Proceedings of IMEC 2019

International Mechanical Engineering Congress (IMEC-2019),

29th November- 1st December 2019, NIT Tiruchirappalli, India.

IMEC2019-xx-xxxx

**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](mailto:160100012@iitb.ac.in)

+Corresponding author email: [asim.tewari@iitb.ac.in](mailto:asim.tewari@iitb.ac.in)

**ABSTRACT**

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: Deep Learning, cantilever beams, free vibrations, natural frequencies, mode shapes.***

**INTRODUCTION**

Various papers have already been published regarding the vibration analysis of a certain class of truncated tapered beams using Bessel functions [1-5]. Cantilever beam structure has numerous applications, thus estimation of natural frequencies and node locations is crucial [6-7]. In this paper artificial neural networks were used to predict natural frequency parameters and node locations for first three modes of the 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:

|  |  |  |
| --- | --- | --- |
|  | , | (\*) |

where, AC and IC are the cross-sectional area and moment of inertia about flexural axis at the clamped end, δ=x/L ϵ [δ0, 1], δ0=LT/L, 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.

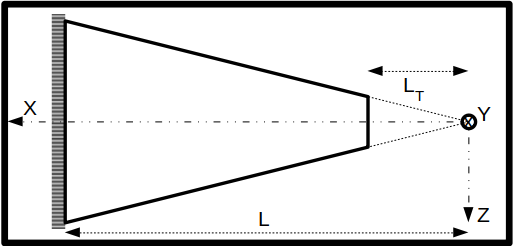
****

Figure 1: Cantilever beam structure.

**THEORY**

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

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

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,

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

General solution for in Eq. (2) can be given as,

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

, ,

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 Fig. (1),

|  |  |  |
| --- | --- | --- |
|  |  | (a) |
|  |  | (4b) |

After applying boundary conditions (Eq. 4a-4b) on Eq. (3), following set of linear equations was obtained,

|  |  |  |
| --- | --- | --- |
|  |  | (a) |
|  |  | (5b) |
|  |  | (5c) |
|  |  | (5d) |

where, ,

Now, this linear system of equations (Eq. 5a-5d) to have non-trivial solution following condition should satisfy,

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

After solving Δ = 0 (Eq. 6), we can get values of β = βr, where, r is the mode number. From which, we can compute and using following relations,

|  |  |  |
| --- | --- | --- |
|  | , | () |

Finally, mode shapes can be obtained using Eq. 7 as follows,

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

**DEFINING INPUT AND OUTPUT VARIABLES**

Inputs- **η,** **δ0**.

Outputs- **β1, β2, β3**, **δ21, δ31, δ32**.

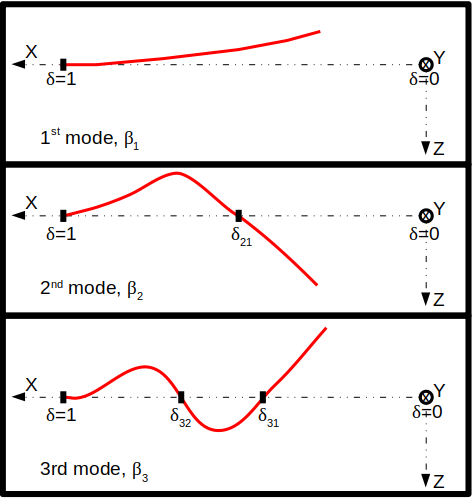
where, **η** is exponent defining the variation of Area of cross-section of the cantilever along x-axis and **δ0** is fraction of truncated portion of cantilever. **β1**, **β2**, **β3** are parameters defining first three natural frequencies and **δ21**, **δ31**, **δ32** are non-dimensionalized numbers defining node locations of first 3 mode shapes (Figure 2).

Figure 2: Pictorial representation of output variables.

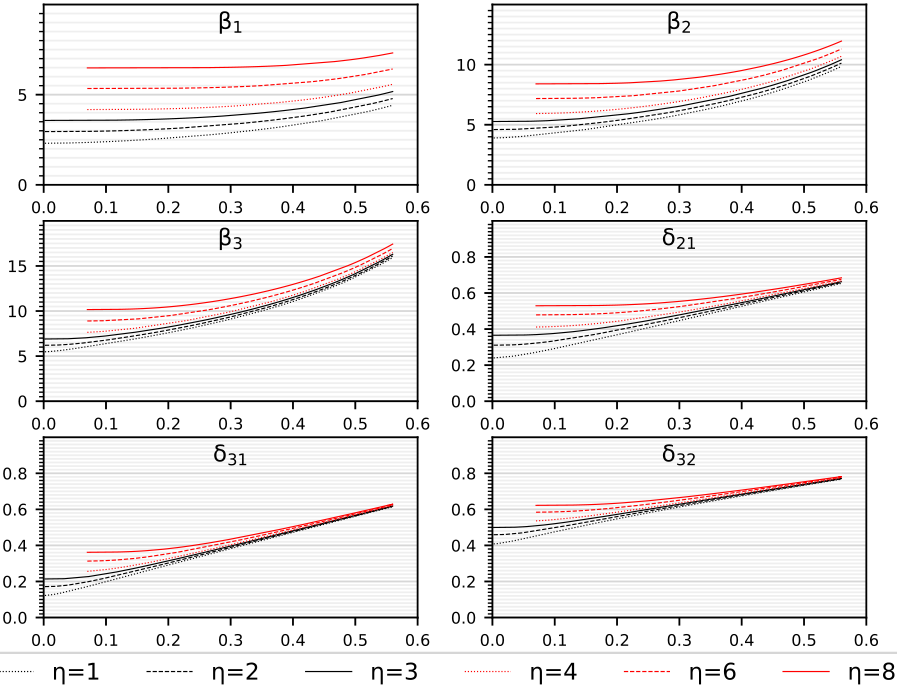
****Based on the theory, following plots (Figure 3-4) were generated using MATLAB [8] showing theoretical variation in output variables with respect to input variables. For higher values of **η** and very low values **δ0** our solver [9] was not able to solve the equations with enough accuracy, therefore, those values were skipped as can be seen in figure 3.

Figure 3: Actual variation of output values with respect to for 6 different values of .

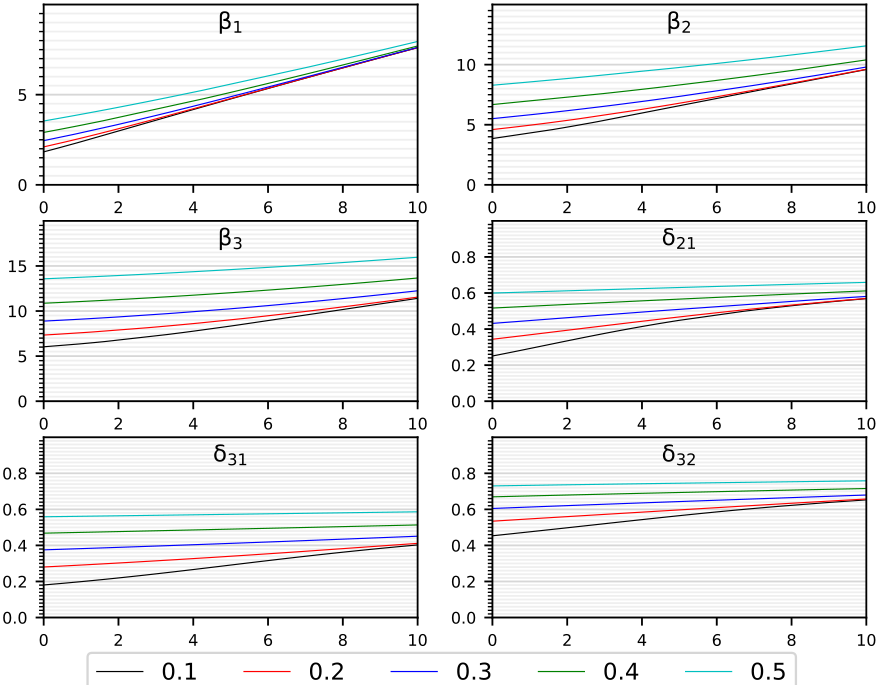
****

Figure 4: Actual variation of output values with respect to for 5 different values of .

**DATA GENERATION**

1. 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**.

2. So, overall distribution of data points was as following:

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

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

- 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.

3. 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.

**DEFINING LEARNING MODEL**

1. **Neural network architecture**.

[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]

2. **Layer Normalisation** [10].

Output of normalisation layer is given by :

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

,

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.

3. **Activation function** [11] – PReLU

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

where, a0 is a learnable parameter.

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

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

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.

5. **Adam optimizer** [12] used for minimising the loss, betas = {0.9, 0.999}, along with mini-batch gradient descent.

6. 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.

**TRAINING THE NEURAL NETWORK**

1. Models were tuned for two different intervals of **η**:

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

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

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

I. **L1** = Number of hidden layers with layer normalisation and activation.

II. **N1** = Number of units in “**L1**” hidden layers.

III. **N2** = Parameter defining number of units in remaining hidden layers.

IV. **Initial LR** = Initial learning rate.

V. **Update period =** Number of iterations after which the learning rate will be updated.

VI. **Update factor =** Factor by which to divide on each update.

3. For tuning the hyperparameters mentioned above, 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.

4. 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 1: DL model training parameters | | |
|  | **Case 1** | **Case 2** |
| # Training datapoints | 2\*211 | 20\*211 |
| # Iterations | 30,000 | |
| Initial LR | 0.01 | |
| Update period | 5,000 | |
| Update factor | 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.

Models were trained using Pytorch [13] on NVIDIA GeForce GTX 1060 6GB GPU.

|  |  |  |
| --- | --- | --- |
| Network architecture | | |
| Hyper-parameter | **Case 1** | **Case 2** |
| L1 | 3 | 7 |
| N1 | 173 | 207 |
| N2 | 3 | 3 |

**RESULTS OBTAINED AFTER TRAINING**

1. 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.

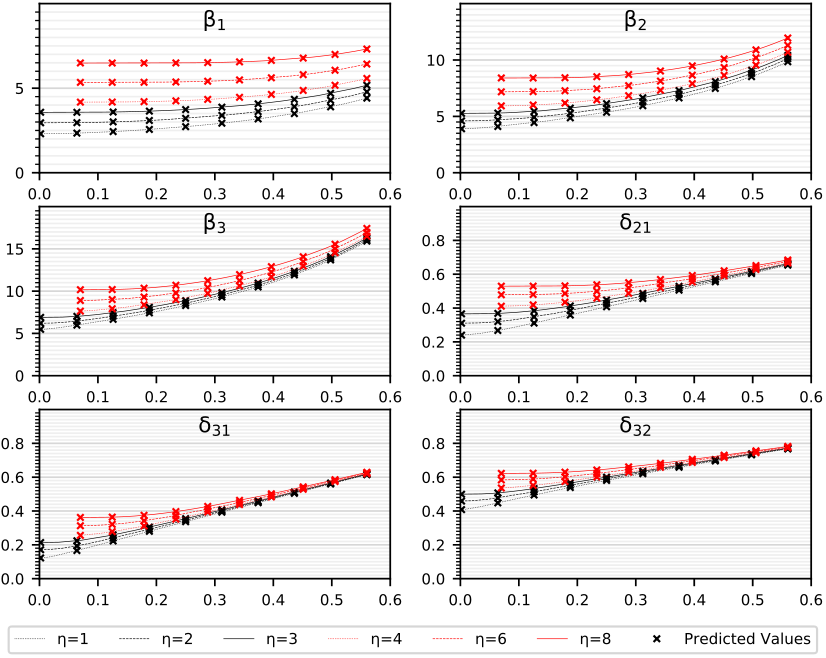
**Case 1**: **L1** = 3, **N1** = 173, **N2** = 3.

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

2. 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 2: For 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 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 3: For 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 |

3. 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 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.

4. 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 tables above. 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 Figure 5-6.

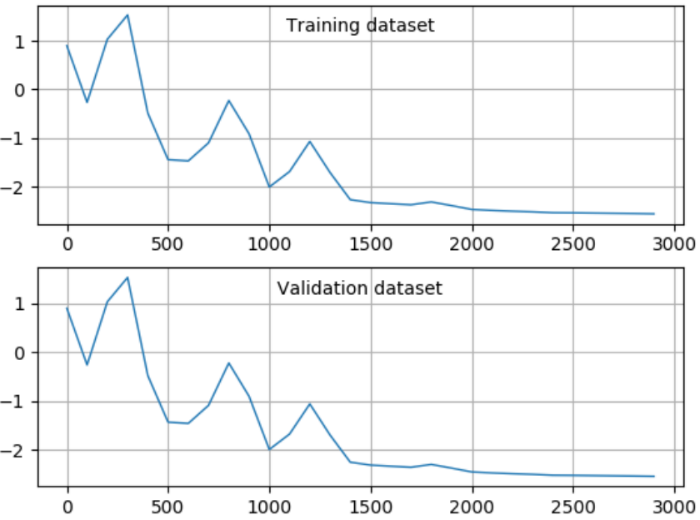
****

Figure 5: Mean squared error on log10 scale vs no. of iterations.

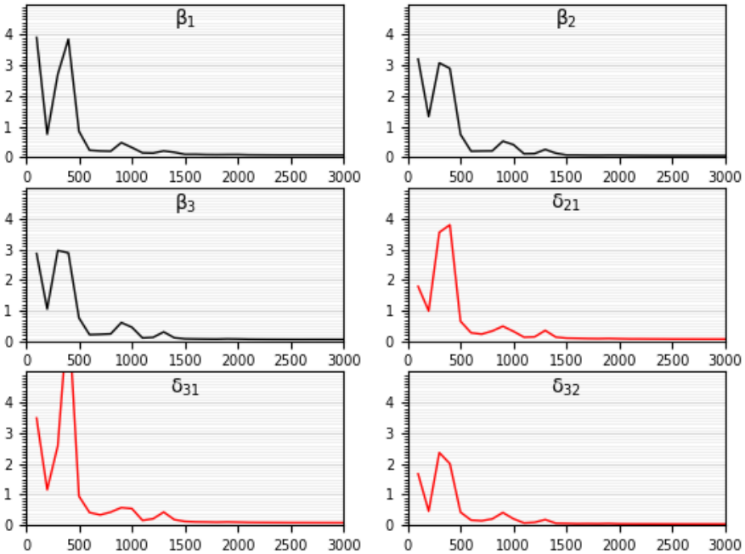
****

Figure 6: Validation dataset- 99th percentile error vs no. of iterations.

**GRAPHICAL COMPARISON BETWEEN ACTUAL AND PREDICATED VALUES**

1. Following plots are for model with 7 hidden layers, i.e. training case 2, trained over entire range.

2. 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.

Figure 7: Comparison between actual and predicted values of output variables for six different values of.

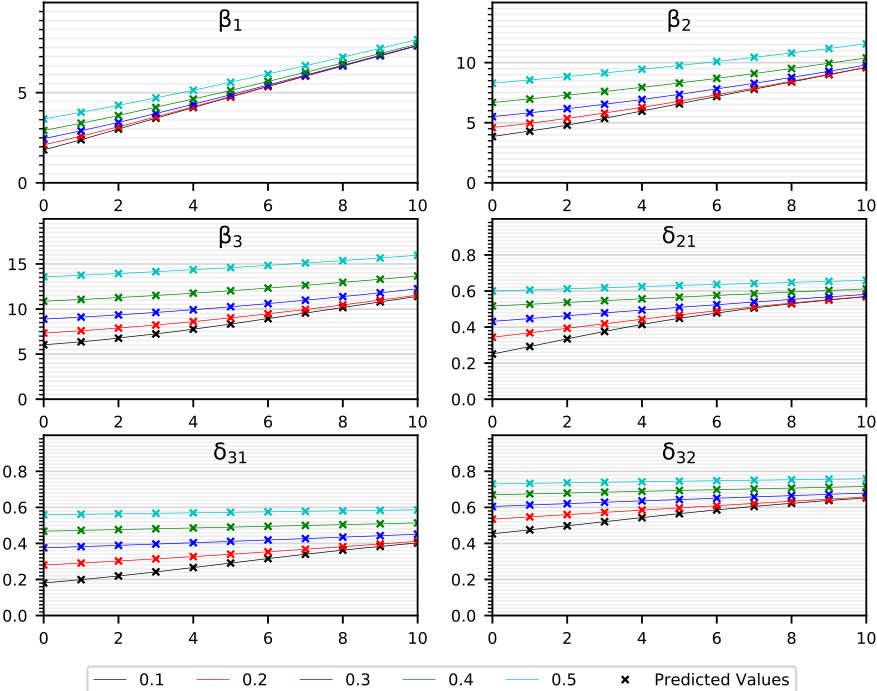


Figure 8: Comparison between actual and predicted values of output variables for five different values of .

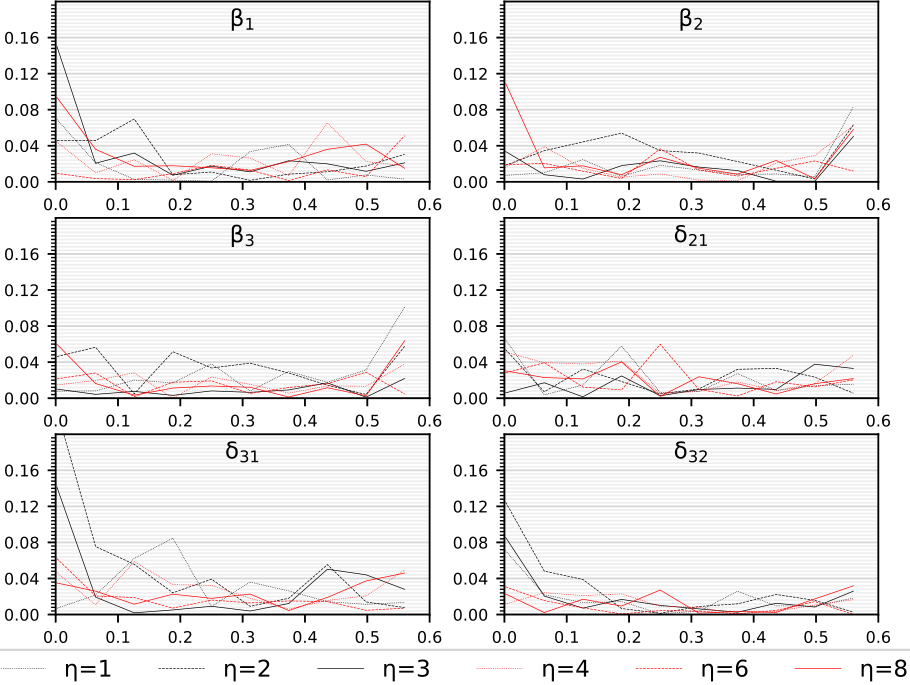


Figure 9: Percent deviation vs for six different values of .

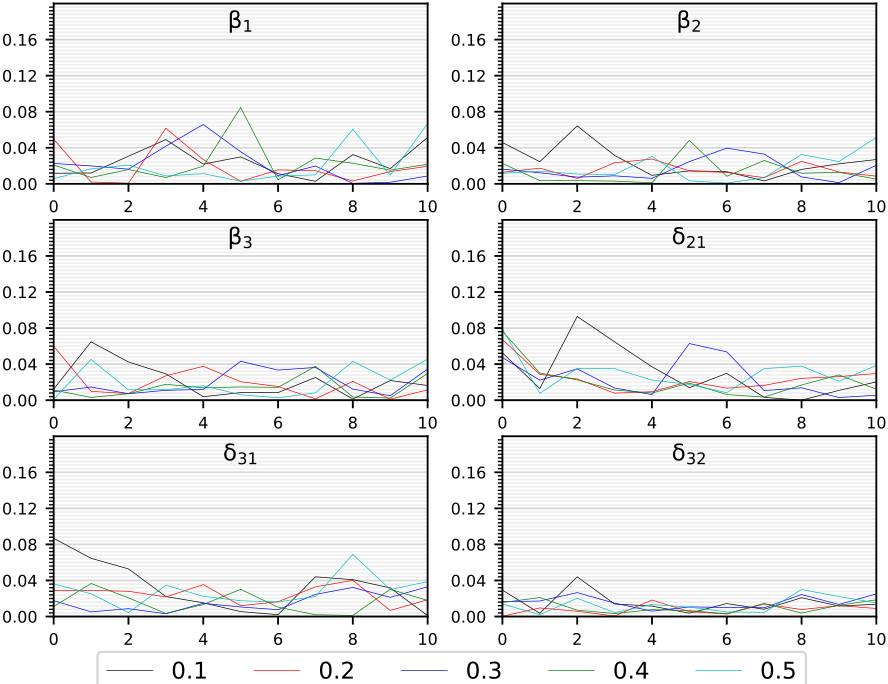
****

Figure 10: Percent deviation vs for five different values of .

**EVALUATION OVER UNSEEN DATA RANGE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 4: Trained over η є [0, 3] and  evaluated over η є [3, 10] | | | | |
| 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 |

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.

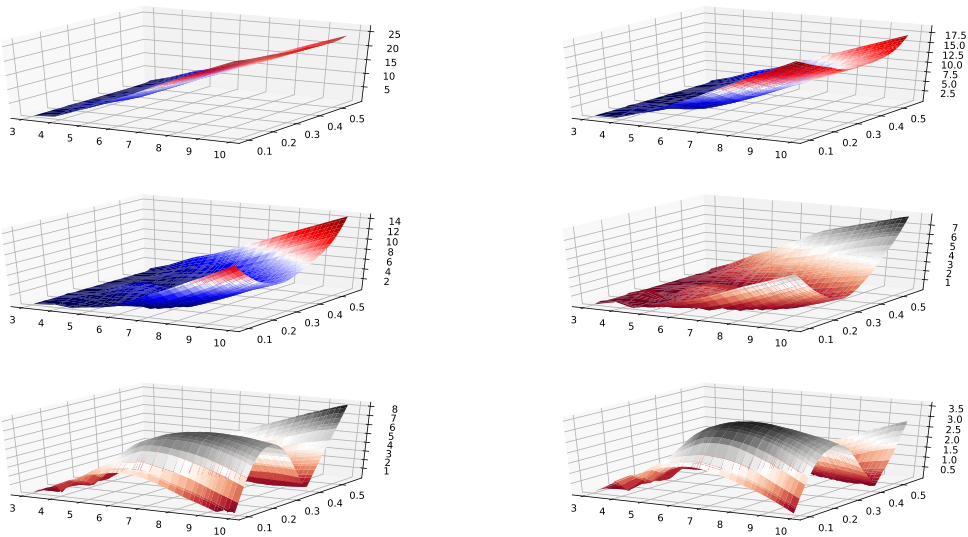


Figure 11: Percent deviation vs and for case 1 over unseen range.

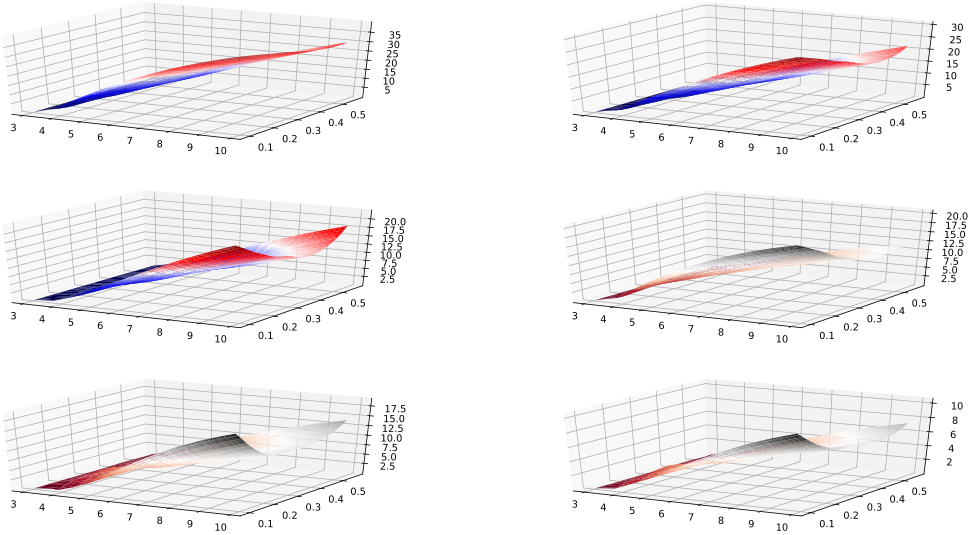


Figure 12: Percent deviation vs and for case 2 over unseen range.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 5: Trained over η є [3, 10] and  evaluated over η є [0, 3] | | | | |
| 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 |

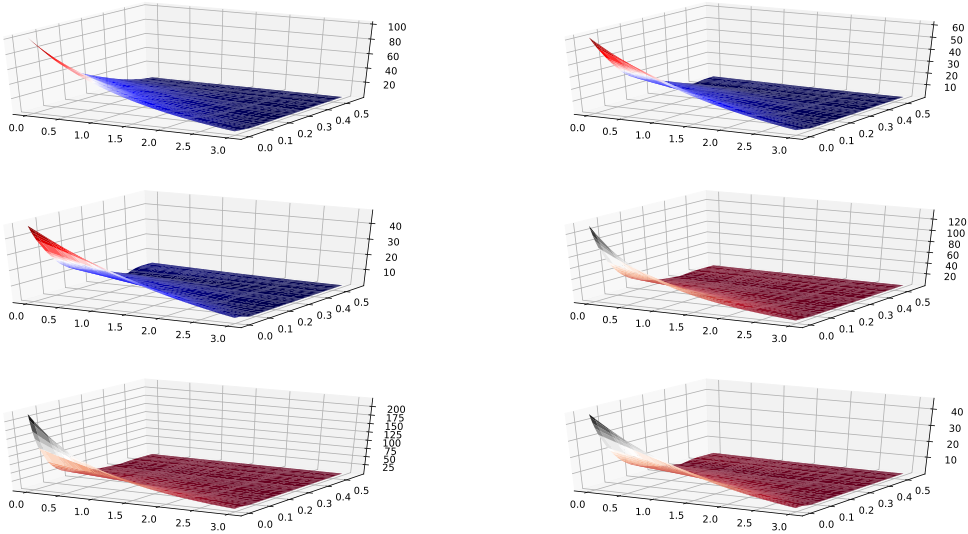


Figure 13: Percent deviation vs and for case 1 over unseen range.

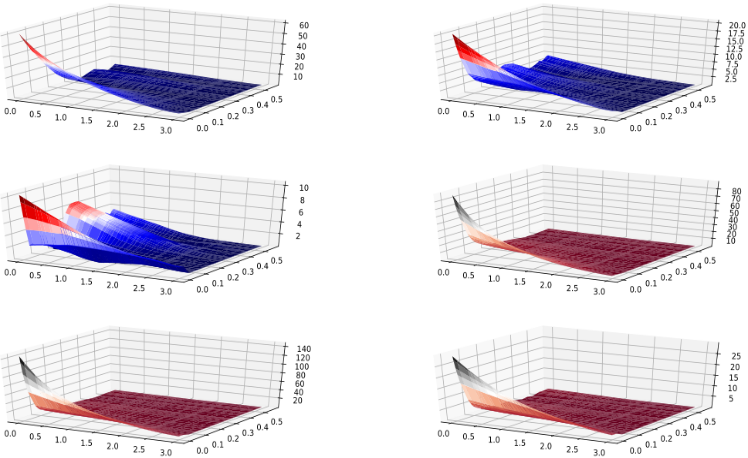


Figure 14: Percent deviation vs and for case 2 over unseen range.

From the results shown in Tables 4-5 and plots (Figure 11-14) 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**.

**DISCUSSION**

From the results presented above, it can be observed that when the value of input variables (**η** and **δ0)** is small (**η** є [0, 3], **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, baseline conclusion is that always use simpler neural network for extrapolation purposes in absence of any knowledge about unseen data.

**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. Also, better models could be developed to improve the extrapolation accuracy.

**REFERENCES**

1. D. Conway, H & C. H. Becker, E & F. Dubil, J. (1964). Vibration Frequencies of Tapered Bars and Circular Plates. Journal of Applied Mechanics. 31. 10.1115/1.3629606.

2. Sanger, D. J. (1968). Transverse Vibration of a Class of Non-Uniform Beams. Journal of Mechanical Engineering Science, 10(2), 111–120.

3. 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. 32.10.1115/1.3625766.

4. F. Ward, P. (1913). IX. The transverse vibrations of a rod of varying cross-section. Philosophical Magazine Series 6. 25. 85-106. 10.1080/14786440108634312.

5. 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. 188.203-216.10.1016/S0045-7825(99)00148-6.

6. 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

7. 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

8. MATLAB Release 2019a, The MathWorks, Inc., Natick, Massachusetts, United States.

9. T. A. Driscoll, N. Hale, and L. N. Trefethen, editors, Chebfun Guide, Pafnuty Publications, Oxford, 2014

10. Lei Ba, Jimmy & Ryan Kiros, Jamie & E. Hinton, Geoffrey. (2016). Layer Normalization.

11. 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. 10.1109/ICCV.2015.123.

12. Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

13. Pytorch, <https://github.com/pytorch/pytorch>