Vibration Analysis of Tapered Truncated cantilever beams using Deep Learning

Abstract

This work relates to the use of deep learning in vibration analysis of tapered truncated cantilever beams. In this report, an attempt was made to train a deep learning neural network to learn the natural frequency parameters and node locations of first three mode shapes for tapered truncated cantilever beams. Beam shapes that were used to generate data and train the neural network belong to a quite general class. For reporting the final results, trained neural networks were evaluated an a separate teat dataset different from both training and validation set. After tuning some hyperparameters, maximum percent deviation of all the outputs was brought down to less than 0.1 percent. Finally, cross-evaluation of trained models was done, to see the extrapolating capabilities of it over unseen data.

Introduction

Various papers have already been published regarding the vibration analysis of a certain class of truncated tapered beams using Bessel functions. There are papers written on predicting natural frequencies of cantilever beams of constant cross-section properties using neural networks. 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 existing solutions to obtain the natrural frequencies and mode shapes for generating 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. In the following work only cantilever beams were analysed, with larger end clamped and smaller end free. This work deals with the class of beams satisfying the variation in sectional properties as following:

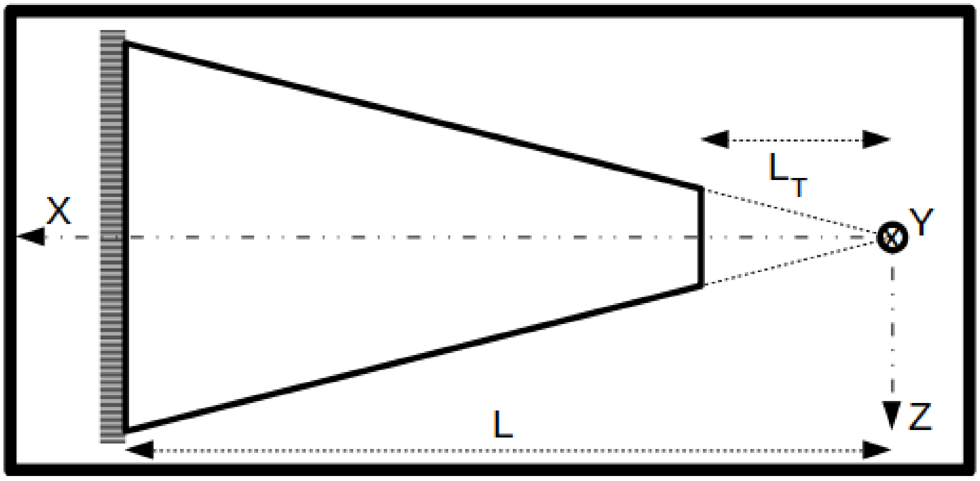
$$A = A_C \, \delta^{\eta}$$

$$I = I_C \, \delta^{\eta+2}$$

where, A_C and I_C are the cross-sectional area and moment of inertia about flexural axis at the clamped end,

$\delta = x/L \in [\delta_0, 1]$, $\delta_0 = L_T/L$, L_T is the length of truncated portion, L is the total length of beam without truncation,

η is a real number $\in [0, 10]$ specifying the variation in the area cross-section.



Theory

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

$$\frac{\partial^2}{\partial x^2}\left(EI(x)\frac{\partial^2 y(x,t)}{\partial x^2}\right)+\rho A(x)\frac{\partial^2 y(x,t)}{\partial t^2}=0 \quad \text{----------------- (1)}$$

where, y(x, t) is displacement at position x in yz-plane 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.

R is the density of the material.

A(x) is the area of cross section at position x.

Since variable separable is applicable here, therefore, we can assume solution in the folllowing form,

$$y(x,t)=W(x)e^{i\omega t}$$

General solution for W(x) can be given as,

$$W(\delta)=(L\delta)^{-\eta/2}\left[c_1 J_\eta(z)+c_2 Y_\eta(z)+c_3 I_\eta(z)+c_4 K_\eta(z)\right] \quad \text{------------- (2)}$$

where, J_η- Bessel Function of first kind.

Y_η-Bessel Function of second kind.

I_η- Modified Bessel Function of first kind.

K_η- Modified Bessel Function of second kind.

$$z=2\beta\delta^{1/2} \quad , \quad \beta=\omega^2 L^4\left(\frac{\rho A_C}{EI_C}\right) \quad , \quad \delta=\frac{x}{L}$$

Applying Boundary conditions on equation (2),

$$\frac{\partial}{\partial\delta}\left(EI(\delta)\frac{\partial^2 W(\delta)}{\partial\delta^2}\right) = \frac{\partial^2 W(\delta)}{\partial\delta^2} = 0, \quad at\,\delta=\delta_0$$

$$W(\delta) = \frac{\partial W(\delta)}{\partial\delta} = 0, \quad at\,\delta=\delta_1=1$$

After applying boundary conditions following set of linear equations obtained,

$$c_1 J_3(z_0)+c_2 Y_3(z_0)+c_3 I_3(z_0)-c_4 K_3(z_0) = 0$$

$$c_1 J_4(z_0)+c_2 Y_4(z_0)+c_3 I_4(z_0)+c_4 K_4(z_0) = 0$$

$$c_1 J_2(z_1)+c_2 Y_2(z_1)+c_3 I_2(z_1)+c_4 K_2(z_1) = 0$$

$$c_1 J_3(z_1) + c_2 Y_3(z_1) - c_3 I_3(z_1) + c_4 K_3(z_1) \;=\; 0$$

where, $z_0 = 2\beta\delta_0^{1/2}$, $z_1 = 2\beta\delta_1^{1/2} = 2\beta$

Now, for the above linear system of equations to have non-trivial solution following condition should satisfy,

$$\Delta \;=\; \begin{vmatrix} J_{\eta+1}(z_0) & Y_{\eta+1}(z_0) & I_{\eta+1}(z_0) & -K_{\eta+1}(z_0) \\ J_{\eta+2}(z_0) & Y_{\eta+2}(z_0) & I_{\eta+2}(z_0) & K_{\eta+2}(z_0) \\ J_{\eta}(z_1) & Y_{\eta}(z_1) & I_{\eta}(z_1) & K_{\eta}(z_1) \\ J_{\eta+1}(z_1) & Y_{\eta+1}(z_1) & -I_{\eta+1}(z_1) & K_{\eta+1}(z_1) \end{vmatrix} \;=\; 0$$

By solving $\Delta = 0$, we can get values of $\beta = \beta_r$, where, r is the mode number.

From values of $\beta_{r,}$ we can compute ω_r and z_r using following relation relations,

$$\omega_r = \left(\frac{\beta_r}{L}\right)^2 \sqrt{\frac{EI_C}{\rho A_C}} \quad , \quad z_r = 2\beta_r \delta^{1/2}$$

Finally, mode shapes can return as,

$$W(\delta) = (L\delta)^{-\eta/2}\left[c_1 J_\eta(z_r) + c_2 Y_\eta(z_r) + c_3 I_\eta(z_r) + c_4 K_\eta(z_r)\right]$$

<u>Defining Input and Output variables of Neural Network</u>

<u>Inputs</u>- **η, δ₀**.

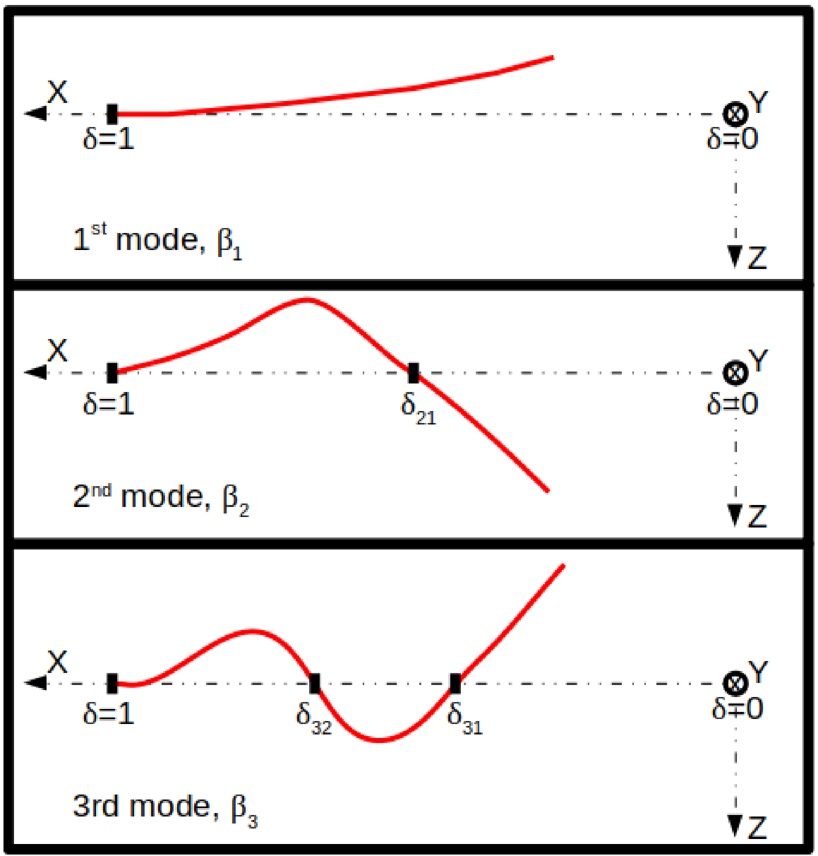
where, **η** – exponent defining the variation of Area of cross-section of the cantilever along x-axis.

δ_0 – fraction of truncated portion of cantilever.

<u>Outputs</u>- **β_1, β_2, β_3, δ_{21}, δ_{31}, δ_{32}**.

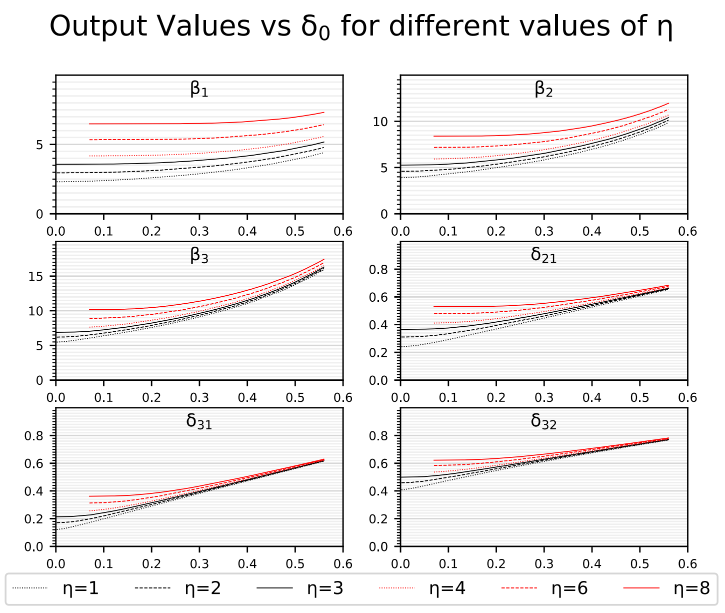
where, **β_1, β_2, β_3** – parameters defining first three natural frequencies.

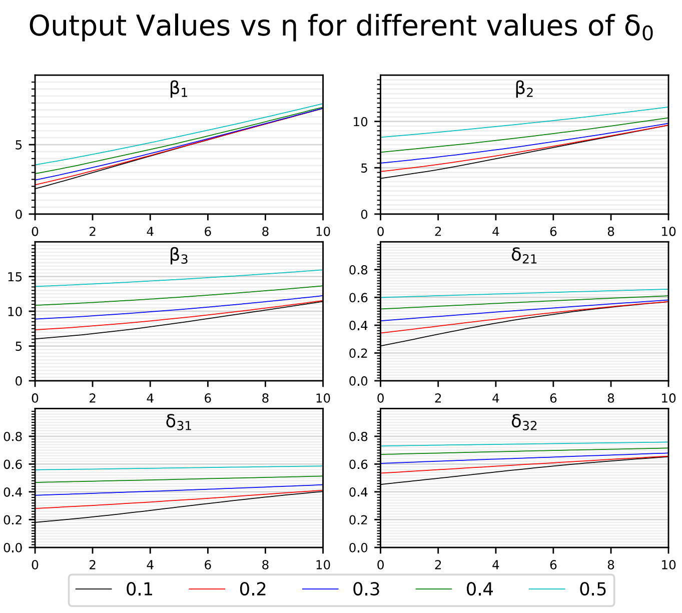
δ_{21}, δ_{31}, δ_{32} – non-dimensionalized numbers defining node locations of first 3 modes



<u>**Actual variation of output variables**</u>

Based on the theory, following plots were generated using MATLAB showing theoretical variation in output variables with respect to input variables. For higher values of n and very low values d0 our solver was not able to solve the equations with sufficient accuracy, therefore, those values were skipped as you can see in graphs.





Data generation

1. Data was generated for 1,001 values of **η** ϵ [0, 10] and for each **η**, 15000 data points were collected by varying **δ_0** between [2e-3, 0.56] for **η** ϵ [0, 3] and [7e-2, 0.56] for **η** ϵ (3, 10], reason for this split being insufficient accuracy in the frequency parameters calculated for small values of **δ_0** and large values of **η**.

2. So, overall data points were generated in the following way:

   - For **η** ϵ [0, 3], **δ_0** ϵ [2e-3, 0.56] --> 301*15000=4,515,000.
   - For **η** ϵ [3, 10], **δ_0** ϵ [7e-2, 0.56] --> 701*15000=10,515,000.
   - For **η** ϵ [0, 10], data for above two cases was merged as described below,
     + **η** ϵ [0, 3], **δ_0** ϵ [2e-3, 0.56] --> 301*15000=4,515,000.
     + **η** ϵ (3, 10], **δ_0** ϵ [7e-2, 0.56] --> 700*15000=10,500,000.

3. Apart from above datapoints a separate Test dataset of 50*100=5,000 examples was created for each interval (**η** ϵ [0,3] and **η** ϵ [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 **δ_0** were randomly chosen.

4. For data generation MATLAB was used.

Defining Neural Network 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 layer having N1 units each] -->

   --> [Fully connected Hidden layer without activation function, with 6*N2 units] -->

   --> [Output layer, 6 units]

2. **Layer Normalisation**.

$$\overline{z}_i^l = \frac{z_i^l - \mu[z_i]^l}{\sqrt{Var[z_i]^l + \epsilon}} * \gamma + \beta \quad \text{where, } \mu[z_i]^l = \left(\frac{1}{H}\right)\sum_{i=1}^{H} z_i^l,\ Var[z_i]^l = \left(\frac{1}{H}\right)\sum_{i=1}^{H}\left(z_i^l - \mu^l\right)^2$$

where, z_i^l 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 $\mu[z_i]^l$ and $Var[z_i]^l$, but different training datapoints have different normalization terms.

Note: γ and β are learnable parameters.

3. **Activation function –** PreLU

$$\text{PreLU}(z_i^l) = max(0, z_i^l) + a_0 * min(0, z_i^l)$$

where, a_0 is a lernable parameter.

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

$$\left(\frac{1}{N}\right)\sum_{j=1}^{N}\sum_{i=1}^{6}\left(\frac{act_{ij}-pred_{ij}}{act_{ij}}*100\right)^2$$

Note: N is the number training examples.

act$_{ij}$ is actual value of ith output of jth training example.

pred$_{ij}$ is predicted value of ith output of jth training example.

5. **Adam optimizer** 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, Sum squared percent error and Adam gave best results. Layer normalisation didn't improve the accuracy significantly, but still it gave positive results.

Training the Neural Network

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

1) **η** ϵ [0, 3], **δ_0** ϵ [2e-3, 0.56].

2) **η** ϵ [3, 10], **δ_0** ϵ [7e-2, 0.56].

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

1) Number of hidden layers.

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

2) Number of units in each hidden layer.

- **N1** = Number of units in "**L1**" hidden layers.

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

4) Learning Rate.

- **Initial LR** = Initial learning rate.

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

- **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 cretain interval best suited for that particular hyperparameter. Sampling interval of all hyperparameters was narrowed down over subsequent iterations, around the region giving best performance for finner 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:

	# Training datapoints	**# Iterations**	**Initial LR**	**Update period**	**Update factor**
Case 1	$2*2^{11}$	30,000	0.01	5,000	5
Case 2	$20*2^{11}$	30,000			

Note: # Training examples = $n*2^{m}$, where n=#mini-batches, m=mini-batch size.

Iteration over n mini-batches,i.e. whole training set, was caonsidered as one iteration.

4. Models were trained using pytorch on NVIDIA GTX GeForce 1060 6GB GPU.

Results obtained after training

1. Out of several models trained and evaluated Neural networks with architecture similar to what mentioned below performed best depending upon the number of datapoints used for training. It can be easily observed from following results that a bigger network can be trained using even more training data to further reduce errors.

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

I. For **η** ϵ [0, 3], $\boldsymbol{\delta_0}$ ϵ [2e-3, 0.56]

Output	**Mean Percent deviation**		**Max. Percent deviation**	
	Case 1	**Case 2**	**Case 1**	**Case 2**
$\boldsymbol{\beta_1}$	0.022360881790519	0.011400782503188	0.115375153720379	0.106297180056572
$\boldsymbol{\beta_2}$	0.019665371626616	0.008312780410051	0.110458210110664	0.071469284594059
$\boldsymbol{\beta_3}$	0.02081055007875	0.008577669970691	0.111438997089863	0.052421003580093
$\boldsymbol{\delta_{21}}$	0.015991751104593	0.011992040090263	0.171569153666496	0.100973047316074
$\boldsymbol{\delta_{31}}$	0.023540297523141	0.013064150698483	0.316999644041061	0.255279392004013

δ_{32}	0.012375002726913	0.006703139282763	0.159147039055824	0.082595251500607

II. For **η** ϵ [3, 10], **δ_0** ϵ [7e-2, 0.56]

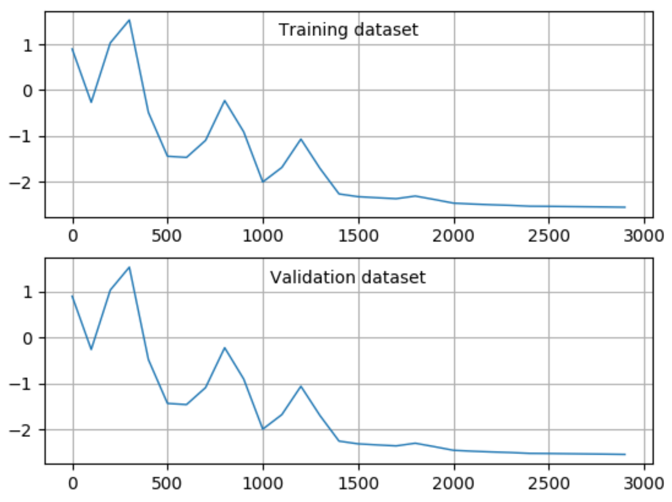
Output	**Mean Percent deviation**		**Max. Percent deviation**	
	Case 1	**Case 2**	**Case 1**	**Case 2**
β_1	0.018967673182488	0.010999544523656	0.130405351519585	0.079486951231957
β_2	0.017713138833642	0.009572512470186	0.130365923047066	0.046215891838074
β_3	0.019474877044559	0.010280021466315	0.12267205119133	0.054073955863714
δ_{21}	0.013449708931148	0.009197227656841	0.090339682996273	0.050381261855364
δ_{31}	0.019299373030663	0.009406255558133	0.168648302555084	0.05143016949296
δ_{32}	0.009729047305882	0.004982729908079	0.054613891988993	0.037380788475275

- From the results shown above it can be inferred that larger and more complex neural network imporves 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 that mapping function from input to output is more complex for smaller values of **η** compared to larger values.

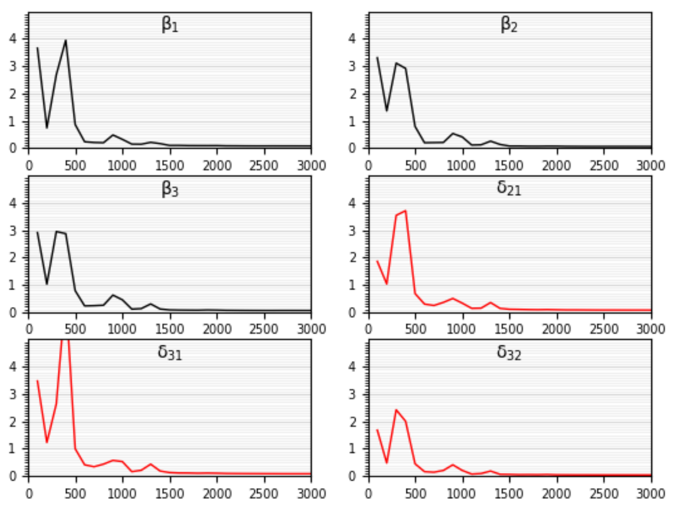
4. Models were finally trained and evaluated over the entire range of **η** ϵ [0, 10], as per expectations similar trend in percent deviations was observed as shown in tables above. Data for the two ranges was merged and used for training and evaluation. Training and validation plots for the model trained over larger number of datapoints, i.e. case 2 are shown below.

Note: For validation 10,000 datapoints were used and only 20*3,000 iterations were done using update period for learning rate to be 500, all other training parameters were same.

Mean squared percent error vs number of iterations on $\log_{10}$ scale

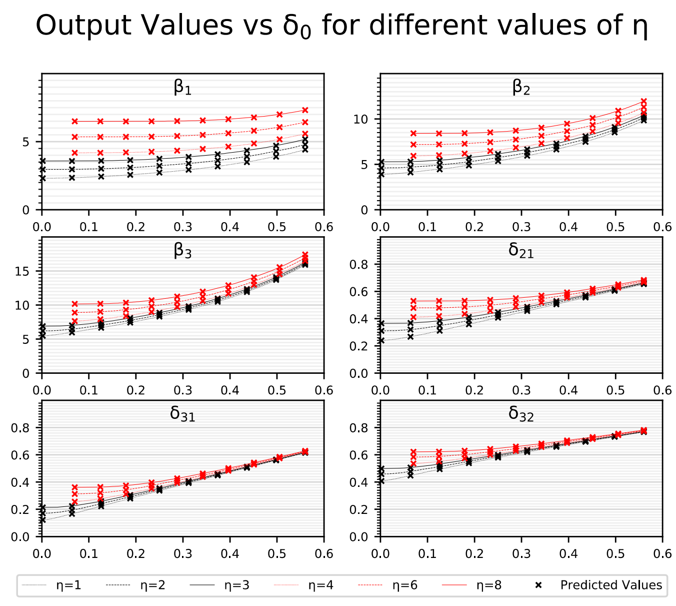


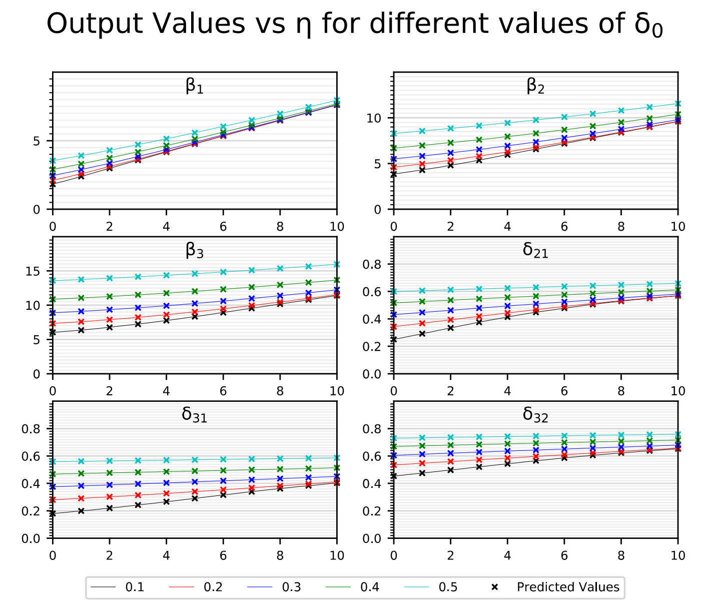
Validation set- 99th percentile percent deviation vs number of iterations

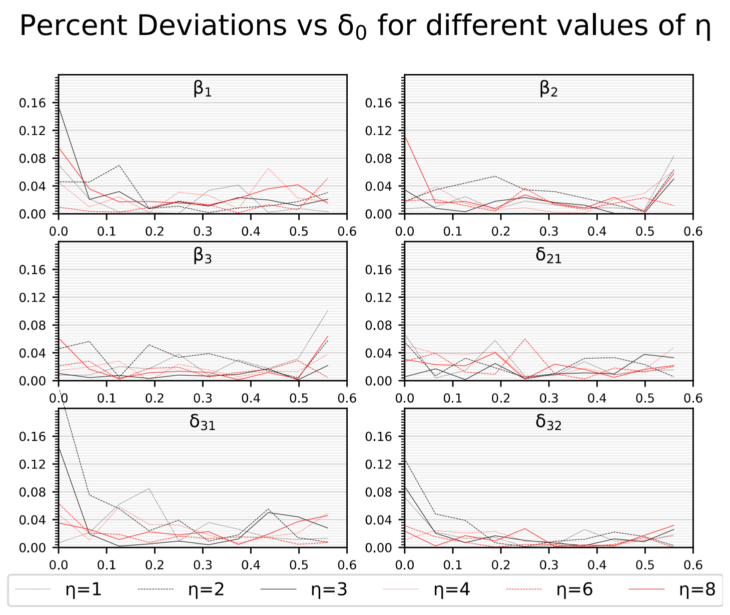


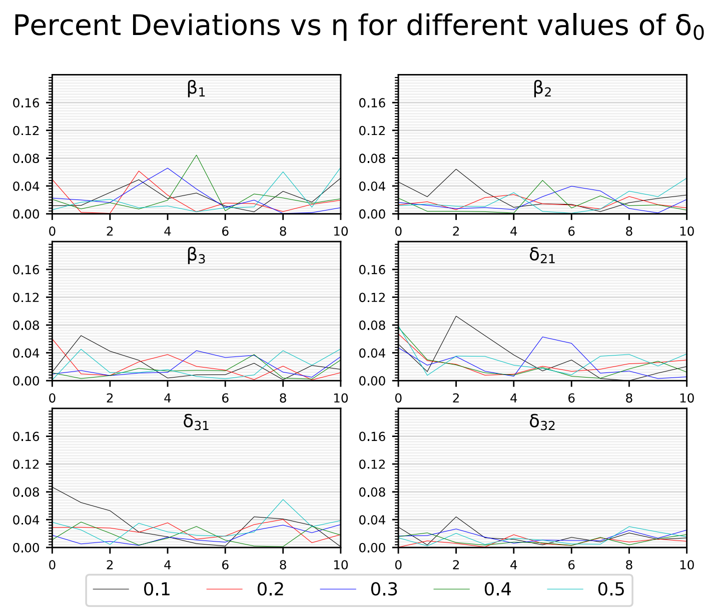
Graphical Comparison between Actual and Predicated values

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







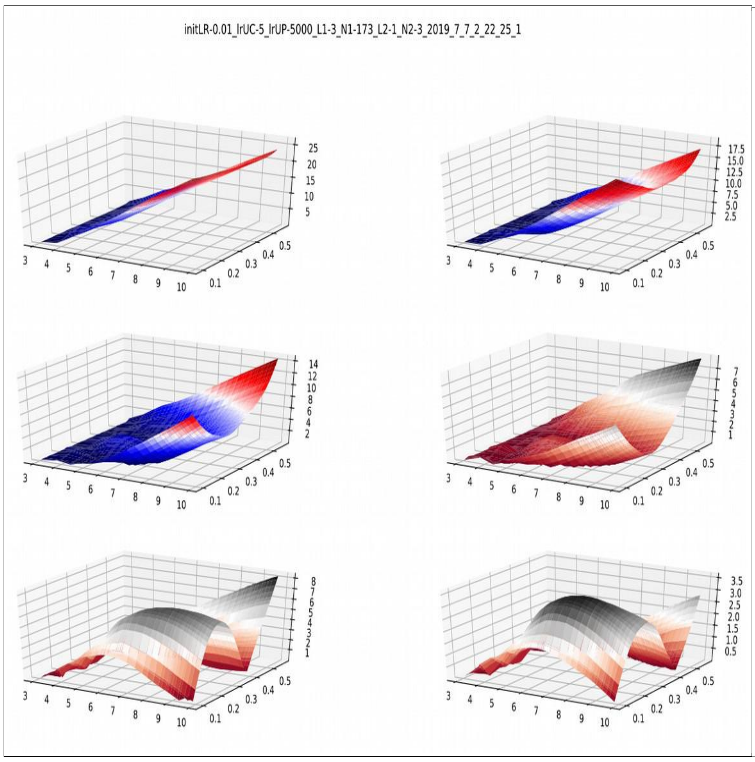


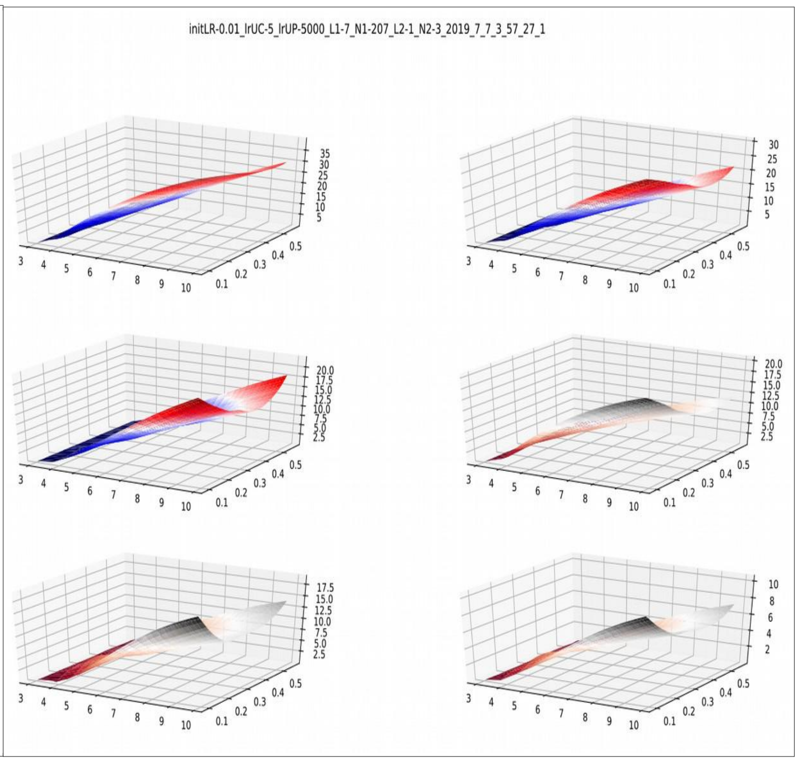
Evaluation over unseen data range

1. 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

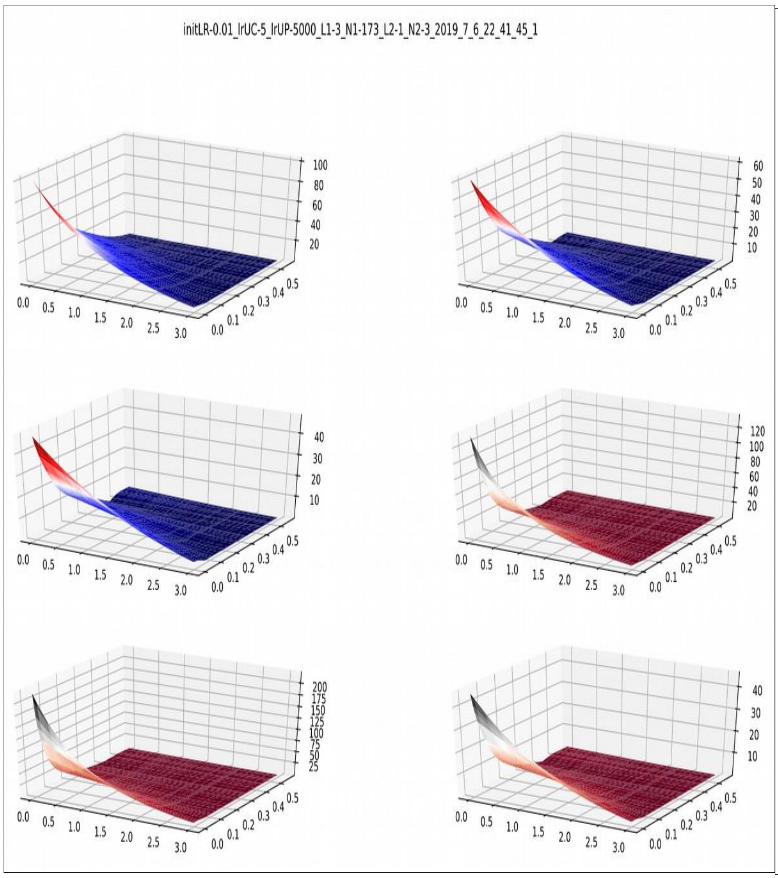
- Models obtained for case 1 and 2 were trained over the dataset generated from interval **η** ϵ [3, 10], **δ_0** ϵ [7e-2, 0.56] and evaluated on Test dataset generated for **η** ϵ [0, 3], **δ_0** ϵ [2e-3, 0.56] and vice-versa.

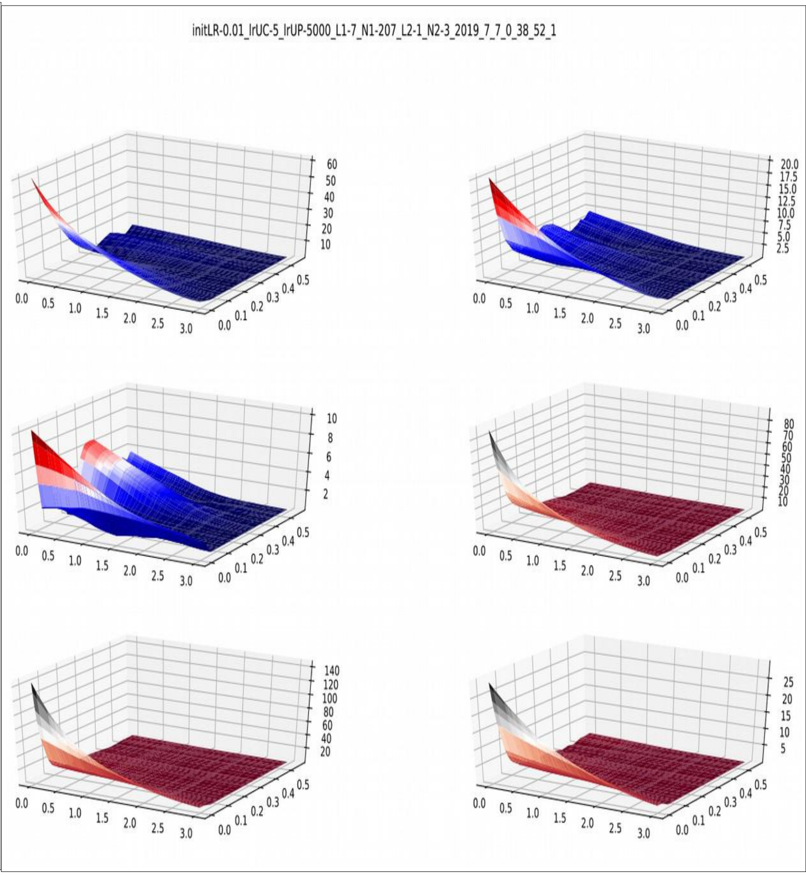
Trained over η ϵ [0, 3] and evaluated over η ϵ [3, 10]				
Output	**Mean percentage deviation**		**Max. Percent deviation**	
	Case 1	**Case 2**	**Case 1**	**Case 2**
β_1	12.9218463897705	19.3269710540771	26.6705741882324	40.0682411193848
β_2	7.08222198486328	12.5821762084961	19.4162845611572	31.1556816101074
β_3	3.73657917976379	8.60919952392578	13.8280553817749	23.1500701904297
δ_{21}	1.8658355474472	8.3175573348999	7.64275169372559	21.1663093566895
δ_{31}	2.91930770874023	7.45307874679565	7.81620454788208	20.6211223602295
δ_{32}	1.33073949813843	4.31031465530396	3.59301519393921	11.1002731323242





Trained over η ϵ [3, 10] and evaluated over η ϵ [0, 3]				
Output	**Mean percentage deviation**		**Max. Percent deviation**	
	Case 1	**Case 2**	**Case 1**	**Case 2**
β_1	10.6198329925537	4.6297755241394	94.8128280639648	57.2422027587891
β_2	6.18077898025513	1.80768477916718	58.5376930236816	19.5435829162598
β_3	4.8261022567749	1.36109101772308	45.6776428222656	10.1280870437622
δ_{21}	7.06011962890625	3.66388201713562	121.944679260254	80.4353332519531
δ_{31}	9.47541522979736	4.97512435913086	202.181198120117	134.400527954102
δ_{32}	3.17380809783936	1.40759789943695	43.3747024536133	27.9628009796143





- From the results shown in tables and plots above, we can observe that relatively smaller network with less training data when trained on interval **η** ϵ [0, 3], **δ_0** ϵ [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** ϵ [3, 10], **δ_0** ϵ [7e-2, 0.56] larger network does better over unsen data range(**η** ϵ [0, 3], **δ_0** ϵ [2e-3, 0.56]) compared to smaller network trained with lesser amount of data. These observations can also be attributed 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 **δ_0)** is small (**η** ϵ [0, 3], **δ_0** ϵ [2e-3, .56]), a bigger neural network is required to attain the accuracy comaprable to when the value of input variables is slightly big (**η** ϵ [3, 10], **δ_0** ϵ [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 attributed to more complex mapping function in smaller regime. Also, using bigger neural network for training over smaller values of n 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. Altough, 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 exrapolating in absence of 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 resonably well on other boundary conditions as well. Also, better models could be constructed to improve the extrapolation accuracy.

References

Conway HD, Dubil JF. Vibration Frequencies of Truncated-Cone and Wedge Beams. ASME. *J. Appl. Mech.* 1965;32(4):932-934. doi:10.1115/1.3627338

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

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.

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.

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

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

5) **Loss function**.

a. LF1: Sum squared percent error.

$$\sum_{j=1}^{N}\left(\sum_{i=1}^{6}\left(\frac{act_{ij}-pred_{ij}}{act_{ij}}*100\right)^2\right)$$

b. LF2: Sum absolute percent error.

$$\sum_{j=1}^{N}\left(\sum_{i=1}^{6}\left|\frac{act_{ij}-pred_{ij}}{act_{ij}}*100\right|\right)$$

c. LF3: Sum squared error without regularisation.

$$\sum_{j=1}^{N}\left(\sum_{i=1}^{6}\left|act_{ij}-pred_{ij}\right|^2\right)$$

d. LF4: Sum squared error with regularisation coefficient 400.

$$\sum_{j=1}^{N}\left(\sum_{i=1}^{3}\left|act_{ij}-pred_{ij}\right|^{2}+400*\sum_{i=3}^{6}\left|act_{ij}-pred_{ij}\right|^{2}\right)$$

e. LF5: Sum absolute error without regularisation.

$$\sum_{j=1}^{N}\left(\sum_{i=1}^{6}\left|act_{ij}-pred_{ij}\right|\right)$$

f. LF6: Sum absolute error with regularisation coefficient 20.

$$\sum_{j=1}^{N}\left(\sum_{i=1}^{3}\left|act_{ij}-pred_{ij}\right|+20*\sum_{i=3}^{6}\left|act_{ij}-pred_{ij}\right|\right)$$

Note: N is the number training examples.

act_{ij} is actual value of ith output of jth training example.

$pred_{ij}$ is predicted value of ith output of jth training example.