural network.



**Figure 4.** Neural network

* 1. **Layer Normalisation** [13]

Output of normalisation layer is given by :

|  |  |  |
| --- | --- | --- |
|  |  | (14) |
|  |  |  |

where, is the value at ith hidden unit of lth layer and H is the number of hidden units in a layer. All the hidden units in a layer share the same normalization terms and , but different training datapoints have different normalization terms. γ and β are learnable parameters.

* 1. **Activation function** [14] – PReLU

|  |  |  |
| --- | --- | --- |
|  | PReLU() = *max*(0,)+a0∗*min*(0,) | (15) |

where, a0 is a learnable parameter.

* 1. **Loss function** - Mean squared percent error.

|  |  |  |
| --- | --- | --- |
|  | PReLU() = *max*(0,)+a0∗*min*(0,) | (16) |

where, N is the number training examples, actij is actual value of ith output of jth training example and predij is predicted value of ith output of jth training example.

* 1. **Optimizer**

Adam optimizer[15] was used for minimising the value of loss function, with parameters beta1 = 0.9 and beta2 = 0.999, along with mini-batch gradient descent.

Choice of activation function, loss function and optimizer was made after training and evaluating some simple neural networks on small dataset. It was observed that PReLU, mean squared percent error and Adam gave best results. Layer normalisation did not significantly improve the accuracy; however, it still gave positive results.

# Neural network training parameters

Models were trained using Pytorch [16] on NVIDIA GeForce GTX 1060 6GB GPU. Due to inconsistency in the range of input variable **δ0**, different models were tuned for two different intervals of **η**:

1. **η** є [0, 3], **δ0** є [2e-3, 0.56].
2. **η** є [3, 10], **δ0** є [7e-2, 0.56].

Various models were trained on each of the above cases by tuning following hyperparameters:

**Table 1.** Neural network hyperparameters

|  |  |
| --- | --- |
| **Hyperparameter** | **Description** |
| L1 | Number of hidden layers with layer normalisation and activation |
| N1 | Number of units in the hidden layers with activation and normalisation units |
| N2 | Parameter defining number of units in remaining hidden layers |
| Initial learning rate | Initial learning rate |
| Update period | Number of iterations after which the learning rate will be updated |
| Update factor | Factor by which to divide learning rate on each update. |

For tuning the hyperparameters mentioned in Table 1, several models were trained by randomly sampling the values of each hyperparameter from a certain interval best suited for that hyperparameter. Sampling interval of all hyperparameters was narrowed down over subsequent iterations, around the region giving best performance for finer tuning.

Dataset generated earlier was shuffled and then datapoints were randomly selected from it, to generate training and validation datasets. Models were trained over training dataset and then, validation dataset was used to pick the best model. Finally, the model that performed best on the validation dataset of 1,000 datapoints was evaluated on the test dataset and results are tabulated in the next sections. After training few simple models, following values of learning rate parameters, mini-batch size number of mini-batches, number of iterations, etc. worked best:

**Table 2.** DL model training parameters

|  |  |  |
| --- | --- | --- |
|  | **Case 1** | **Case 2** |
| # Training datapoints | 2\*211 | 20\*211 |
| # Iterations | 30,000 | 30,000 |
| Initial LR | 0.01 | 0.01 |
| Update period | 5,000 | 5,000 |
| Update factor | 5 | 5 |

Note**:** # Training examples = n\*2m,where n = #mini-batches, m = mini-batch size. Iteration over n mini-batches, i.e. whole training set, was considered as one iteration.

Only difference between Case 1 and Case 2 is that of the number of training points, all other training parameters were kept same for both the cases as can be seen in Table 2.

# training results

Out of several models trained and evaluated Neural networks with architecture similar to what mentioned below performed best depending on the size of training dataset:

**Case 1**: **L1** = 3, **N1** = 173, **N2** = 3.

**Case 2**: **L1** = 7, **N1** = 207, **N2** = 3.

Performance of the neural networks mentioned above obtained after evaluation on the Test data set is tabulated below for two different intervals of **η.** It can be easily observed from following results that a bigger network can be trained using even more training data to further reduce errors.

**Table 3.** Mean and Max. percent deviations

1. **η** є [0, 3], **δ0** є [2e-3, 0.56]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Output** | **Mean Percent dev.** | | **Max. Percent dev.** | |
| **Case 1** | **Case 2** | **Case 1** | **Case 2** |
| β1 | 0.022 | 0.011 | 0.115 | 0.106 |
| β2 | 0.020 | 0.008 | 0.110 | 0.071 |
| β3 | 0.021 | 0.009 | 0.111 | 0.052 |
| δ21 | 0.016 | 0.012 | 0.172 | 0.101 |
| δ31 | 0.024 | 0.013 | 0.317 | 0.255 |
| δ32 | 0.012 | 0.007 | 0.159 | 0.083 |

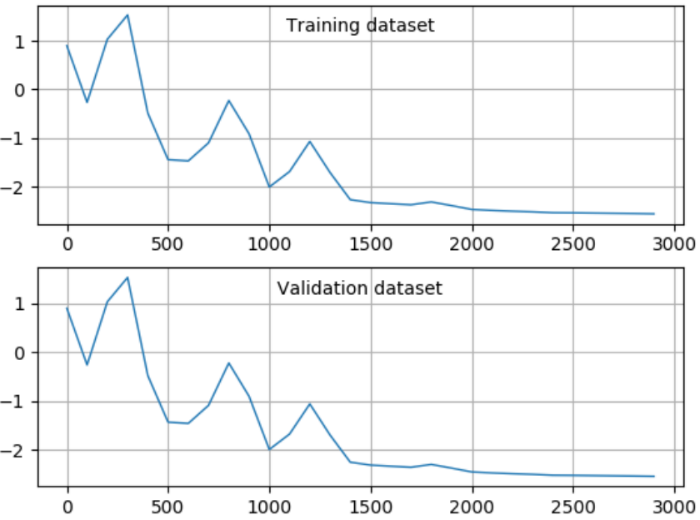
1. **η** є [3, 10], **δ0** є [7e-2, 0.56]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Output** | **Mean Percent dev.** | | **Max. Percent dev.** | |
| **Case 1** | **Case 2** | **Case 1** | **Case 2** |
| β1 | 0.019 | 0.011 | 0.130 | 0.079 |
| β2 | 0.018 | 0.010 | 0.130 | 0.046 |
| β3 | 0.019 | 0.010 | 0.123 | 0.054 |
| δ21 | 0.013 | 0.009 | 0.090 | 0.050 |
| δ31 | 0.019 | 0.009 | 0.169 | 0.051 |
| δ32 | 0.010 | 0.005 | 0.055 | 0.037 |

Although the size of entire dataset that was generated was enormous, but we used only few of the generated datapoints. It can be seen in Table 3 that even with very few data points our neural network is able to predict frequency parameters and node locations of a certain class of tapered truncated cantilever beams within 0.1 percent deviation from actual values. Since, predictions of our neural network are highly accurate, therefore we can also use it to predict for the input values that are near but outside the seen data range. Extrapolating capabilities of the network are explored in furthher sections.

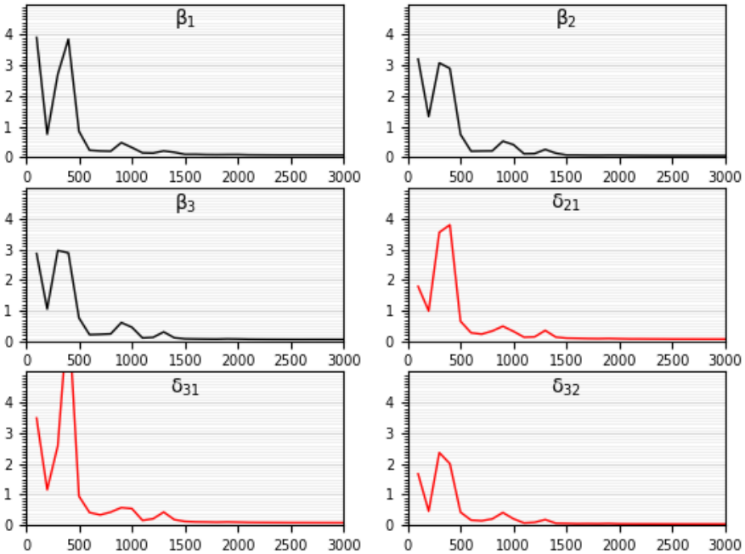
Also, from the results tabulated in Tables 2-3 it can be inferred that larger and more complex neural network improves accuracy in predications in both intervals, but the margin of improvement in accuracy when trained with bigger network and more data, is slightly more significant for smaller values of **η**, this observation goes in sync with the fact thatmapping function from input to output is more complex for smaller values of **η** compared to larger values.

Models were finally trained and evaluated over the entire range of **η** є [0, 10]. As per expectations similar trend in percent deviations was observed as shown in Table 3. Data for the two ranges was merged and used for training and evaluation. Training and validation plots for the model trained over larger number of datapoints, i.e. case 2 are shown below. For validation 10,000 datapoints were used and only 3,000 iterations were done using update period for learning rate to be 500, all other training parameters were same. Obtained results are shown in Figures 5 and 6.

****

**Figure 5.** Loss on log10 scale vs no. of iterations.

In Figure 6, Ninety-Ninth percentile of the percent deviations of the predictions over the validation dataset are plotted against the number of iterations for each output variable of the network.

****

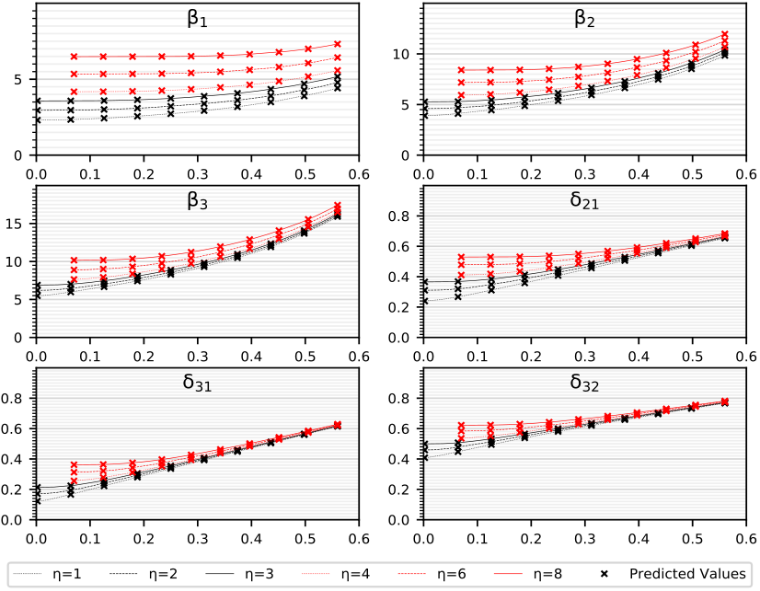
**Figure 6.** Validation set- 99th percentile percent deviation.

# Graphical comparison between actual and predicted values

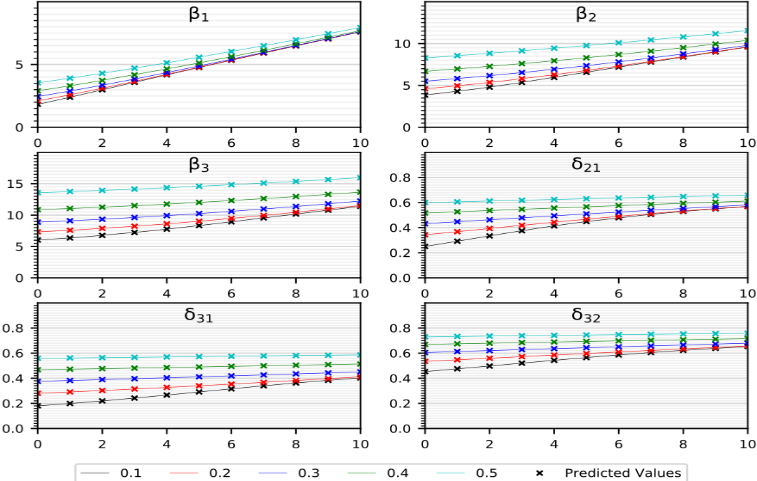
Following plots are for model with parameters mentioned in Table 4, trained over entire range of **η** є [0, 10]. As mentioned in previous sections for large **η** small **δ0** data was not generated, thus for **η** є (3, 10], **δ0** є [7e-2, 0.56] whereas for **η** є [0, 3], **δ0** є [2e-3, 0.56].

**Table 4.** Training parameters

|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| L1 | 7 |
| N1 | 207 |
| N2 | 3 |
| # Training datapoints | 20\*211 |
| # Validation datapoints | 10,000 |
| # Iterations | 3,000 |
| Initial LR | 0.01 |
| Update period | 500 |
| Update factor | 5 |



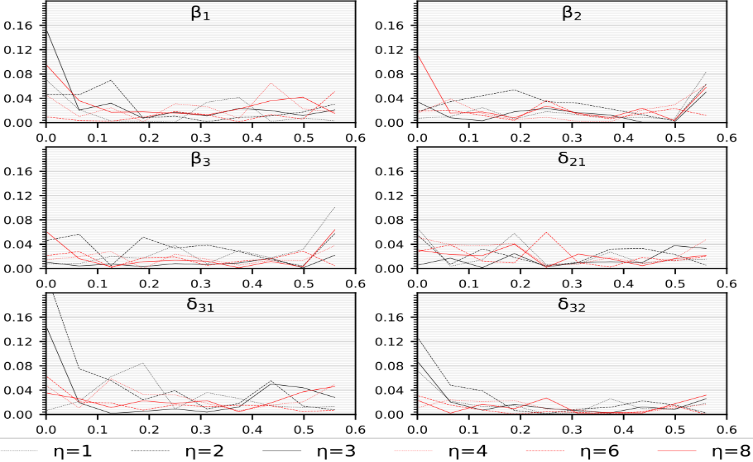
(a) Output variables vs for five different values of .



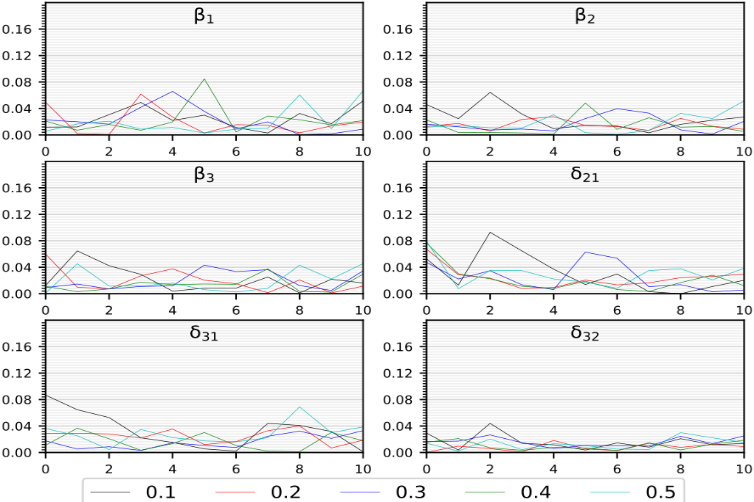
(b) Output variables vs for five different values of .

**Figure 7.** Comparison between actual and predicted values.

In Figures 7 and 8, continuous line denotes actual variation of output variables and predicted values are shown using markers. Figures 9 and 10 show percent deviations of predicted values from actual values of output variables.



(a) Percent deviation vs for six different values of .

****

(b) Percent deviation vs for five different values of .

**Figure 8.** Percent deviation of predicted values.

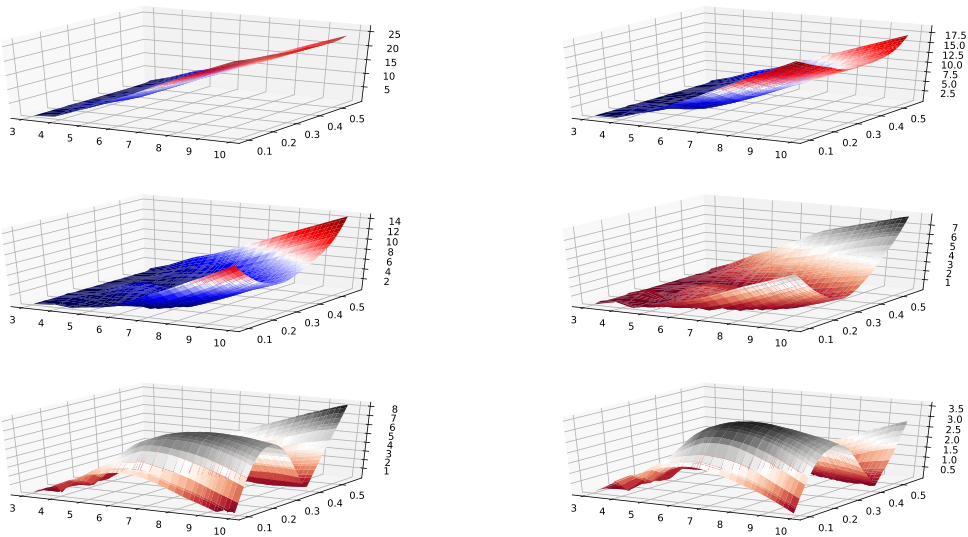
# Evaluation over unseen Data range

As shown in previous sections how well the models were predicting over the Test dataset. Now, to check the ability of models to predict over data from outside the interval of values used for training. Models obtained for case 1 and 2 were trained over the dataset generated from interval **η** є [3, 10], **δ0** є [7e-2, 0.56] and evaluated on Test dataset generated for **η** є [0, 3], **δ0** є [2e-3, 0.56] and vice-versa.

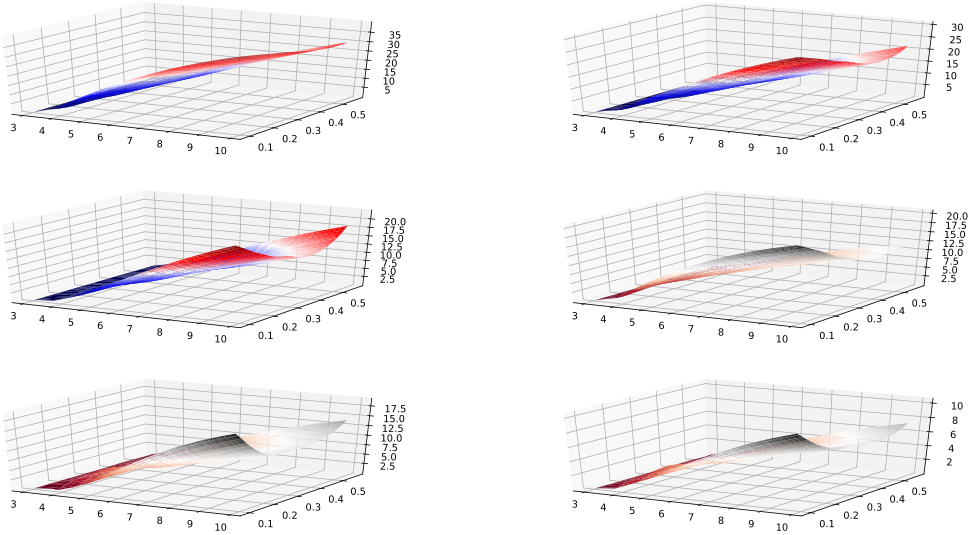
Firstly, both the models (Case 1 and Case 2) were trained on the smaller side of the data range of **η** (**η** є [0, 3], **δ0** є [2e-3, 0.56])and these trained models were then used to predict the frequency parameters and the node locations over the larger values of **η** (**η** є (3, 10], **δ0** є [7e-2, 0.56])**.**

**Table 5.** Trained on small and predicted for big values of **η.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Output** | **Mean Percent dev.** | | **Max. Percent dev.** | |
| **Case 1** | **Case 2** | **Case 1** | **Case 2** |
| β1 | 12.922 | 19.327 | 26.671 | 40.068 |
| β2 | 7.082 | 12.582 | 19.416 | 31.156 |
| β3 | 3.737 | 8.609 | 13.828 | 23.150 |
| δ21 | 1.866 | 8.318 | 7.643 | 21.166 |
| δ31 | 2.919 | 7.453 | 7.816 | 20.621 |
| δ32 | 1.331 | 4.310 | 3.593 | 11.100 |



(a) Case 1



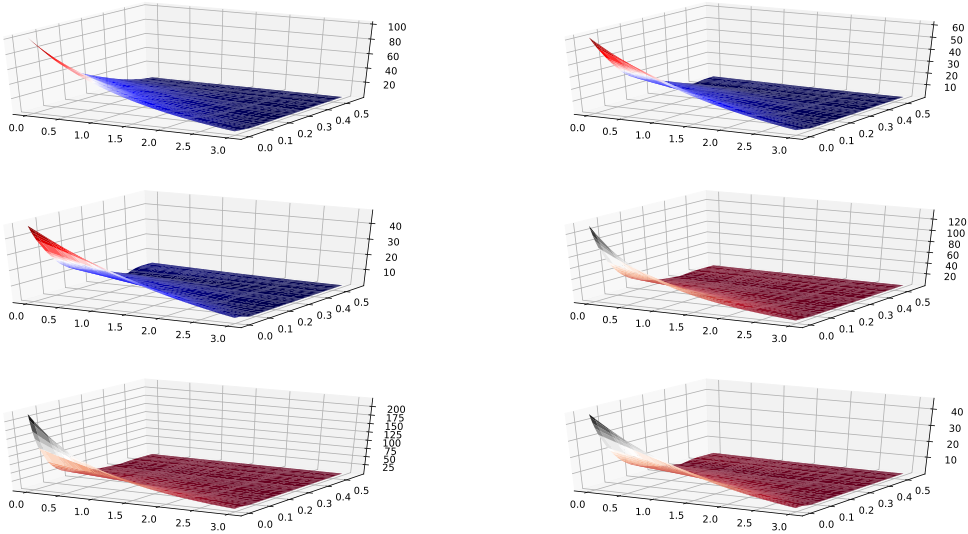
(b) Case 2

**Figure 9.** Percent deviations vs and (Table 5).

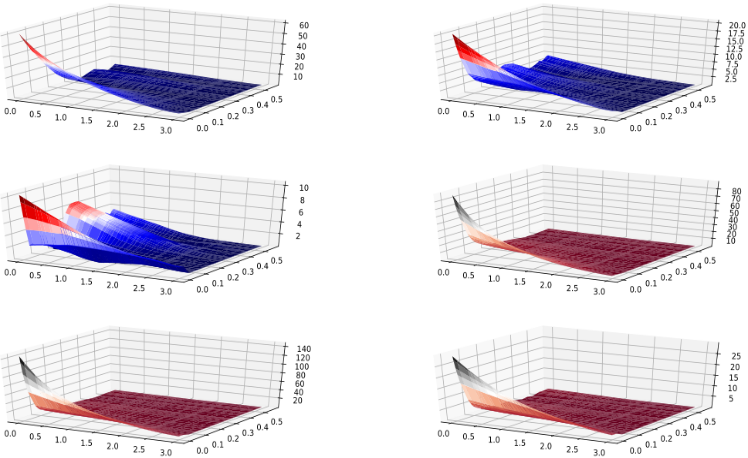
Then, both the models (Case 1 and Case 2) were trained on the large values of the data range of **η** (**η** є (3, 10], **δ0** є [7e-2, 0.56])and these trained models were then used to predict the frequency parameters and the node locations over the smaller values of **η** (**η** є [0, 3], **δ0** є [2e-3, 0.56])**.**

**Table 6.** Trained on big and predicted for small values of **η.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Output** | **Mean Percent dev.** | | **Max. Percent dev.** | |
| **Case 1** | **Case 2** | **Case 1** | **Case 2** |
| β1 | 10.620 | 4.630 | 94.813 | 57.242 |
| β2 | 6.181 | 1.808 | 58.538 | 19.544 |
| β3 | 4.826 | 1.361 | 45.678 | 10.128 |
| δ21 | 7.060 | 3.664 | 121.945 | 80.435 |
| δ31 | 9.475 | 4.975 | 202.181 | 134.401 |
| δ32 | 3.174 | 1.408 | 43.375 | 27.963 |



(a) Case 1



(b) Case 2

**Figure 10.** Percent deviations vs and (Table 6).

From the percent deviations shown in Tables 5 and 6 and plots (Figures 9 and 10) above, we can observe that relatively smaller network with less training data when trained on interval **η** є [0, 3], **δ0** є [2e-3, 0.56] gives better extrapolation over the data from unseen interval compared to larger network with more training datapoints, whereas if the network is trained on **η** є [3, 10], **δ0** є [7e-2, 0.56] larger network does better over unseen data range (**η** є [0, 3], **δ0** є [2e-3, 0.56]) compared to smaller network trained with lesser amount of data. These observations can also be associated to more complex nature of mapping function for smaller values of **η**.

Also, in Figure 10 we can see that huge percent deviations occur only while predicting for very small values of **η**,otherwise for the values of **η,** nearer to 3 percent deviations of the predicted values were not that large**.**

# Discussion

From the results presented above, it can be observed that when the value of input variable **η** is small (i.e., **η** є [0, 3] and **δ0** є [2e-3, .56]), a bigger neural network is required to attain the accuracy comparable to when the value of input variables is slightly big (**η** є [3, 10], **δ0** є [7e-2, .56]). Reason for lesser accuracy in smaller regime can be attributed to mapping function from input to output being more complicated for smaller values of **η**. Also, in cross-evaluation, model trained over small values of **η** performed better over bigger values of **η** compared to when model trained over bigger values of **η** evaluated on smaller values of **η**. This can also be ascribed to more complex mapping function in smaller regime. Also, using bigger neural network for training over smaller values of **η** overfits the data and thus, gives poor results while predicting over unseen data.

# conclusion

It can be concluded from the above results that for extrapolating over unseen data ranges, it is better to train smaller and less complex neural network rather than a complex one, if no prior knowledge about the complexity of unseen data is present. Although, if you know that unseen data is more complex than training data, then using a complex data for training will improve accuracy of predictions over unseen data. So, the baseline conclusion is that always use simpler neural network for extrapolation purposes in absence of any knowledge about unseen data.

As referred earlier, for generating the data MATLAB was used which is a licensed software and neural network training was done in python which is an open source software, therefore once model is trained it can be used in python by anyone. Also, this work shows how effectively data driven approach can be used to solve physics based problem.

# further applications

Models mentioned above can be trained on the data generated for beams under different boundary conditions other than cantilever structure, such as sliding, simply supported, etc. Since the underlying mapping function between input and output is similar for different boundary conditions, therefore, model with architecture similar to that used for analysis of cantilever beams should work reasonably well on other boundary conditions as well.

As dicussed in previous sections, our deep learning model was not able to predict natural frequencies and node locations for input values far from the range of input parameters used in training, therefore better models could be developed to improve the extrapolation accuracy by using the activation functions such that dynamics of underlying physics could be captured more accurately by the neural network. Some research has already been done to improve the extrapolating abilities of neural networks, these papers used appropriate activaion functions in order to capture the exact dynamics of physical problem [17-21]. Since, the solution of Euler-Bernoulli beam equation contains integration of expressions with factorial and many other complex non-linear functions in numerator and denominator, therefore finding activation functions that represent the dynamics of freely vibrating beam structure is quite challenging.

# acknowledgment

The authors would like to acknowledge partial financial support from the grant under Advanced Manufacturing Technology Program of Department of Science and Technology, Government of India.

**REFERENCES**

1. Mituletu, I. C., Gillich, G. -., & Maia, N. M. M. (2019). A method for an accurate estimation of natural frequencies using swept-sine acoustic excitation. Mechanical Systems and Signal Processing, 116, 693-709. DOI: 10.1016/j.ymssp.2018.07.018
2. Shafiei, N., Ghadiri, M., & Mahinzare, M. (2019). Flapwise bending vibration analysis of rotary tapered functionally graded nanobeam in thermal environment. Mechanics of Advanced Materials and Structures, 26(2), 139-155. DOI: 10.1080/15376494.2017.1365982
3. Sanders, Wesley. (2019). Introduction to Atomic Force Microscopy. DOI: 10.1201/9780429266553-1.
4. Flueckiger, Jonas & Ko, Frank & Cheung, Karen. (2009). Microfabricated Formaldehyde Gas Sensors. Sensors (Basel, Switzerland). 9. 9196-215. DOI: 10.3390/s91109196.
5. D. Conway, H & C. H. Becker, E & F. Dubil, J. (1964). Vibration Frequencies of Tapered Bars and Circular Plates. Journal of Applied Mechanics. 31. DOI: 10.1115/1.3629606.
6. Sanger, D. J. (1968). Transverse Vibration of a Class of Non-Uniform Beams. Journal of Mechanical Engineering Science, 10(2), 111–120.
7. D. Conway, H & C. H. Becker, E & F. Dubil, J. (1965). Closure to ``Discussion of `Vibration Frequencies of Tapered Bars and Circular Plates''' (1965, ASME J. Appl. Mech., 32, pp. 234-235). Journal of Applied Mechanics. DOI: 32.10.1115/1.3625766.
8. F. Ward, P. (1913). IX. The transverse vibrations of a rod of varying cross-section. Philosophical Magazine Series 6. 25. 85-106. DOI: 10.1080/14786440108634312.
9. Zhou, Ding & K. Cheung, Y. (2000). Free vibration of a type of tapered beams. Computer Methods in Applied Mechanics and Engineering-COMPUT METHOD APPL MECH ENG. DOI: 188.203-216.10.1016/S0045-7825(99)00148-6.
10. Avcar, Mehmet & Saplıoğlu, Kemal. (2015). An Artificial Neural Network Application for Estimation of Natural Frequencies of Beams. International Journal of Advanced Computer Science and Applications. 6. 94-102. DOI: 10.14569/IJACSA.2015.060614.
11. MATLAB Release 2019a, The MathWorks, Inc., Natick, Massachusetts, United States.
12. T. A. Driscoll, N. Hale, and L. N. Trefethen, editors, Chebfun Guide, Pafnuty Publications, Oxford, 2014.
13. Lei Ba, Jimmy & Ryan Kiros, Jamie & E. Hinton, Geoffrey. (2016). Layer Normalization.
14. He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. IEEE International Conference on Computer Vision (ICCV 2015). 1502. DOI: 10.1109/ICCV.2015.123.
15. Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.
16. Pytorch, <https://github.com/pytorch/pytorch>
17. Trask, Andrew & Hill, Felix & Reed, Scott & Rae, Jack & Dyer, Chris & Blunsom, Phil. (2018). Neural Arithmetic Logic Units.
18. Jiang, Weiguang & Hagen, G. & Papenbrock, T.. (2019). Extrapolation of nuclear structure observables with artifical neural networks.
19. Xie, Guangzeng & Wang, Yitan & Zhou, Shuchang & Zhang, Zhihua. (2018). Interpolatron: Interpolation or Extrapolation Schemes to Accelerate Optimization for Deep Neural Networks.
20. Martius, Georg & Lampert, Christoph. (2016). Extrapolation and learning equations.
21. Haley, P.J. & Soloway, Don. (1992). Extrapolation limitations of multilayer feedforward neural networks. 25 - 30 vol.4. DOI: 10.1109/IJCNN.1992.227294.

**NOMENCLATURE**

|  |  |
| --- | --- |
| AC | cross-sectional area at the clamped end |
| A(x) | area of cross section at position x |
| E | Young’s modulus of the beam material, Pa |
| IC | moment of inertia about flexural axis at the clamped end |
| I(x) | moment of inertia about flexural axis at x |
| L | length of the conical beam with same AC as that of the truncated beam |
| L1 | number of hidden layers with layer normalisation and activation |
| LT | length of truncated portion of the beam |
| N1 | number of units in the hidden layers with activation and normalisation units |
| N2 | parameter defining number of units in the hidden layers without activation function and normalisation. Number of units in these layers is 6\*N2. |

**Greek symbols**

|  |  |
| --- | --- |
| β | dimensionless frequency parameter |
| δ0 | fraction of truncated portion, δ0 = LT/L |
| η | exponent governing the croos-sectional properties of the beam |
|  | density of the material of beam, kg m-3 |