
Machine Learning Assisted
System Identification of a Composite Plate



Stuti Pathak
Department of Physics
Indian Institute of Technology Indore

A thesis submitted for the degree of
Master of Science

June 15, 2021

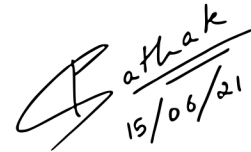


Indian Institute of Technology Indore

Candidate's Declaration

I hereby certify that the work which is being presented in the thesis entitled **Machine Learning Assisted System Identification of a Composite Plate** in the partial fulfillment of the requirements for the award of the degree of **Master of Science** and submitted in the **Discipline of Physics, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July 2020 to June 2021 under the supervision of **Dr. Ankhi Roy**, Associate Professor, Indian Institute of Technology Indore and **Dr. Rajendra Machavaram**, Assistant Professor, Indian Institute of Technology Kharagpur.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.



15/06/21

Signature of the student with date
(Stuti Pathak)

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.



15/06/2021

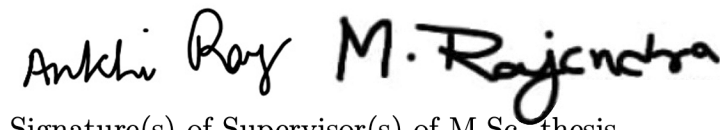
Signature of the Supervisor of
M.Sc. thesis No. 1 (with date)
(Dr. Ankhi Roy)




15-06-2021

Signature of the Supervisor of
M.Sc. thesis No. 2 (with date)
(Dr. Rajendra Machavaram)


Stuti Pathak has successfully given her M.Sc. Oral Examination held on **June 18, 2021**.




Signature(s) of Supervisor(s) of M.Sc. thesis
Date: 22/06/2021



24-06-2021
Convener, DPGC
Date:



Signature of PSPC Member No. 1
Date: 20/6/2021



Signature of PSPC Member No. 2
Date:

*Dedicated to my parents
and my brother ...*

Abstract

Parallel advancements in the fields of system identification and machine learning have been occurring for years now, but only recently have both the concerned communities realized how they have been trying to address fundamentally similar types of problems more often than not. Not only have almost all the major research fields started exhibiting a significant bias towards machine learning techniques for solving complex problems, but also a whole new market for machine learning based technology has emerged with the rise of powerful computer systems.

To summarize the research work done in this thesis, an Artificial Neural Network (ANN)-based approach has been proposed to identify the mechanical properties of an orthotropic composite plate, modelled using Finite Element Analysis (FEA) and sampled using Latin Hypercube Sampling (LHS), in frequency as well as time domain. The two networks employed for the same are Multi-Layer Perceptron (MLP) and Radial Basis Function Network (RBFN), both of which predict, to a certain degree of error, the four parameters characteristic of any composite plate: Young's moduli in two directions, Poisson's ratio, and shear modulus. Eventually, frequency domain identification using the MLP network was accepted as a much superior model as compared to the RBFN with an accuracy of 2.7% and a training time of less than 7 minutes averaged over the said parameters.

Keywords: System Identification; Finite Element Analysis (FEA); Latin Hypercube Sampling (LHS); Artificial Neural Network (ANN).

Acknowledgement

First and foremost, I would like to express my deep gratitude towards my supervisors, Dr. Ankhi Roy and Dr. Rajendra Machavaram, without whose guidance and continuous support it would have been impossible to get a more profound understanding of the subject. It has been a huge privilege to work under them for my thesis and it goes without saying how much their knowledge, supervision and encouragement has helped me grow, not only in academics but also as a person. They, as mentors, gave the most invaluable suggestions and boosted my morale whenever I faced any kind of dispiriting hurdles and for that I am indebted to them forever.

To have Dr. Sudip Chakraborty and Dr. Debajyoti Sarkar as my committee members has been the most humbling experience ever. Their inspiration and motivation has played a major role in the development of this thesis and for that I am extremely grateful to both of them. I am immensely thankful to the entire faculty of the Department of Physics at IIT Indore for all the courses which I have been taught during my degree and for conducting all my seminars in such a smooth and efficient manner.

This thesis has been written during one of the most difficult times the whole world has ever faced, and the entire credit for its successful completion goes to my family and friends. It is very hard to put into words how much my parents have sacrificed to provide for me and my education. I do not think I would ever be able to thank them enough. My brother and my parents have always motivated me to

look at the bigger picture and strive for nothing but perfection in every endeavour.

My stay at IIT Indore could not have been more thrilling and thoroughly enjoyable, all thanks to to my dearest friends, especially my roommates. They have always helped me maintain the much needed work-life balance. I have been really fortunate to have made so many friends during these two years and this thesis is as much an ode to everything about life that I have learnt from them as it is to the great amount of hard work that I have put into it.

Contents

Abstract	i
Acknowledgement	ii
Contents	iv
List of Tables	vi
List of Figures	vii
List of Algorithms	viii
1 Introduction	1
2 System Identification	5
2.1 Stepwise breakdown of methodology	6
2.2 Classification of system identification techniques	7
3 Finite Element Analysis (FEA)	9
3.1 Stepwise breakdown of methodology	10
3.2 Newmark integration	14
4 Sampling Techniques	17
4.1 Simple Random Sampling (SRS)	18
4.2 Latin Hypercube Sampling (LHS)	19
5 Artificial Neural Networks (ANNs)	22

5.1	Maximum Likelihood Estimation (MLE)	24
5.2	Gradient descent: An optimization technique	26
5.3	Advanced optimization techniques	27
5.4	Perceptron: A single layer neural network	32
5.5	Multi-Layer Perceptron (MLP)	34
5.6	Radial Basis Function Network (RBFN)	40
6	Results and Discussion	45
7	Summary and Conclusions	52
	Bibliography	54

List of Tables

6.1	Table showing the performance the proposed three layer MLP with 200K sample size.	49
6.2	Table showing the performance the proposed RBFN with 200K sample size.	50
6.3	Table showing the average percentage error calculated over a larger test dataset for different networks with different sample sizes.	51

List of Figures

2.1	Block diagram representation of the process of system identification. .	6
3.1	Slicing and discretization of the composite plate into 4 slices and 16 elements.	11
4.1	Stratified sampling in different dimensions.	20
5.1	Block diagram representation of how weights are updated in case of Stochastic Gradient Descent (SGD) with different batch sizes.	29
5.2	The building-block of neural networks, perceptron.	32
5.3	Different types of activation functions used in neural networks.	35
5.4	Three layer Multi-Layer Perceptron (MLP).	37
5.5	Different types of Radial Basis Functions for one-dimensional input space.	41
5.6	Radial Basis Function Network (RBFN).	42
6.1	Acceleration response of the composite plate for two sets of parameter values for the first 0.6 s.	46
6.2	Learning curves of the proposed three layer MLP for the four parameters E_1 , E_2 , ν_{12} and G_{12} for two sample sizes 2K and 200K.	49
6.3	Learning curves of the proposed RBFN for the four parameters E_1 , E_2 , ν_{12} and G_{12} for two sample sizes 2K and 200K.	50

List of Algorithms

1	Algorithm for Finite Element Analysis (FEA) of a composite plate.	12
2	Algorithm for Newmark integration method.	15
3	Algorithm for stratified sampling.	20

Chapter 1

Introduction

Even before the term *composite* was coined, common man, although unconscious of the science behind the remarkable characteristics of composite materials, had already put them to great use. The blades of Japanese sabers forged from steel and soft iron showed better resistance to impact as opposed to pure steel swords. For construction purposes, Israelites reinforced clay bricks with straw as the straw fibers majorly improved the structure's strength. Wood was replaced with plywood by Egyptians for wood frameworks in order to avoid expansion and swelling, to the maximum possible extent, in extreme thermal and humid conditions.

A composite material is formed when two or more two materials are combined on a macroscopic level to achieve qualities superior than its constituent materials ([Jones, 1999](#)). Major revolution was seen in automobile, marine and aerospace industries when composite materials started replacing conventionally reliable materials such as steel and aluminium. Their great resistance to corrosion, low wear and tear as well as high strength-to-weight ratio made them exceptional materials to be used in aircraft and buildings as opposed to traditionally used homogeneous materials.

Composites are tailor-made and are specifically designed for the task at hand.

A number of ingredients and recipes can be used to prepare them and for this very reason, generally, composite materials are both inhomogeneous and orthotropic (or anisotropic). Not only this, the mechanical properties of composites are significantly affected during the manufacturing process and thereafter the properties of the original composite are altered altogether. This makes the task of determining the mechanical properties of composites, both efficiently and accurately, an active research area in today's date.

These mechanical properties can be measured using direct methods like tensile test, four-point bending test, or by vibration-based indirect methods which can be further sub-categorized as follows: classical methods derived from mathematical statistics, and non-classical methods based on heuristic concepts. Least squares method, maximum likelihood method and extended Kalman filter method come under the umbrella of classical methods while evolutionary algorithms such as Particle Swarm Optimization, and Artificial Neural Networks come under non-classical methods (Imai et al., 1989; Koh et al., 2003).

Time and again researchers encounter problems where ample amount of input and output data from a system is known to them but the actual model of the system is completely unknown. Similarly, above mentioned techniques essentially use the dynamic response of an unknown system to determine its properties, and are therefore grouped under a major subject of research, in almost all engineering and scientific fields, called system identification.

Lying at the intersection of computer science and statistics, machine learning is one of the most flourishing technical fields nowadays. Image classification, spam detection, speech recognition, stock market trading, drug discovery and medical diagnosis can be considered as only the tip of the iceberg when it comes to the applications of machine learning (Witten et al., 2011; Crawford et al., 2015; Patel et al., 2015; Lavecchia, 2015). Machine learning as a technique has received

even more impetus in the recent years owing to the increasing data availability and economical high-speed computational resources.

Not only are the traditionally used classical system identification paradigms not much reliable for multi-degree-of-freedom systems but also they are computationally very expensive. Here comes the need to explore this problem using various non-classical techniques such as hybrid response surface methodology and particle swarm optimization (Sankar et al., 2014) and convolutional neural networks (Ye et al., 2019; Abueidda et al., 2019), which have recently proved to be very handy in predicting the mechanical properties of composite materials. Pillonetto et al. (2014) have also surveyed machine learning assisted system identification techniques for linear systems.

The motivation behind the investigation presented in this thesis is given as follows:

1. When a structure made up of composite materials gets damaged, the mechanical properties of the structure get significantly altered, which in turn alters the structure's vibration response. If some reliable techniques are available to measure these structural properties efficiently using the vibration response, countless hazardous situations can be anticipated, and hence avoided. A quick regular monitoring of these mechanical properties can eliminate such risks which have the potential of causing major danger to economy as well as life.
2. From the design perspective, it is important to know whether a structure can withstand certain environmental conditions or not. If the structural properties are known one can easily calculate the structure's strength and hence its breaking point. Again, a good parameter identification technique definitely assists this process.
3. The classical methods described above are extremely susceptible to converging

to a local minima instead of a global one because their implementation involves point-to-point search technique.

The thesis predominantly involves three main objectives:

1. To simulate the response of a composite plate in both frequency and time domain using Finite Element Analysis (FEA) and Newmark integration, and then sample the input-output signals using Latin Hypercube Sampling (LHS).
2. To develop two neural network models: Multi-Layer Perceptron (MLP) and Radial Basis Function Network (RBFN) for system identification of the composite plate. This involves accurate prediction of the following four properties of any orthotropic composite plate: Young's moduli in two directions (E_1 and E_2), Poisson's ratio (ν_{12}), and shear modulus (G_{12}).
3. To compare the performance of the two said models in terms of prediction accuracy and the computational effort required.

Organization of this thesis has been done in the following manner. The importance of system identification as a technique and its classification has been discussed in [Chapter 2](#). In [Chapter 3](#), the entire procedure of modelling a composite plate, to generate its set of natural frequencies and its forced acceleration response, has been described in a systematic manner using FEA and Newmark integration. [Chapter 4](#) presents a variety of sampling techniques, out of which the methods for Simple Random Sampling and LHS have been given in depth. The basic concepts behind Artificial Neural Networks have been dealt with comprehensively in [Chapter 5](#), and two networks: MLP and RBFN have been described thoroughly. The whole research work has been summarized in [Chapter 6](#) along with the results and the future direction. Finally, in [Chapter 7](#) all the important observations and conclusions drawn from the investigation conducted have been discussed in detail.

Chapter 2

System Identification

Drawing valuable inferences from the observations made while studying dynamic systems have led to revolutionary discoveries in the field of science and technology over and over again. Right from our childhood, even before we acquire any knowledge about science and how the nature around us works, we as humans are accustomed to making observations and most certainly drawing conclusions. It is basically one of those things which comes without saying. However, in strict scientific terms, system identification is finding solutions to scientific problems by developing mathematical models while studying the inputs and the corresponding outputs of the system under consideration as has been summarised in [Figure 2.1](#). Least square method is one the earliest forms of system identification techniques. Later techniques such as maximum likelihood method and extended Kalman filter were introduced which proved to be much more efficient than the least square method.

Among the various types of identification problems, there exists a set of problems where the primary goal is to find the characteristics of an unknown system and may involve determining one or more parameters of the said system. This summarises the main objective behind this thesis, which will be more and more

apparent in the upcoming chapters.

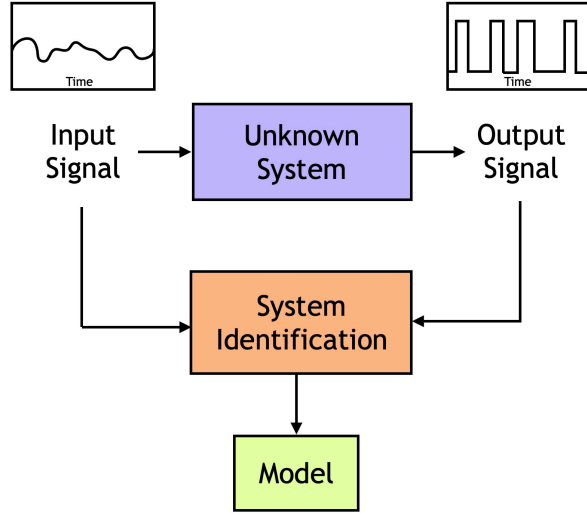


Figure 2.1: Block diagram representation of the process of system identification.

2.1 Stepwise breakdown of methodology

The formal introduction of system identification as major field was done by the works of [Åström and Wittenmark \(1995\)](#), and [Ljung \(1987\)](#). According to Lennart Ljung, the system identification procedure predominantly comprises of the following three steps:

1. Recording the input and output data from the unknown system.
2. Designing a set of appropriate models.
3. Selecting the most efficient model among the set.

To make it easier to stick to the flow of this thesis, it is extremely crucial to mention here that these three steps have been realised one after the other in the forthcoming chapters. To simulate the input and output signals of our dynamic system, which is the first step, we have used dynamic Finite Element Analysis

(FEA) along with Newmark integration as given in [Chapter 3](#). Also, sampling of this input and output data has been done using Latin Hypercube Sampling (LHS) presented in [Chapter 4](#). The second step above has been realized by designing two types of Artificial Neural Networks: Radial Basis Function Network and Multi-Layer Perceptron which have been discussed in [Chapter 5](#). And finally, model efficiencies of both the networks have been discussed in [Chapter 6](#).

2.2 Classification of system identification techniques

System identification is categorized into various types: parametric and non-parametric system identification; black box, white box and grey box system identification; time domain and frequency domain system identification; classical and non-classical system identification.

Non-parametric system identification results in curves, tables, etc. which are characteristic of the system, while *parametric* system identification involves determining the system parameters themselves. Although parametric system identification is a more accurate and reliable technique as compared to the non-parametric system identification, it is a lot more computationally expensive as compared to the later.

In *white box* system identification, a great deal about the system is known beforehand for example the underlying equations, etc. whereas in *black box* system identification, no characteristic information of the system is known and the model is purely derived from the observations made. As obvious, *grey box* system identification is a mixture of the two.

When the data provided by the system is in time domain, *time domain* system identification is employed, however if observations are made in frequency domain we

use *frequency domain* system identification. In time domain, the system's dynamic response is used, whereas in frequency domain, modal natural frequencies, modal shapes and damping ratios of the system are used.

The last category has been already discussed in the introduction which is *classical* system identification and *non-classical* system identification. Slowly but surely non-classical methods of system identification such as evolutionary algorithms and neural networks have started replacing the classical methods.

Chapter 3

Finite Element Analysis (FEA)

Not every system in our surroundings is as simple to understand as a ball rolling down a hill or a mass hanging from a spring. More or less, all of the systems that we encounter in our daily lives behave in such a complex manner that it is rather unreasonable to study them as a whole. A better way to approach such real life complex problems is to first subdivide them into smaller components, whose behaviour is much easier to understand, then use this knowledge about these individual components' characteristics to reconstruct our original system, and finally explore its characteristics as well. This method of simplifying the analysis of such elaborate systems is called Finite Element Analysis (FEA), or more commonly Finite Element Method (FEM). It has not only become general practice among scientists nowadays in almost all engineering and scientific fields, but also is readily used by economists for modelling purposes.

FEA can be seen as a computational technique which is used to find solutions to field problems, also called boundary value problems. A sufficiently good solution of a boundary value problem can be obtained by using finite number of such well-defined components. This is termed as *discretization*. With the rise of computers,

it has become quite convenient to use FEA for modelling complex structures.

In the field of solid mechanics, [McHenry \(1943\)](#), [Hrennikoff \(1941\)](#), [Newmark \(1949\)](#), and [Southwell \(1946\)](#) in the early 1940s were the first to show how an approximate solution of a continuous problem can be obtained by the discretization of complex system. However, the first formal introduction to the term *finite element* was done by [Clough \(1960\)](#) in his work on complex plane elasticity problems. A detailed mathematical outline of this method has been presented in the following sections.

3.1 Stepwise breakdown of methodology

Regardless of the type of system we choose to model, FEA as a technique majorly involves a few universal steps as are given below:

1. Discretization: The first step is to subdivide the solution space into finite number of elements. These elements can be of any shape and even the same solution space can be divided into a variety of shapes.
2. Choosing interpolation function: Next step is to assign nodes to each element and then select the interpolation function or the shape function which approximates the value of the field variable over an element. Generally, interpolation functions are chosen to be polynomials because of their simple differentiation and integration properties.
3. Determining individual element properties: In this step, matrix equations of individual element properties are determined using either direct approach or variational approach.
4. Assembling: Finally, the original system is reconstructed by assembling the

properties of the individual elements it was divided into. This is done by combining the matrix equations found out in the previous step for individual element behaviour to form the matrix equations for the entire original system.

5. Applying boundary conditions: Next step is to apply the boundary conditions of the given problem.
6. Solving equation of the system: In the final step, the equations of the system are solved. These equations can be linear or nonlinear, algebraic or differential depending on whether the unknowns are dependent on time or not. Once solved, the results can be studied in more understandable forms such as plots and curves.

In this project, FEA has been used to model an in-plane bending rectangular orthotropic plate and to determine its frequency and acceleration response to an external sinusoidal force.

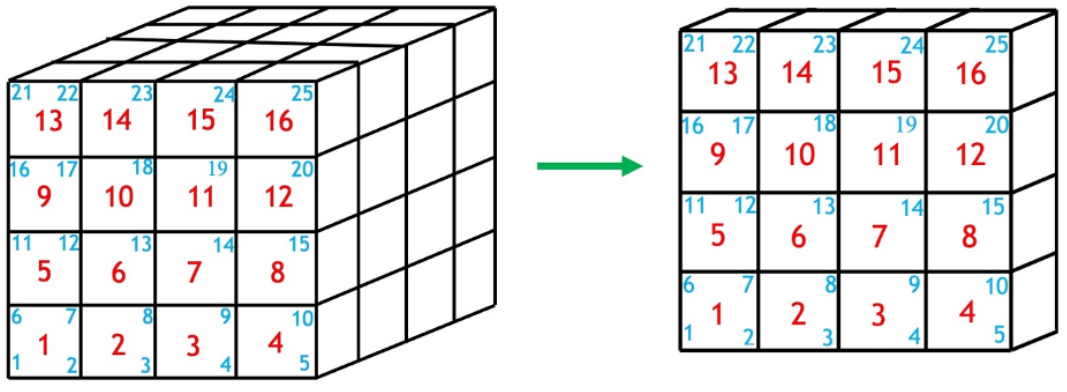


Figure 3.1: Slicing and discretization of the composite plate into 4 slices and 16 elements.

The plate was divided into 16 equal elements with each element having 4 nodes and each node having 3 degrees-of-freedom (DOFs) as shown in [Figure 3.1](#). All the 4 boundaries of the plate were fixed as boundary conditions. This implies nodes [1, 2, 3, 4, 5, 6, 10, 11, 15, 16, 20, 21, 22, 23, 24, 25] were fixed nodes. Hence, in

total there were 16 elements, 25 nodes, 75 total DOFs, 16 fixed nodes, 48 fixed DOFs, and 27 variable DOFs. A brief walk-through of the process of FEA modelling has been discussed below in a very simple manner:

Algorithm 1: Algorithm for Finite Element Analysis (FEA) of a composite plate.

Input: Values of Young's moduli in two directions (E_1 and E_2), Poisson's ratio (ν_{12}), shear modulus (G_{12}), density of the plate (ρ), length of the plate (L_x), width of the plate (L_y), and thickness of the plate (h).

Output: Frequency response of the composite plate.

1. Discretization of the composite plate is done by constructing a mesh. The plate is divided into 16 small elements by sectioning the plate into 4 equal parts in the x-direction as well as in the y-direction. Also, the plate is sliced in 4 equal parts, each of thickness t along the thickness.
2. An array of dimensions 16×4 is constructed where the four entries of each row (i^{th} row) are the four nodes of the corresponding element (i^{th} element) counted in anticlockwise direction. For example, the 3^{rd} row for the 3^{rd} element is $[3, 4, 9, 8]$.
3. Another array of dimensions 25×2 is constructed where the two entries of each row (i^{th} row) are the two length coordinates (x-coordinate and y-coordinate) of the corresponding node (i^{th} node) with the 1^{st} node considered to be at the origin. For example, the 3^{rd} row for the 3^{rd} node is $[0.121375, 0]$.
4. Two more arrays of dimensions 1×48 and 27×1 are constructed. The first array contains the indices of all the fixed DOFs while the second array contains the indices of all the variable DOFs. Here, each node is assigned with 3 indices for the 3 DOFs corresponding to it. These arrays will be used later for simulating the acceleration response of the plate.
5. This step realizes one of the main results of the FEA modelling which is the assembly of the global stiffness and the global mass matrices $[K_G]$ and $[M_G]$ respectively of the plate using the elementary stiffness and elementary mass matrices $[K_e]$ and $[M_e]$ (their expanded forms) as given below:

$$[K_e] = \int_{z=0}^t \int_{y=0}^{l_y} \int_{x=0}^{l_x} [B]^T [Q] [B] dx dy dz ; \quad [K_G] = \sum_{e=1}^{16} [K_e]$$

$$[M_e] = \int_{z=0}^t \int_{y=0}^{l_y} \int_{x=0}^{l_x} [N]^T \rho [N] dx dy dz ; \quad [M_G] = \sum_{e=1}^{16} [M_e]$$

where, $[B]$ is the strain-displacement matrix, $[Q]$ is the constitutive matrix and $[N]$ is the shape function.

6. Boundary conditions are applied and the rows and columns corresponding to fixed nodes in $[K_G]$ and $[M_G]$ are removed.
7. Natural frequencies and mode shapes of the structure are calculated by solving the following eigenvalue equation:

$$([K_G] - \lambda [M_G]) \phi = 0$$

where, λ is the eigenvalue or the square of the natural frequencies (ω^2) and ϕ is the eigenvector or mode shapes.

8. Global damping matrix $[C_G]$ is modelled by using Rayleigh damping:

$$[C_G] = a [M_G] + b [K_G]$$

where, a and b are Rayleigh damping coefficients which can be determined using the following equation at two natural frequencies and taking assumed damping ratios:

$$\xi_r = \frac{a}{2\omega_r} + \frac{b\omega_r}{2}$$

where, ω_r is natural frequency of the r^{th} mode and ξ_r is the assumed damping ratio.

9. Frequency response of the system (composite plate) is obtained by the following equation:

$$f = \frac{\omega}{2\pi}$$

3.2 Newmark integration

Newmark integration method, developed by [Newmark \(1959\)](#), is a direct numerical integration method which implies that the equations are not transformed into any other form before carrying out the integration. In this method, velocity and displacement are given by the following equations:

$$\dot{U}_{t+\Delta t} = \dot{U}_{t+\Delta t} + \left[(1 - \delta)\ddot{U}_t + \delta\ddot{U}_{t+\Delta t} \right] \Delta t \quad (3.1)$$

$$U_{t+\Delta t} = U_t + \dot{U}_t \Delta t + \left[\left(\frac{1}{2} - \alpha \right) \ddot{U}_t + \alpha \ddot{U}_{t+\Delta t} \right] \Delta t^2 \quad (3.2)$$

where, α and β are Newmark integration parameters, and U_t , \dot{U}_t and \ddot{U}_t are the displacement, velocity and acceleration vectors at time t .

Also, along with the above two equations, equation of motion at time $t + \Delta t$ as given below is also used to reach the solution:

$$M\ddot{U}_{t+\Delta t} + C\dot{U}_{t+\Delta t} + KU_{t+\Delta t} = F_{t+\Delta t} \quad (3.3)$$

Since the equation of motion ([Equation 3.3](#)) is combined with the velocity and acceleration approximations ([Equation 3.1](#) and [Equation 3.2](#)) to find the solution, Newmark integration method is an implicit technique of direct integration.

This method of integration is unconditionally stable for the following two conditions:

$$\delta \geq \frac{1}{2} ; \quad \alpha \geq \frac{1}{4} \left(\frac{1}{2} + \delta \right)^2 \quad (3.4)$$

The most commonly used values for these integration parameters are $\alpha = 0.25$ and $\delta = 0.5$, and the method is called the *constant average acceleration* method.

To obtain the acceleration response of a composite plate the following system

of second order linear differential equations need to be solved:

$$M\ddot{U} + C\dot{U} + KU = F \quad (3.5)$$

where, M is the mass matrix, C is the damping matrix, K is the stiffness matrix, F is the driving force vector, U is the displacement vector, \dot{U} is the velocity vector and \ddot{U} is the acceleration vector of the assembly of finite elements. To solve this, Newmark integration has been used. A time step of 0.001 s is taken. Sinusoidal force of $20 \sin(2\pi 50t)$ Newton is applied along the z-direction at 13^{th} node of the composite plate for 6 seconds. Initial displacement and initial velocity are taken to be zero and the acceleration response of the plate at 14^{th} node along z-direction is obtained for a time period of 6 seconds. The following algorithm shows in detail the process of Newmark integration ([Bathe, 2006](#)):

Algorithm 2: Algorithm for Newmark integration method.

Input: Mass matrix (M), damping matrix (C), stiffness matrix (K) of the dynamic system, time vector for acceleration response (t), initial displacement vector ($U_{t=0}$), initial velocity vector ($\dot{U}_{t=0}$), and initial driving force vector ($F_{t=0}$).

Output: Final desired acceleration vector ($\ddot{U}_{t=T}$) of the system.

1. Initial calculations:

- (a) Form K , M and C matrices.
- (b) Initialize U_0 , \dot{U}_0 and \ddot{U}_0 .
- (c) Choose appropriate Δt , α and δ satisfying:

$$\delta \geq \frac{1}{2} ; \quad \alpha \geq \frac{1}{4} \left(\frac{1}{2} + \delta \right)^2$$

- (d) Calculate the following:

$$a_0 = \frac{1}{\alpha \Delta t^2} ; \quad a_1 = \frac{\delta}{\alpha \Delta t} ; \quad a_2 = \frac{1}{\alpha \Delta t} ;$$

$$a_3 = \frac{1}{2\alpha} - 1 ; \quad a_4 = \frac{\delta}{\alpha} - 1 ; \quad a_5 = \frac{\Delta t}{2} \left(\frac{\delta}{\alpha} - 2 \right) ;$$

$$a_6 = \Delta t (1 - \delta) ; \quad a_7 = \delta \Delta t$$

(e) Form effective stiffness matrix:

$$\hat{K} = K + a_0 M + a_1 C$$

2. For each time step:

(a) Calculate effective force vector at time $t + \Delta t$:

$$\hat{F}_{t+\Delta t} = F_{t+\Delta t} + M \left(a_0 U_t + a_2 \dot{U}_t + a_2 \ddot{U}_t \right) + C \left(a_1 U_t + a_4 \dot{U}_t + a_5 \ddot{U}_t \right)$$

(b) Calculate displacements at time $t + \Delta t$:

$$\hat{K} U_{t+\Delta t} = \hat{F}_{t+\Delta t}$$

(c) Calculate accelerations and velocities at time $t + \Delta t$:

$$\ddot{U}_{t+\Delta t} = a_0 (U_{t+\Delta t} - U_t) - a_2 \dot{U}_t - a_3 \ddot{U}_t$$

$$\dot{U}_{t+\Delta t} = \dot{U}_t + a_6 \ddot{U}_t + a_7 \ddot{U}_{t+\Delta t}$$

Chapter 4

Sampling Techniques

Long before machine learning caught the attention of scientists and proved its worth for approximating functions, complex problems which are experimentally impossible to solve, especially fluid flow problems, were solved numerically to get an approximate solution. Even the calculation of a numerical solution computationally results in a very complicated code and this code takes several inputs and outputs to gather enough information required to model the problem much similar to the training dataset that is provided to a neural network. So, to model the problem not only should the dataset be a good representative of the function we want to model, but also it should be small enough for making the code computationally less expensive. This is called sampling, which in broader terms implies selecting a smaller subset from a large population so as to make a good statistical estimate of some desired attributes of the population, and it makes the modelling process less expensive as compared to using data from the whole population. Several such sampling techniques have been introduced over the years for selecting these input datasets as have been presented below.

These sampling methods are generally classified into two types: probability

sampling or random sampling, and non-probability sampling or non-random sampling. Just as their names suggest, in *probability sampling*, each individual has an equal probability to get selected in the sample, making it just a matter of chance, however, in *non-probability sampling*, sampling is not performed randomly because this kind of sampling is expected to study real-life phenomena and not to determine a statistical estimate of the entire population. Simple random, stratified random, systematic sampling, cluster sampling, and multi-stage sampling are some random sampling methods, whereas quota sampling, judgement sampling, snowball sampling, and convenience sampling come under non-random sampling methods.

In the coming sections, two of the most widely used probability sampling techniques: simple random sampling and latin hypercube sampling, a type of stratified sampling, have been discussed comprehensively.

4.1 Simple Random Sampling (SRS)

Before we try to understand some sampling methods, rather than stating in layman terms like we did in the beginning of this chapter, an introduction to some common terms frequently used in literature is called for. A single element or a group of elements which can provide useful observations is called a *sampling unit*. An assembly of all the sampling units at a certain point of time is called the *population*. A collection of one or more sampling units chosen from the population is called a *sample*. The process of determining a good representative sample from a population is called *sampling*.

As has been already mentioned, in simple random sampling method there is no bias for any sampling unit whatsoever, which gives every sampling unit equal opportunity to be selected in the sample. However, this method can be again subdi-

vided into two categories. When choosing a sample of m units out of a population of M units, one at a time, if the chosen units are never placed back again into the population then it is called *SRS without replacement*, whereas if each unit after being chosen is placed back into the population then it is called *SRS with replacement*.

4.2 Latin Hypercube Sampling (LHS)

To investigate and study an attribute of a population it is extremely important to know beforehand whether the population is homogeneous or heterogeneous with respect to that particular attribute. Also, as already mentioned, to avoid high computational time while modelling a system it is always advised to keep the sample size as small as possible. In case the population is homogeneous, SRS can produce a representative sample with a comparatively smaller sample size, but fails miserably when it is heterogeneous and the sample size is kept small. The main idea behind stratified sampling is to divide a heterogeneous population into smaller subpopulations so that each subpopulation, called *stratum*, is now homogeneous with respect to the desired attribute, and then the collection of samples generated from each such stratum will be a representative sample of the whole population.

Latin hypercube sampling was proposed by [McKay et al. \(1979\)](#) in his paper in which different sampling methods are compared for sampling input dataset for creating a mathematical model. LHS is a stratified sampling method which serves basically as an extension to a Latin square ([Raj, 1968](#)) in any number of dimensions.

For one dimension only, to perform LHS, the input cumulative probability distribution is divided into equal intervals and one and only one sample is randomly taken from each interval ([Figure 4.1\(a\)](#)) and hence the number of intervals the distribution is to be divided into is equal to the number of samples required to sample

the whole population. Similarly, in two dimensions, a Latin square is constructed which is essentially a stratified square grid with one and only one sample in each row and in each column (Figure 4.1(b)) making the stratified sample size to be equal to the number of intervals along any axis.

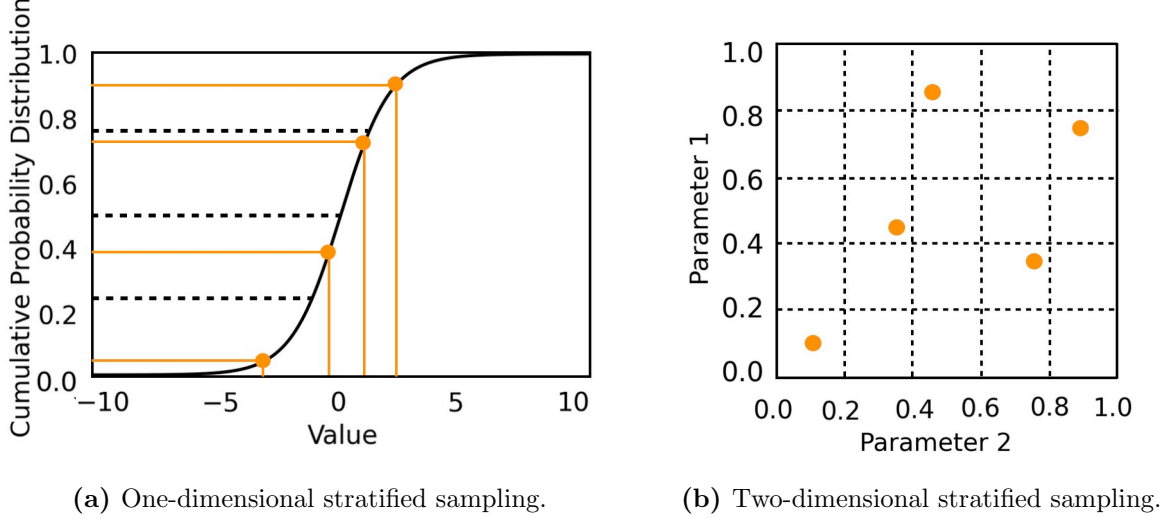


Figure 4.1: Stratified sampling in different dimensions.

Advantage of LHS over random sampling is that we need to use large sample size in case of random sampling to get a good picture of the input distribution while in LHS a rather smaller sample size is capable enough to give a good representation and this in turn reduces the computational time required to build a model by a huge amount. Monte Carlo simulations, which are popularly used for solving complex problems statistically, run much faster if the input distribution is sampled using LHS. An elaborate algorithm for stratified sampling is given below:

Algorithm 3: Algorithm for stratified sampling.

Input: An input dataset which needs to be sampled, minimum and maximum values of all the parameters of the dataset and the number of strata.

Output: A stratified sample of the input dataset.

1. Divide the population containing M sampling units into j strata. such that:

- (a) no stratum overlaps with any other stratum.
 - (b) the i^{th} stratum has M_i number of sampling units where $i = 1, 2, 3, \dots, j$ which implies $\sum_{i=1}^j M_i = M$.
 - (c) each stratum is in itself homogeneous with respect to the attribute under study.
2. Using simple random sampling, generate a sample from each stratum independently such that the size of the sample from i^{th} stratum is m_i .
 3. Generate a collective stratified sample for the whole population of size $m = \sum_{i=1}^j m_i$, by merging all the individual samples drawn from each stratum.
-

Chapter 5

Artificial Neural Networks (ANNs)

Over the years scientists have thought of several ways to make computers evolve by experience, that is, to make them learn from data, without relying completely on hard-coded software designed by humans. They believed that if somehow computers could be made to achieve even a fraction of human level intelligence, then they would have the potential to revolutionize the current state of each and every existing field, be it medical, technical, scientific, industrial, financial, etc. And it has not been long since their intense belief has started turning into reality owing completely to the rise of the field of machine learning. The science of making machines which can automatically learn from experience is called machine learning.

Machine learning has emerged as an amalgamation of a bunch of other fields: computer science, statistics and probability, biology, control theory, and many more. The ever growing field encompasses uncountable number of different algorithms to model input data: k-means: a clustering algorithm, Support Vector Machines (SVM): an instance-based algorithm, random forest: an ensemble algorithm, naive

bayes: a Bayesian algorithm, linear and logistic regression: regression algorithms, Multi-Layer Perceptron (MLP) and Radial Basis Function Network (RBFN): Artificial Neural Network (ANN) algorithms, etc.

There is already a lot to talk about machine learning concepts, however it must be briefly mentioned that machine learning techniques are broadly classified into three categories: supervised learning, unsupervised learning and reinforcement learning. While in *supervised learning* the model trains on a labelled dataset, in *unsupervised learning* a dataset without any labels is provided to the model for training. *Reinforcement learning* is when a model learns to maximize its reward by taking appropriate measures in complex circumstances.

At present, while working with extremely complex datasets, one of the most widely preferred machine learning frameworks are Artificial Neural Networks (ANNs), simply because their architecture is such that it allows them to recognize complex patterns in the dataset pretty accurately, eventually leading to much precise predictions of real-valued as well as discrete-valued data, as compared to other frameworks.

The term *neural network* came from the numerous studies conducted to understand the mathematics behind the information processing in a nervous system (McCulloch and Pitts, 1943; Rosenblatt, 1962). A nervous system can be fundamentally defined as a densely-connected network of neurons, and this particular definition makes it somewhat analogous to the modern day neural network. The thought of being able to replicate a biological system was quite fascinating on its own, and was solely enough to make the study of neural networks gain the much required momentum in the beginning. Although, artificial neural networks have not evolved enough to be capable of performing all the complex functions of a neurological system, they already have countless real life applications in today's date.

5.1 Maximum Likelihood Estimation (MLE)

A very accurate definition for what learning means for a computer program has been presented by [Mitchell \(1997\)](#) in his book: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ”. For example, for a binary image classification problem with two classes: cats and dogs, the task T is to identify whether a given image is of a dog or a cat, performance measure P can be the fraction of images correctly recognized, and the experience E is a collection of labelled cat and dog images.

Although the performance of a machine is evaluated simultaneously along with the learning process, the data used for the two tasks should not be the same, but should be drawn from the same distribution. The dataset used for learning is called the *training data* while the one used to evaluate performance is called the *testing data*. The primary role of the machine is to fit unseen data accurately. This is called *generalization*. For the machine to generalize over the testing data it is extremely important to assume the complete dataset to be *independent and identically distributed* (i.i.d.) which means that all the data points in the dataset are drawn independently and are from the same distribution ([Bishop, 2006](#)). In this case the probability of our i.i.d dataset $\mathbf{X} = (x_1, x_2, \dots, x_N)^T$ with N number of data points all drawn from a Gaussian distribution with mean μ and variance σ^2 can be given as:

$$p(\mathbf{X}|\mu, \sigma^2) = \prod_{n=1}^N \mathcal{N}(x_n|\mu, \sigma^2) \quad (5.1)$$

Random variables can be drawn from a number of different distributions such as bernoulli, exponential, uniform, normal, etc. Each such distribution has its own set of parameters, for example, a normal distribution has two parameters mean μ and variance σ^2 . While understanding the key concepts of statistics it is generally made

clear that the variables have been drawn from a particular distribution and hence the distribution's parameters are known, as has been done above for explaining the i.i.d assumption. However, vaguely speaking, in machine learning problems, instead of random variables we have a dataset generated from an unknown distribution with unknown parameters, and the solution to such problems are this distribution's parameters. The two most popularly used methods to find these parameters are: Maximum Likelihood Estimation (MLE) and Maximum a Posteriori (MAP) given that all the data points are i.i.d. samples.

To fully understand the main idea behind MLE, the difference between probability and likelihood should be clear. Let X be a data of N i.i.d. samples: X_1, X_2, \dots, X_N . *Likelihood* is the joint probability of the data if the data belongs to a discrete distribution whereas it is the joint probability density of the data if the distribution is continuous. Since all the samples are i.i.d., the likelihood of our data given parameters θ is given by the product of the likelihood of each individual data point:

$$L(\theta) = \prod_{i=1}^N p(X_i|\theta) \quad (5.2)$$

which is a basically a generalized representation of [Equation 5.1](#). Now that finally the likelihood of the data has been framed as a function of θ , the MLE method can be defined as the process of the finding the parameters (θ) from a set of allowed values (Θ) which maximize the *likelihood function* $L(\theta)$ because the correct parameters $\hat{\theta}$ as given below will make the data much more probable as compared to incorrect parameters:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} L(\theta) = \arg \max_{\theta \in \Theta} \prod_{i=1}^N p(X_i|\theta) \quad (5.3)$$

To avoid the problem arithmetic underflow because of very small values of likelihood, the monotonically increasing property of the logarithm function is taken advantage of and instead of maximizing the likelihood, the log likelihood is maxi-

mized as both give the same value of θ :

$$\begin{aligned}\hat{\theta} &= \arg \max_{\theta \in \Theta} \log L(\theta) = \arg \max_{\theta \in \Theta} l(\theta) = \arg \max_{\theta \in \Theta} \log \prod_{i=1}^N p(X_i|\theta) \\ &= \arg \max_{\theta \in \Theta} \sum_{i=1}^N \log p(X_i|\theta)\end{aligned}\tag{5.4}$$

where, $l(\theta)$ is the *log likelihood function*. This is crux of almost all machine learning algorithms. The maximization this log likelihood function can be posed as an optimization problem which will discussed in the coming sections.

5.2 Gradient descent: An optimization technique

The maximum of the log likelihood can at times be found analytically:

$$\nabla_{\theta} l(\theta) = 0\tag{5.5}$$

However, in the majority of cases, there exist no closed-form solution to the above equation and in such cases the most frequently used method to find the correct parameters is *gradient ascent*. For a function f , the gradient $\nabla_{\theta} f(\theta)$ gives the direction in which the function increases most quickly and shows steepest ascent and for obvious reasons, $-\nabla_{\theta} f(\theta)$ gives the direction of steepest descent. This property of gradient has been used to formulate the iterative method of gradient ascent where on t^{th} iteration θ is updated as follows:

$$\theta^{(t)} = \theta^{(t-1)} + \alpha(t) \nabla_{\theta} l(\theta)\tag{5.6}$$

where, α is a small positive value which determines the step size of each iteration and is called the *learning rate*. The positive sign implies that we are maximizing our function. But in practice, rather than maximizing the log likelihood function, another function called the *cost function* which can be derived easily using

MLE in some cases, is minimized to determine the correct parameters and this method is appropriately named as *gradient descent*. The terms: cost function, loss function and error function are used alternatively in case of neural networks. The learning rule for gradient descent is:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha(t) \nabla_{\theta} J(\theta) \quad (5.7)$$

where, $J(\theta)$ is the cost function. As already mentioned, the fact that $-\nabla_{\theta} f(\theta)$ gives the direction of steepest descent of a function f makes this learning rule feasible. Both of these techniques: gradient ascent of log likelihood and gradient descent of cost function coincide with each other and can be used alternatively to provide the solution. The cost function depends on the type of machine learning problem. Till now, a generalized approach has been followed to understand how a machine learns from data but since a regression problem has been solved in this thesis, the cost function for regression and its minimization in the case of neural networks has been discussed in detail in the upcoming sections of this chapter. Before that, mentioning some advanced optimization algorithms seems much more appropriate as has been done in the next section.

5.3 Advanced optimization techniques

Training a neural network is a highly iterative procedure as it takes a large number of small iterations to attain a good prediction. In general to achieve good results huge training datasets are used which undoubtedly make the training process very slow. Hence, a continuous search for faster optimization algorithms exists. Although the standard gradient descent optimization works well for a lot of regression as well as classification problems and in most cases does reach a global or a local minima, however since the neural network model has to go through the entire training dataset before taking a small little step of gradient descent, it is a highly

time consuming optimization algorithm.

To overcome this limitation, a modified version of gradient descent called Stochastic Gradient Descent (SGD) has been proposed in which the weights of the neural network can be either optimized after each training sample or after a few training samples. Because of this, many updates are performed before processing the whole training dataset and hence the network trains a lot faster. Mathematically, the learning rule can be written as:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha(t) \nabla_{\theta} J_m(\theta) \quad (5.8)$$

where, J_m is the cost function computed for only m number of samples, instead of the entire dataset. If the weights are updated after each sample, the updates become very noisy, however, if the training dataset is divided into small batches of some fixed size, called the *batch size*, and the weights are optimized after the model processes all the samples in a single batch, then the noise issue gets eliminated and this particular method is referred to as *mini-batch gradient descent*. One major disadvantage of SGD with batch size = 1 is that because of regular updates after each training sample the computational time that could have been saved by doing vectorization of training samples is completely nullified.

Therefore, for a training dataset of size M , a batch size (m) of somewhere between $m = 1$, and $m = M$ is chosen, where the second case is of standard gradient descent. Also, it is extremely important to randomly shuffle the entire data before doing SGD to avoid poor convergence in case the data is ordered in a specific manner. SGD is almost always used as it provides a much faster convergence rate as compared to the standard GD by minimizing unnecessary computations caused due to possible redundancy in the dataset. Not only this, since huge datasets are used for training neural networks, computing the cost function as well as the gradient for the entire dataset may cause insufficient memory issues in a single machine which

SGD eliminates completely.

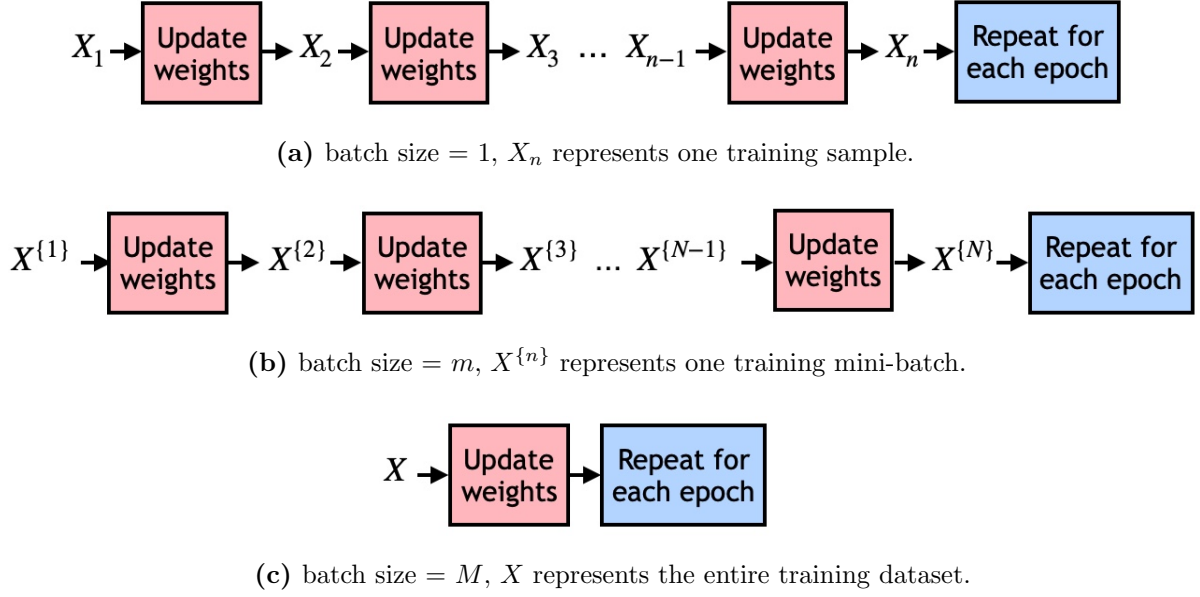


Figure 5.1: Block diagram representation of how weights are updated in case of Stochastic Gradient Descent (SGD) with different batch sizes.

Optimization algorithms with even better performance use the concept of exponentially moving average (EMA), also called exponentially weighted moving average (EWMA). In layman's terms, it is simply a method of smoothening noisy data. Scientifically speaking, bias-corrected EMWA $v^{(t)}$ at time t of a time series θ is given by:

$$v^{(t)} = \frac{\beta v^{(t-1)} + (1 - \beta) \theta^{(t)}}{(1 - \beta^t)} \quad (5.9)$$

where, $\theta^{(t)}$ is the value of our data at time t , $v^{(0)}$ is taken to be zero, and β is a coefficient whose value lies between 0 and 1. A value of β closer to 1 gives a greater smoothening effect. The denominator on the right hand side of the above equation is introduced for bias-correction purposes, that is to avoid low values of EWMA at the beginning of the time series.

To boost up the performance of SGD even further it can be coupled with a technique called *momentum* (Polyak, 1964). In SGD with momentum, first the

EWMA of the gradient of the cost function $\nabla_{\theta} J_m(\theta)$ is calculated using [Equation 5.9](#) and this average is then used to update the weights.

$$v^{(t)} = \beta v^{(t-1)} + (1 - \beta) \nabla_{\theta} J_m(\theta) \quad (5.10)$$

$$\theta^{(t)} = \theta^{(t-1)} - \alpha v^{(t)} \quad (5.11)$$

Sometimes to avoid a redundant constant, the above two equations are written as:

$$v^{(t)} = \gamma v^{(t-1)} + \alpha \nabla_{\theta} J_m(\theta) \quad (5.12)$$

$$\theta^{(t)} = \theta^{(t-1)} - v^{(t)} \quad (5.13)$$

Here, there is no need to do bias-correction because within a few iterations the EWMA gets to the desired value. This SGD enhancing technique called momentum is computationally inexpensive and leads to a quicker convergence by smoothening out the oscillations of the steps of SGD.

At this point, it is important to discuss two more optimization algorithms: Root Mean Square Propagation (RMS Prop) and Adaptive Moment Estimation (ADAM) proposed by [\(Tieleman and Hinton, 2012\)](#) and [\(Kingma and Ba, 2014\)](#) respectively. These two have gained much popularity over time, mainly because they have been found to be effective for a wide range of problems. For RMS Prop, EWMA of the square of $\nabla_{\theta} J_m(\theta)$ is calculated and the weights are updated accordingly:

$$s^{(t)} = \mu s^{(t-1)} + (1 - \mu) (\nabla_{\theta} J_m(\theta))^2 \quad (5.14)$$

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \frac{\nabla_{\theta} J_m(\theta)}{\sqrt{s^{(t)}} + \epsilon} \quad (5.15)$$

where, ϵ is an extremely small positive number which avoids the algorithm from running into numerical instability for values of $s^{(t)}$ very close to zero.

To combine the advantages of both RMS Prop and SGD with momentum,

ADAM optimizer was designed. For this algorithm we calculate EWMA of both $\nabla_{\theta} J_m(\theta)$ and its square. The first one is to account for SGD with momentum and the second one is from RMS Prop. Also, unlike the previous optimizers, in the original algorithm of ADAM, bias-corrected EWMA has been used. The learning rule for ADAM is given below:

$$v_{corr}^{(t)} = \frac{\beta v^{(t-1)} + (1 - \beta) \nabla_{\theta} J_m(\theta)}{1 - \beta^t} \quad (5.16)$$

$$s_{corr}^{(t)} = \frac{\mu s^{(t-1)} + (1 - \mu) (\nabla_{\theta} J_m(\theta))^2}{1 - \mu^t} \quad (5.17)$$

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \frac{v_{corr}^{(t)}}{\sqrt{s_{corr}^{(t)} + \epsilon}} \quad (5.18)$$

where, $v_{corr}^{(t)}$ and $s_{corr}^{(t)}$ are the corrected EWMA of $\nabla_{\theta} J_m(\theta)$ and its square respectively.

To summarize the advantages of all the advanced optimizers discussed, as compared to the vanilla SGD without momentum, a popularly used analogy is of a ball rolling down a hill, where the hill surface represents the cost function and therefore, the bottom of the hill represents the minimum value of the cost function, and the rolling of the ball downhill vaguely represents the model learning process. In SGD with momentum, in the beginning the ball starts rolling down in the direction of the gradient of the cost function, but once it gains some velocity it no longer follows the path of steepest descent as its momentum forces it to continue in the previous direction. This slows down learning in the directions in which gradient is oscillating and speeds it up in the directions where gradient remains relatively consistent. Whereas, in RMS Prop, the ball slows down in the directions along which the ball has already taken a significant number of steps, and speeds up in the directions along which the ball has taken only a few steps. Both of these qualities

significantly improve the optimization process, but inevitably, the learning efficiency gets enhanced even more when both are combined as in the case of the ADAM optimizer.

5.4 Perceptron: A single layer neural network

The building block of almost all existing neural networks is called the perceptron. The perceptron algorithm was presented by [Rosenblatt \(1962\)](#) and was later implemented in both hardware as well as software. By convention, while counting the number of layers in a neural network the input layer is not added up, hence, a perceptron is simply a single layer neural network which outputs a single real value when a vector of real numbers $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$ is given as an input ([Figure 5.2](#)).

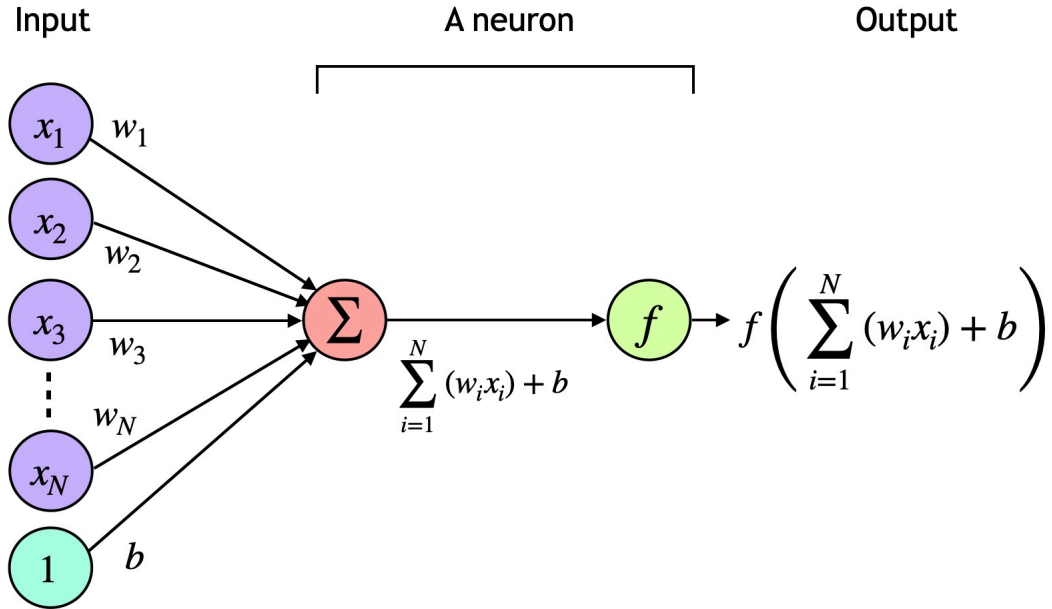


Figure 5.2: The building-block of neural networks, perceptron.

A function called the *activation function* (f) plays a key role in determining the output of the perceptron or neuron. These activation functions will be discussed

in detail in the next section. Mathematically speaking, the perceptron attaches different weights ($\mathbf{w} = \{w_1, w_2, \dots, w_N\}$) to each value of the input vector and determines a constant called the bias (b) to finally perform the following operation to get the output:

$$y = f\left(\sum_{i=1}^N (w_i x_i) + b\right) \quad (5.19)$$

In the original paper, this activation function was taken to be the step function such that the output of the perceptron could be either $+1$ or -1 :

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases} \quad (5.20)$$

A perceptron can learn to determine an appropriate weight vector and a constant bias in several ways, one of which is called the *perceptron rule*. Before that, consider the bias of the perceptron to be a weight itself corresponding to an additional entry of input vector equal to 1 as given in [Figure 5.2](#). Hence, the input vector now becomes $\mathbf{x} = \{x_1, x_2, \dots, x_N, 1\}$ and the weight vector becomes $\mathbf{w} = \{w_1, w_2, \dots, w_N, b\}$.

To understand this rule take an example of a supervised binary classification problem with one class represented by $+1$ and the other by -1 . Hence, the input will be a matrix of dimension: $M \times (N + 1)$ where M is the number of training samples and N is length of each training sample vector also called the *input feature vector*, whereas the output will be a matrix of dimension $M \times 1$ in which each entry can be either $+1$ or -1 . The weights are initialized randomly in the beginning. Each row of the input matrix, representing a single sample, is provided as input to the perceptron one by one and the output of the perceptron is compared with the actual output matrix entry. The weights are updated whenever the perceptron misclassifies a sample and this process is continued till the perceptron classifies majority of the

samples correctly. The update rule of the weights is given by:

$$w_i \leftarrow w_i + \alpha (y - \hat{y}) x_i \quad (5.21)$$

However, it did not take much time before people started noticing the limitations of the perceptron model. [Minsky and Papert \(1969\)](#) gave a very clever demonstration as to how a perceptron with a single neural unit is only suitable for linearly separable classes and can never solve the XOR problem. Later, it was demonstrated how efficiently the XOR problem can be solved by just adding one more layer of neural units to our vanilla perceptron model. For a neural network to learn more complex patterns and structures in a dataset, more layers with more number of neurons per layer are required. Thus, began the era of Multi-Layer Perceptron (MLP).

5.5 Multi-Layer Perceptron (MLP)

Multi-layer Perceptron (MLP) is a class of feed-forward neural networks which, undoubtedly, is the most extensively used one in today's date. In a feed-forward neural network, perceptrons are arranged in multiple layers, each layer consisting of several perceptrons. The first layer takes in the input and the last layer produces the output, whereas the layers in the middle are completely disconnected from the surroundings, and therefore are appropriately named as the *hidden layers*. The name *feed-forward* comes from the fact that in such a network each neuron of a particular layer is connected to every other neuron of the next layer but there exists no connections between neurons of the same layer. This allows the transformed input signal to be fed forward through all the layers.

MLPs can be customized a lot depending upon the problem at hand, not only can they have any number of hidden layers more than one, but also one has

a wide variety of non-linear activation functions to choose from in case of an MLP. Traditionally, before the Rectified Linear Unit (ReLU) came into the picture, sigmoid and tanh were the most popularly used activation functions in neural networks (Figure 5.3). The hyperbolic tangent function ranges from -1 to $+1$ while the sigmoid function also called the logistic function ranges from 0 to $+1$. However, both of these activation functions made the neural network suffer from the problem of vanishing gradients.

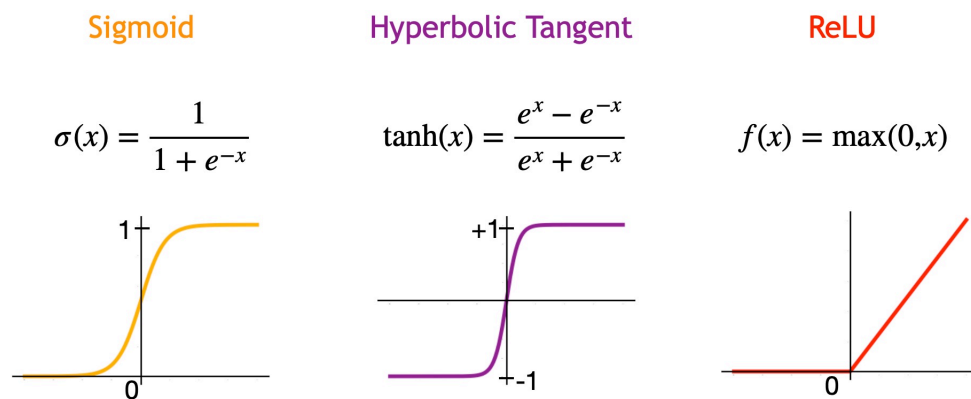


Figure 5.3: Different types of activation functions used in neural networks.

During backpropagation, the cost function gradient is propagated backward from the output layer to the input layer, as will be more clear when we talk about backpropagation, so that after updating its weights, the model improves and makes lesser error. If either tanh or sigmoid or both are used as activation functions in a network with many layers, this error gradient diminishes significantly by the time it reaches close to the input layer. This is because at very large or at very small values of x the derivative or the slope of these activation functions becomes very small as can be seen in Figure 5.3. This makes it rather difficult for the network to update its weights quickly and the gradient descent process of the entire network slows down.

Output of ReLU function is the input itself if the input is positive and zero if

the input is negative. It solves our problem of vanishing gradients pretty efficiently as its derivative is either 0 or 1. ReLU has become the default activation function for neural networks nowadays, in some cases hyperbolic tangent is still used but sigmoid on the other hand is hardly used. Few other activation functions are Leaky ReLU and linear activation functions.

The functioning of a feed-forward neural network can be studied in three segments namely: forward propagation, backward propagation and optimization. In this section, a three layer MLP for regression will be discussed in detail from this point onward, since the same has been used for system identification of a composite plate.

In [Figure 5.4](#), a three layer MLP is drawn where each bubble represents one unit of the network. Let n_l be the number of layers, L_l be the layer l . Also, for one training sample, let $W_{ij}^{(l)}$ be the weight for the connection between unit j in layer l and unit i in layer $l + 1$, $b_i^{(l)}$ be the bias associated with unit i in layer $l + 1$, x_i be the i^{th} entry of input feature vector of size N_0 , N_l be the number of units in layer l excluding the bias unit, $a_i^{(l)}$ be the output of unit i in layer l , $z_i^{(l)}$ be the weighted sum of the inputs to unit i in layer l including the bias, and $h(x)$ be the single valued output for regression. Also, for simplicity, let input layer be considered as the layer 0. After vectorization, all the above values for a single training batch (\mathbf{x}, \mathbf{y}) of size m , where each column represents one sample, can be compactly represented by: $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ as the weight matrix and the bias vector, connecting layer l with layer $l + 1$, respectively, $\mathbf{z}^{(l)}$ as the weighted sum matrix input to the layer l , \mathbf{x} as the input matrix, \mathbf{y} as the actual output vector, $\mathbf{h}(\mathbf{x})$ is the predicted output vector for a batch size of m samples, and $\mathbf{a}^{(l)}$ as the output of layer l .

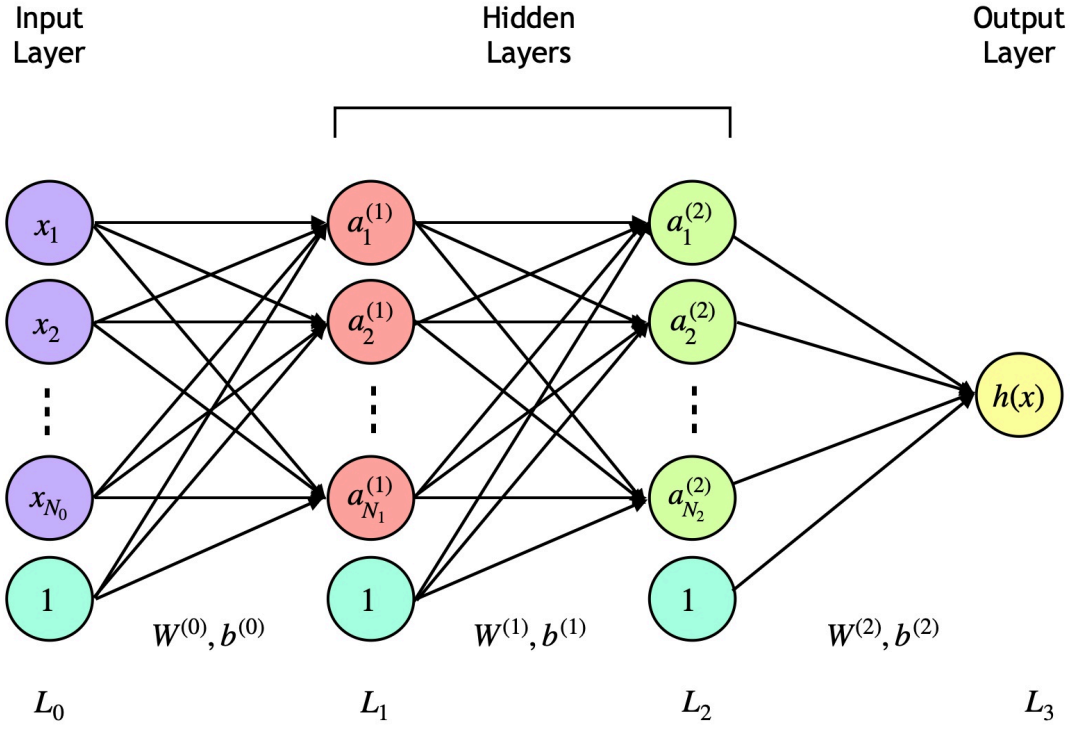


Figure 5.4: Three layer Multi-Layer Perceptron (MLP).

(1) **Forward propagation:** For a single training batch (\mathbf{x}, \mathbf{y}) , a forward pass through network is mathematically represented as:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(0)}\mathbf{a}^{(0)} + \mathbf{b}^{(0)} = \mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)} \quad (5.22)$$

$$\mathbf{a}^{(1)} = f(\mathbf{z}^{(1)}) \quad (5.23)$$

$$\mathbf{z}^{(2)} = \mathbf{W}^{(1)}\mathbf{a}^{(1)} + \mathbf{b}^{(1)} \quad (5.24)$$

$$\mathbf{a}^{(2)} = f(\mathbf{z}^{(2)}) \quad (5.25)$$

$$\mathbf{z}^{(3)} = \mathbf{W}^{(2)}\mathbf{a}^{(2)} + \mathbf{b}^{(2)} \quad (5.26)$$

$$\mathbf{h}(\mathbf{x}) = \mathbf{a}^{(3)} = \mathbf{z}^{(3)} \quad (5.27)$$

where, f is the activation function used in the hidden layers. In this thesis, the activation function for both the hidden layers is taken to be ReLU. Also, it is important to mention that since regression is being performed the output layer has no activation function.

(2) Backward propagation: The most widely used cost function used for regression problems is Mean Squared Error (MSE). In this network, for a training batch (\mathbf{x}, \mathbf{y}) , MSE cost function without any regularization term for some particular values of \mathbf{W} and \mathbf{b} is given by:

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{m}(\mathbf{h}(\mathbf{x}) - \mathbf{y})^T(\mathbf{h}(\mathbf{x}) - \mathbf{y}) \quad (5.28)$$

As already explained in [Section 5.2](#), the main objective here is to minimize this cost function which can be done using SGD, but before that the partial derivatives of the cost function with respect to all the network weights and biases should be calculated. For each unit i in each layer l except the input layer and for one sample only, define $\delta_i^{(l)}$ representing how much that particular unit contributes to any error in the output. In a compact manner, $\boldsymbol{\delta}^{(l)}$ will denote the error term for a whole layer l for a single training batch (\mathbf{x}, \mathbf{y}) . Using the chain rule we can derive the following:

$$\boldsymbol{\delta}^{(3)} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(3)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{h}(\mathbf{x})} = \frac{2}{m}(\mathbf{h}(\mathbf{x}) - \mathbf{y}) \quad (5.29)$$

$$\begin{aligned} \boldsymbol{\delta}^{(2)} &= \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(2)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(3)}} \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} = \boldsymbol{\delta}^{(3)} \frac{\partial}{\partial \mathbf{z}^{(2)}} (\mathbf{W}^{(2)} f(\mathbf{z}^{(2)}) + \mathbf{b}^{(2)}) \\ &= \left((\mathbf{W}^{(2)})^T \boldsymbol{\delta}^{(3)} \right) \cdot \frac{\partial f(\mathbf{z}^{(2)})}{\partial \mathbf{z}^{(2)}} \end{aligned} \quad (5.30)$$

$$\begin{aligned} \boldsymbol{\delta}^{(1)} &= \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(1)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} = \boldsymbol{\delta}^{(2)} \frac{\partial}{\partial \mathbf{z}^{(1)}} (\mathbf{W}^{(1)} f(\mathbf{z}^{(1)}) + \mathbf{b}^{(1)}) \\ &= \left((\mathbf{W}^{(1)})^T \boldsymbol{\delta}^{(2)} \right) \cdot \frac{\partial f(\mathbf{z}^{(1)})}{\partial \mathbf{z}^{(1)}} \end{aligned} \quad (5.31)$$

where, (\cdot) represents element-wise multiplication. Now, the partial derivatives of the cost function with respect to the weights and bias of each layer can be

given by:

$$\begin{aligned}\delta \mathbf{W}^{(2)} &= \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(2)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(3)}} \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} = \delta^{(3)} \frac{\partial}{\partial \mathbf{W}^{(2)}} (\mathbf{W}^{(2)} \mathbf{a}^{(2)} + \mathbf{b}^{(2)}) \\ &= \delta^{(3)} (\mathbf{a}^{(2)})^T\end{aligned}\tag{5.32}$$

$$\begin{aligned}\delta \mathbf{W}^{(1)} &= \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(1)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} = \delta^{(2)} \frac{\partial}{\partial \mathbf{W}^{(1)}} (\mathbf{W}^{(1)} \mathbf{a}^{(1)} + \mathbf{b}^{(1)}) \\ &= \delta^{(2)} (\mathbf{a}^{(1)})^T\end{aligned}\tag{5.33}$$

$$\begin{aligned}\delta \mathbf{W}^{(0)} &= \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(0)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}} = \delta^{(1)} \frac{\partial}{\partial \mathbf{W}^{(0)}} (\mathbf{W}^{(0)} \mathbf{x} + \mathbf{b}^{(0)}) \\ &= \delta^{(1)} (\mathbf{x})^T\end{aligned}\tag{5.34}$$

$$\delta \mathbf{b}^{(2)} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}^{(2)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(3)}} \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{b}^{(2)}} = \delta^{(3)} \frac{\partial}{\partial \mathbf{b}^{(2)}} (\mathbf{W}^{(2)} \mathbf{a}^{(2)} + \mathbf{b}^{(2)}) = \delta^{(3)}\tag{5.35}$$

$$\delta \mathbf{b}^{(1)} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}^{(1)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{b}^{(1)}} = \delta^{(2)} \frac{\partial}{\partial \mathbf{b}^{(1)}} (\mathbf{W}^{(1)} \mathbf{a}^{(1)} + \mathbf{b}^{(1)}) = \delta^{(2)}\tag{5.36}$$

$$\delta \mathbf{b}^{(0)} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}^{(0)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{b}^{(0)}} = \delta^{(1)} \frac{\partial}{\partial \mathbf{b}^{(0)}} (\mathbf{W}^{(0)} \mathbf{x} + \mathbf{b}^{(0)}) = \delta^{(1)}\tag{5.37}$$

(3) Optimization: If all of the parameters are initialized as zeros, every hidden layer neuron of the network will learn the same function of the input, hence, before training is commenced, weights and biases are all initialized randomly. This breaks the so-called *symmetry*. The weights and bias of any layer l are updated as follows:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \alpha \delta \mathbf{W}^{(l)}\tag{5.38}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \delta \mathbf{b}^{(l)}\tag{5.39}$$

These three steps are repeated m times for the entire training set to complete one *epoch*. Although the training process discussed here uses standard SGD without momentum, any one of the advanced optimization algorithms can be used as given

in [Section 5.3](#).

5.6 Radial Basis Function Network (RBFN)

Originally, Radial Basis Functions (RBFs) provided a simple yet effective approach to exact function interpolation for functions depending on two or more variables, called *multivariate functions* ([Micchelli, 1986](#); [Powell, 1987](#)). An RBF is basically a function of the Euclidean distance between the input \mathbf{x} and a fixed point \mathbf{c}_j called the *center* of the RBF.

$$\phi_j(\mathbf{x}) = h(\|\mathbf{x} - \mathbf{c}_j\|) \quad (5.40)$$

For a set of input vectors $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and the corresponding set of target scalars $\{y_1, y_2, \dots, y_N\}$, a function f which can fit all the data points, such that the equation $y_n = f(\mathbf{x}_n)$ is satisfied for every (\mathbf{x}_n, y_n) pair, can be expressed as a linear combination of RBFs with each RBF having a different center:

$$f(\mathbf{x}) = \sum_{n=1}^N W_n \phi(\|\mathbf{x} - \mathbf{x}_n\|) \quad (5.41)$$

where W_n is the weight associated with a single RBF with center \mathbf{x}_n . These weights can be very easily determined using the method of least squares which makes the model exactly fit the given data but simultaneously leads to the major issue of overfitting, where the models fails to generalize over unseen data ([Bishop, 2006](#)).

There are numerous types of RBFs and a few of them and their plots ([Figure 5.5](#)) are given below for one-dimensional input space:

$$\text{Gaussian:} \quad \phi(\mathbf{x}) = e^{-\frac{\mathbf{x}^2}{r^2}} \quad (5.42)$$

$$\text{Multi-quadric:} \quad \phi(\mathbf{x}) = \sqrt{\frac{\mathbf{x}^2}{r^2} + 1} \quad (5.43)$$

$$\text{Inverse multi-quadric:} \quad \phi(\mathbf{x}) = \frac{1}{\sqrt{\frac{\mathbf{x}^2}{r^2} + 1}} \quad (5.44)$$

$$\text{Cauchy:} \quad \phi(\mathbf{x}) = \frac{1}{\frac{\mathbf{x}^2}{r^2} + 1} \quad (5.45)$$

where, r is a scalar called the spread of the RBF which determines how fast the function decays to zero as one moves away from the centre of the RBF.

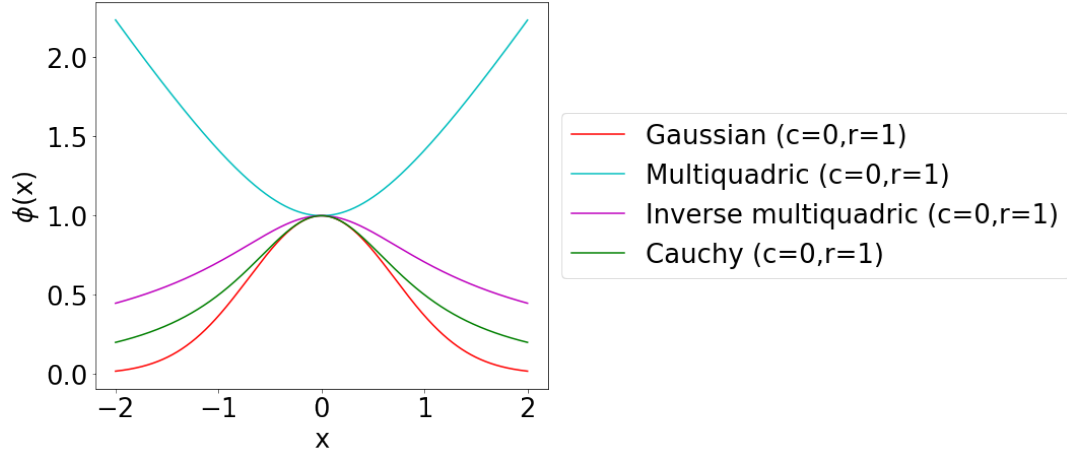


Figure 5.5: Different types of Radial Basis Functions for one-dimensional input space.

[Broomhead and Lowe \(1988\)](#) were the first to develop a feed-forward non-linear Radial Basis Function Network (RBFN) with explicit fitting and optimization procedure. They also explained how the neural network training is synonymous to solving a set of linear equations and similarly how generalization is analogous to data interpolation between the training data points. Since RBFs were already established functions used for solving interpolation problems in high dimensional spaces, as explained above, considering them useful as activation functions for neural networks seemed to be a reasonable choice for modelling complex non-linear relationship between inputs and outputs.

Originally, RBFNs are two layer neural networks, consisting of only a single hidden layer. The activation functions of the hidden layer neurons are individually

centered RBFs. Among others, the most commonly used RBF activation function for RBFNs is the Gaussian RBF (Equation 5.42). A major contrasting characteristic of an RBFN as compared to an MLP is that in RBFN the input and the hidden layer are weightlessly connected, and only the connections between the hidden layer and the output layer are weighted.

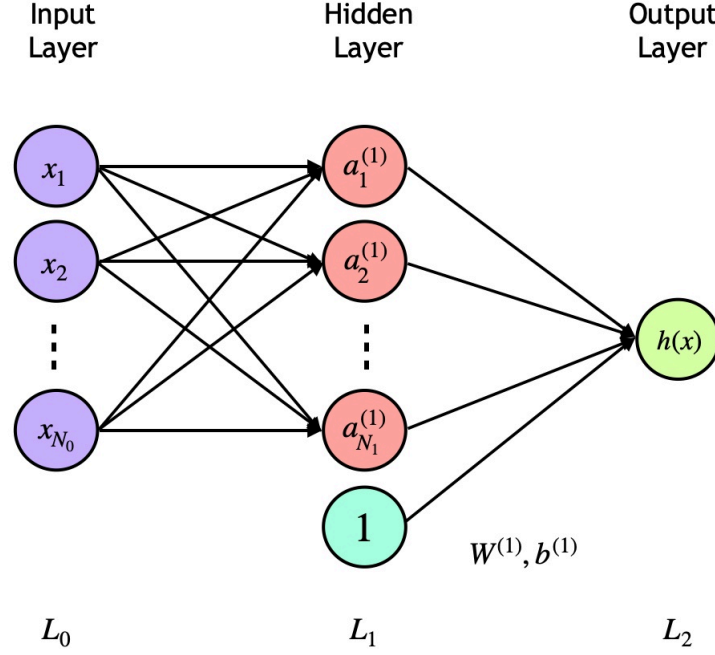


Figure 5.6: Radial Basis Function Network (RBFN).

In Figure 5.6, the activation function of each unit j of the hidden layer is taken to be a Gaussian RBF having spread r_j , and center vector \mathbf{c}_j which has the same length as that of a single input training sample \mathbf{x}_i , which is N_0 . Just as in the case of MLP, here too $\mathbf{a}^{(l)}$ and $\mathbf{z}^{(l)}$ give the output of, and the input to layer l , respectively. $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and the bias vector, connecting layer l with layer $l + 1$, and there exists only one set of these $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ in case of RBFN. Let \mathbf{c} and \mathbf{r} be the vectorized centroids and spreads of the whole hidden layer. Also, \mathbf{x} is the input matrix, \mathbf{y} is the actual output vector, and $\mathbf{h}(\mathbf{x})$ is the predicted output vector for a batch size of m samples.

So basically, unlike MLP, in RBFN four parameters: \mathbf{W} and \mathbf{b} , \mathbf{c} and \mathbf{r} , need

to be calculated or optimized. To determine the centroids, clustering algorithms, such as k-means clustering are used. The number of neurons in the hidden layer define the number of clusters that the training dataset needs to be divided into, which is actually a design parameter. The centers of all these clusters form the centroids of the hidden layer RBF neurons.

Now to determine the RBF spreads of the hidden layer neurons, k-nearest neighbour heuristic is used and for each RBF neuron centroid \mathbf{c}_j , k nearest RBF neuron centroids are determined. Finally to calculate the spread r_j we use:

$$r_j = \sqrt{\frac{1}{k} \sum_{i=1}^K (\mathbf{c}_j - \mathbf{c}_i)^T (\mathbf{c}_j - \mathbf{c}_i)} \quad (5.46)$$

where, \mathbf{c}_i is the centroid of one of the k nearest neighbors of the centroid \mathbf{c}_j . Since matrix \mathbf{c} and vector \mathbf{r} can now be assembled using vectorization, the only thing left is to determine the hidden layer weights and bias which has been already discussed in detail for an MLP, so keeping it brief, the three steps are:

Forward propagation: A complete pass through the whole network for a training batch (\mathbf{x}, \mathbf{y}) looks like:

$$\mathbf{z}^{(1)} = \mathbf{x} \quad (5.47)$$

$$\mathbf{a}^{(1)} = f(\mathbf{z}^{(1)}) \quad (5.48)$$

$$\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{a}^{(1)} + \mathbf{b}^{(1)} \quad (5.49)$$

$$\mathbf{h}(\mathbf{x}) = \mathbf{a}^{(2)} = \mathbf{z}^{(2)} \quad (5.50)$$

where, f is the Gaussian RBF activation functions of the hidden layer. Here again it is important to mention that there is only one unit in the output layer because we are performing regression in this thesis.

Backward propagation: The cost function used here too is MSE given in

[Equation 5.28](#). For calculating the derivative of the cost function with respect to \mathbf{W} and \mathbf{b} , let $\delta^{(l)}$ be the error term for layer l for a single training batch (\mathbf{x}, \mathbf{y}) , and now using the chain rule:

$$\delta^{(2)} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(2)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{h}(\mathbf{x})} = \frac{2}{m}(\mathbf{h}(\mathbf{x}) - \mathbf{y}) \quad (5.51)$$

where, (\cdot) represents element-wise multiplication. Now, the partial derivatives of the cost function with respect to the weights and bias of the hidden layer can be given by:

$$\begin{aligned} \delta \mathbf{W}^{(1)} &= \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(1)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} = \delta^{(2)} \frac{\partial}{\partial \mathbf{W}^{(1)}} (\mathbf{W}^{(1)} \mathbf{a}^{(1)} + \mathbf{b}^{(1)}) \\ &= \delta^{(2)} (\mathbf{a}^{(1)})^T \end{aligned} \quad (5.52)$$

$$\delta \mathbf{b}^{(1)} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}^{(1)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{b}^{(1)}} = \delta^{(2)} \frac{\partial}{\partial \mathbf{b}^{(1)}} (\mathbf{W}^{(1)} \mathbf{a}^{(1)} + \mathbf{b}^{(1)}) = \delta^{(2)} \quad (5.53)$$

(3) Optimization: After randomly initializing weights and bias, the model can be updated as:

$$\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} - \alpha \delta \mathbf{W}^{(1)} \quad (5.54)$$

$$\mathbf{b}^{(1)} \leftarrow \mathbf{b}^{(1)} - \alpha \delta \mathbf{b}^{(1)} \quad (5.55)$$

Repeating these steps m times will complete one epoch. For this thesis, ADAM optimizer has been used instead of standard SGD.

Chapter 6

Results and Discussion

Before jumping to the results and the subsequent discussion, it is important to describe the proposed setup in depth. An orthotropic composite plate has four independent material properties: E_1 and E_2 (Young's moduli in two directions), ν_{12} (Poisson's ratio), and G_{12} (shear modulus). To accomplish the final objective of this thesis, which is to use machine learning techniques for system identification of a composite plate, the very first step is to successfully generate the frequency as well as the acceleration response of the composite plate for a set of these four parameter values.

The algorithms for Finite Element Analysis of the composite plate (Algorithm 1) as well as for the Newmark integration method (Algorithm 2) were implemented in python using the numpy library. For system identification, a carbon fiber reinforced epoxy in-plane bending orthotropic plate was considered. The plate had a length (L_x) of 0.5 m , width (L_y) of 0.5 m , thickness (h) of 0.005 m and density (ρ) of 1846 kgm^{-3} .

Recapitulating the thorough discussion done on the modelling process in Chapter 3, the composite plate was divided into 16 equal elements as shown in

Figure 3.1 and the boundary nodes [1, 2, 3, 4, 5, 6, 10, 11, 15, 16, 20, 21, 22, 23, 24, 25] were kept fixed. A single node had 3 degrees-of-freedom (DOFs) and there were 4 nodes in each element. Altogether, the plate consisted of 16 elements, 25 nodes, 75 total DOFs, 16 fixed nodes, 48 fixed DOFs, and 27 variable DOFs. A sinusoidal force of $20 \sin(2\pi 50t)$ Newton was applied along the z-direction on the 13th node of the plate for 6 s and the acceleration values of the 14th node were recorded for these 6 s for a time step size of 0.001 s. The plot in Figure 6.1 shows the acceleration response of the 14th node for two different sets of parameters for the first 0.6 s.

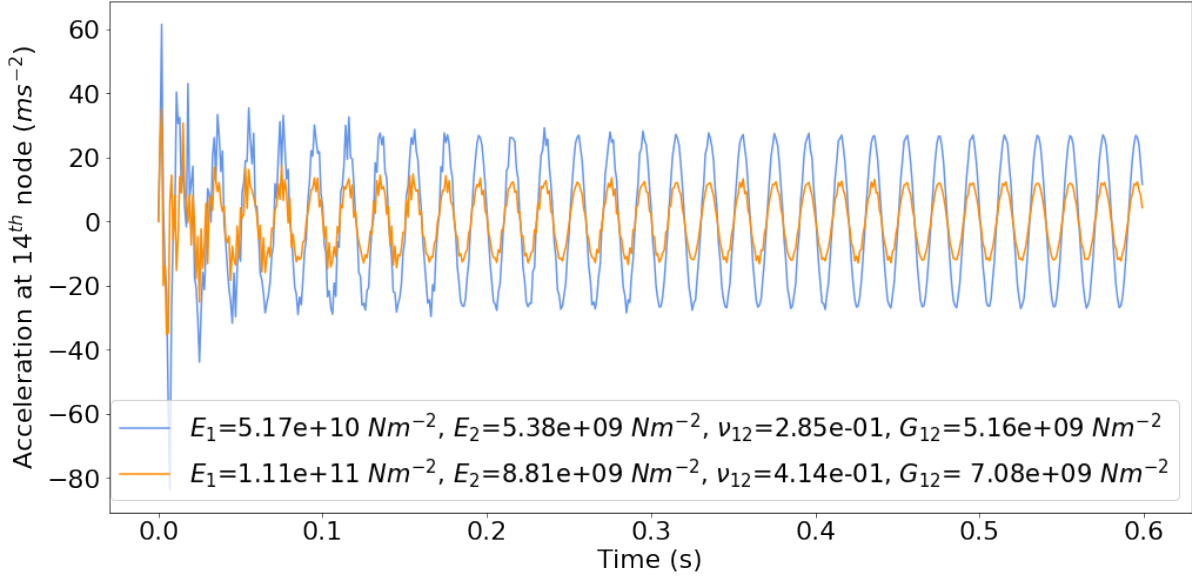


Figure 6.1: Acceleration response of the composite plate for two sets of parameter values for the first 0.6 s.

The first five natural frequencies for the parameter values [$5.17 \times 10^{10} \text{ Nm}^{-2}$, $5.38 \times 10^9 \text{ Nm}^{-2}$, 2.85×10^{-1} , $5.16 \times 10^9 \text{ Nm}^{-2}$] were observed to be: [2843.04, 2631.54, 2520.79, 2006.57, 1880.22].

To choose an appropriate technique to sample a dataset, for any machine learning model to learn upon, is as important as the model itself. For this purpose, a stratified sampling technique called Latin Hypercube Sampling (LHS) has been employed because of its obvious advantages over Simple Random Sampling as given

in [Chapter 4](#). Again, python's numpy library was used to develop a code for LHS (Algorithm 3).

Two datasets of sample sizes 2K and 200K were simulated with minimum parameter values $[50 \times 10^9 Nm^{-2}, 5 \times 10^9 Nm^{-2}, 0.25, 2 \times 10^9 Nm^{-2}]$ and maximum parameter values $[150 \times 10^9 Nm^{-2}, 10 \times 10^9 Nm^{-2}, 0.45, 8 \times 10^9 Nm^{-2}]$, where each entry represents one of the following parameters: E_1 , E_2 , ν_{12} , and G_{12} respectively. Both of these datasets individually contain three matrices: first one is an $s \times 4$ matrix P for the parameter values, second one is an $s \times 27$ matrix F for the frequency response and the last one is an $s \times 6001$ matrix A for the acceleration response, s being the sample size.

Finally, two Artificial Neural Network classes were designed: a three layer MLP ([Figure 5.4](#)) and a two layer RBFN ([Figure 5.6](#)), both mathematically described in detail in [Chapter 5](#), using the highly equipped pytorch library of python. The four parameters were modelled using four different networks to provide more flexibility for hyperparameter tuning while training. Each of the four columns of the matrix P described above serve as the four output vectors for the four said networks, matrix F is used as input vector for frequency domain system identification and matrix A is the input matrix when identification is done in time domain.

All the input and output matrices were normalized using the min-max normalization method given in the python's sklearn library. Each output matrix column was normalized to the range $[0, 1]$ ([Equation 6.1](#)) while each column of both the input matrices was scaled to the range $[-1, +1]$ ([Equation 6.2](#)):

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)} \quad (6.1)$$

$$X' = 2 \left[\frac{X - \min(X)}{\max(X) - \min(X)} \right] - 1 \quad (6.2)$$

where, $\min(X)$ and $\max(X)$ are the minimum-valued and the maximum-valued entries of a single column, X is the vector of original entries of a column and X' is the normalized column.

Talking about the network characteristics, for MLP the first hidden layer was assigned 20 neurons while the second one had 10 neurons. And for RBFN, 300 neurons were assigned to its single hidden layer. The cost function used was MSE loss and the optimization algorithm used for updating the network weights and biases was ADAM optimizer as discussed in [Section 5.3](#).

Now that almost all the intricacies have been dealt with thoroughly, the results are as presented in [Table 6.1](#), [Figure 6.2](#), [Table 6.2](#), [Figure 6.3](#). The codes were run on an Intel^(R) Xeon^(R) 1 Core 2.00GHz CPU. First of all, it should be clearly mentioned that although the identification process was conducted in both time and frequency domain, the results of only frequency domain identification have been reported, mainly because of two reasons: first, a much higher average error was recorded when the acceleration response matrix A was used as the input dataset as opposed to the frequency response matrix F , and second, the computational time required to train the model was very high for the time domain dataset because of the large input feature vector size, 6001 to be precise.

It can be concluded from the error values of the four parameters that the predictions of both the networks were more accurate for the two values of Young's moduli as compared to Poisson's ratio and shear modulus. Comparing the results of the two networks, one can easily gather that an MLP performs way better as opposed to an RBFN. Not only does the MLP demonstrate lower error values but also the time required for its training is only a little over half the RBFN training time. All the parameters show significantly decreased average error values on increasing the sample size of the dataset as can be seen in the [Table 6.3](#).

Table 6.1: Table showing the performance the proposed three layer MLP with 200K sample size.

Parameter	Training Time (s)	Test Case Actual Value	Test Case Predicted Value	Test Case Percentage Error (%)
E_1 (Nm^{-2})	424	1.21e+11	1.21e+11	0
E_2 (Nm^{-2})	376	5.73e+09	5.61e+09	2.1
ν_{12}	402	0.297	0.309	4.0
G_{12} (Nm^{-2})	390	6.14e+09	6.44e+09	4.8

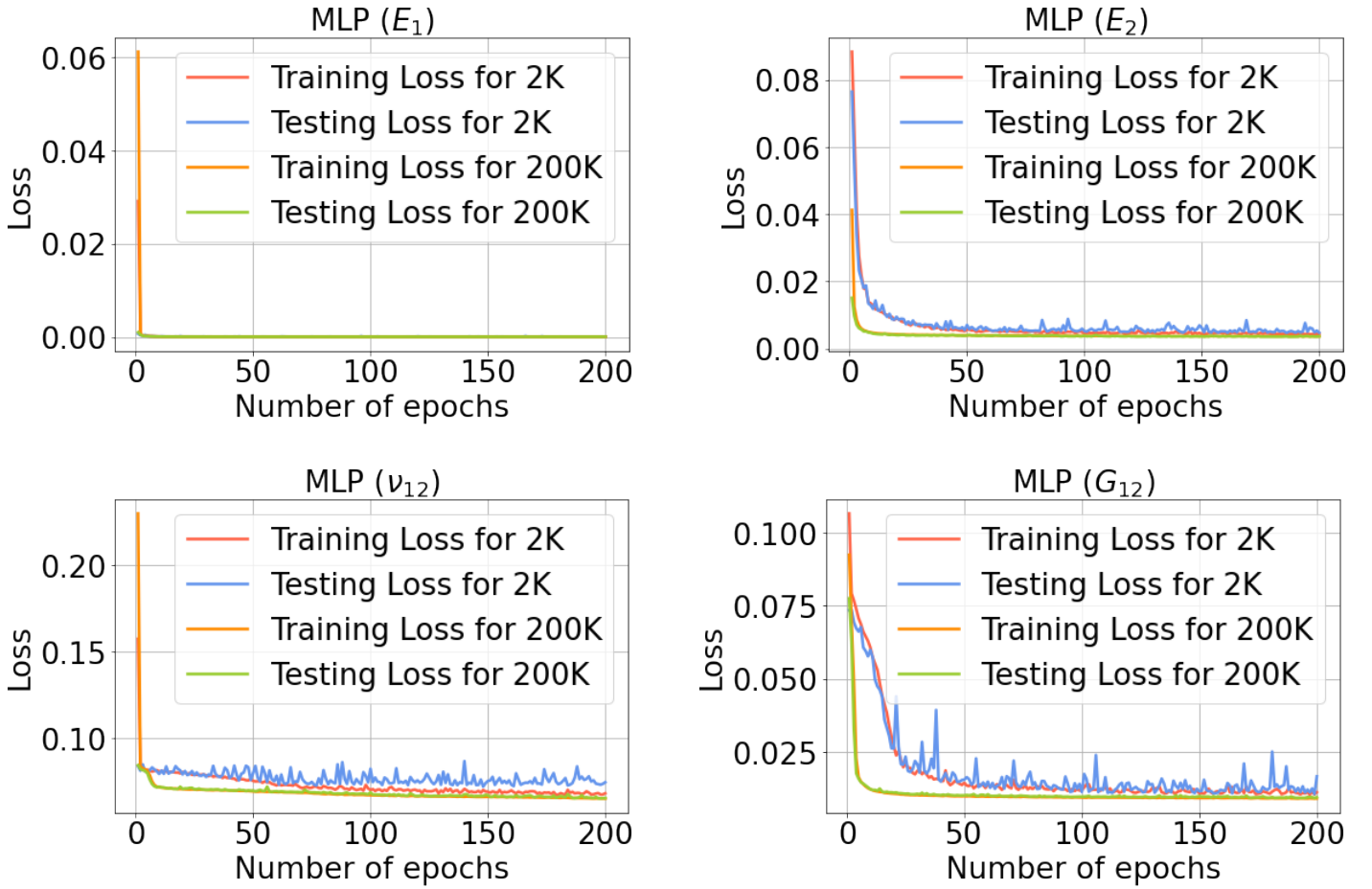


Figure 6.2: Learning curves of the proposed three layer MLP for the four parameters E_1 , E_2 , ν_{12} and G_{12} for two sample sizes 2K and 200K.

Table 6.2: Table showing the performance the proposed RBFN with 200K sample size.

Parameter	Training Time (s)	Test Case Actual Value	Test Case Predicted Value	Test Case Percentage Error (%)
E_1 (Nm^{-2})	712	1.21e+11	1.23e+11	1.7
E_2 (Nm^{-2})	717	5.73e+09	5.95e+09	3.8
ν_{12}	698	0.297	0.336	13.1
G_{12} (Nm^{-2})	711	6.14e+09	4.96e+09	19.4

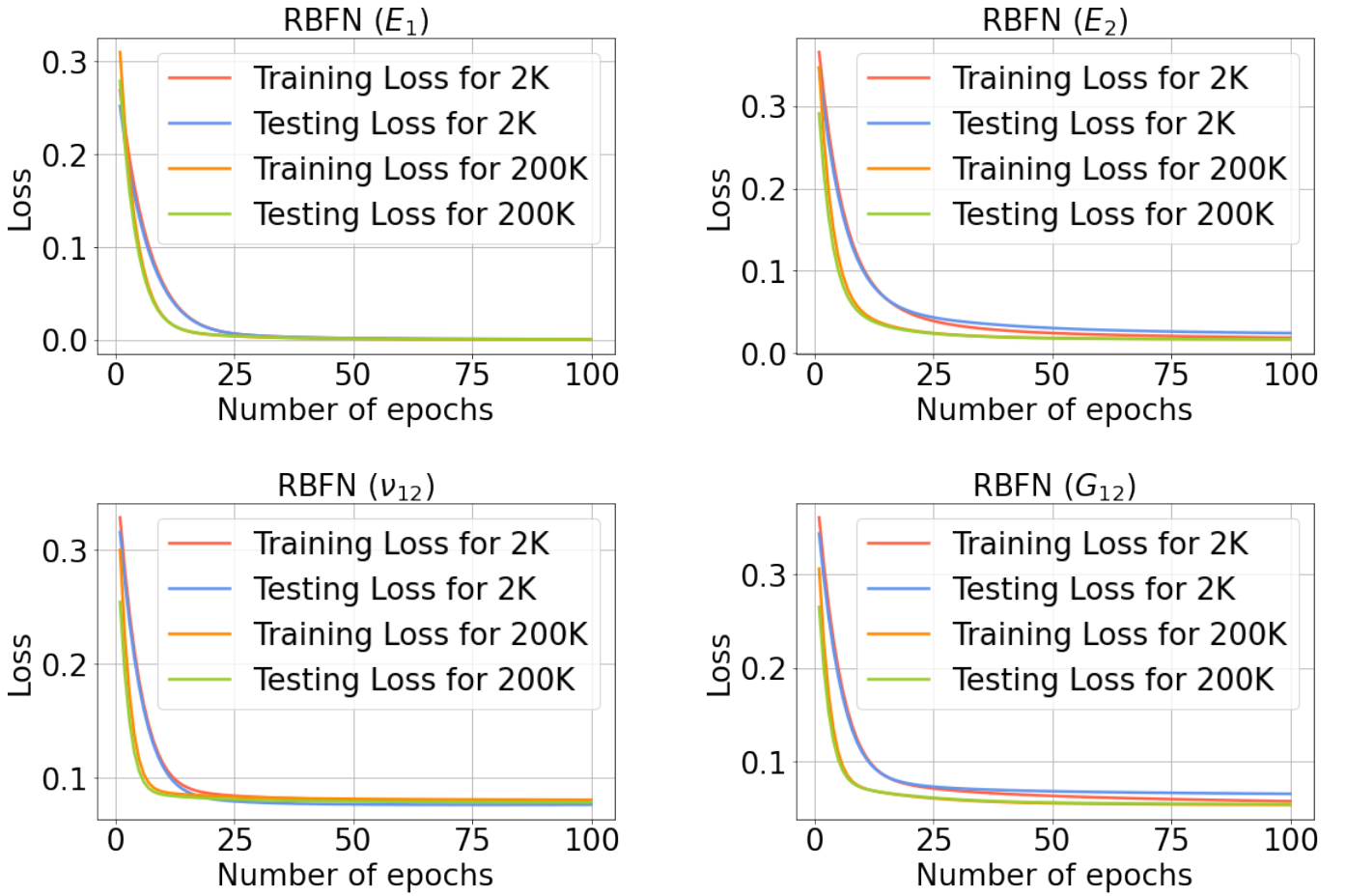


Figure 6.3: Learning curves of the proposed RBFN for the four parameters E_1 , E_2 , ν_{12} and G_{12} for two sample sizes 2K and 200K.

Table 6.3: Table showing the average percentage error calculated over a larger test dataset for different networks with different sample sizes.

Parameter	MLP (2K)	MLP (200K)	RBFN (2K)	RBFN (200K)
E_1	0.65%	0.44%	1.51%	1.40%
E_2	3.58%	2.98%	8.26%	7.99%
ν_{12}	14.32%	11.25%	14.19%	12.32%
G_{12}	11.31%	10.10%	29.11%	25.32%

Chapter 7

Summary and Conclusions

The study of the relationship between the observed input and output signals of an unknown system in order to develop its mathematical model is called system identification. In this thesis, the unknown system is represented by a composite plate on which parametric identification has been conducted to determine its mechanical properties: Young's moduli in two directions, Poisson's ratio, and shear modulus.

A set of 27 natural frequencies of the composite plate were successfully computed using FEA by dividing the plate into 16 small elements. On the 13th node of the discretized plate a sinusoidal force was applied in the z-direction for 6 s and the acceleration of the 14th node was recorded using Newmark integration. The frequency and time domain input-output data was meticulously sampled using LHS, a stratified sampling technique. An RBFN model with 300 hidden layer neurons, and a three layer MLP model with 20 neurons in the first hidden layer and 10 in the second one, were successfully designed for regression. The prediction of the four parameters in both the domains was done in a systematic manner. Finally, the entire investigation presented in this thesis can be concluded as follows:

1. The identification performed using frequency response showed a much superior

accuracy and efficiency as compared to time domain identification.

2. With increase in the training dataset size, from 2K to 200K, a significant decrease in the average percentage error was observed.
3. The MLP model trained with 200K sample size predicted the four parameters for the test case with an average error of 2.7% and an average training time of less than 7 minutes. The individual percentage errors were 0% for E_1 , 2.1% for E_2 , 4.0% for ν_{12} , 4.8% for G_{12} .
4. The RBFN model trained with 200K sample size predicted the four parameters for the test case with an average error of 9.5% and an average training time of less than 12 minutes. The individual percentage errors were 1.7% for E_1 , 3.8% for E_2 , 13.1% for ν_{12} , 19.4% for G_{12} .

Owing to a greater and more efficient performance, the MLP network was acknowledged as the better proposed model as opposed to the RBFN.

Scope for future work: Since Convolutional Neural Networks (CNNs) have a history of outperforming MLPs in case of sequenced time series data, it can serve as a great alternative to the proposed models for time domain identification. Another statistical technique, especially popular for time series analysis, called Autoregressive Integrated Moving Average (ARIMA) which is quite frequently used in economics, can be tested for both frequency domain and time domain identification. Other than these two, another option available is of ensemble learning which has wide industrial applications in today's date for regression analysis, and may also perform well for the proposed parametric identification.

Bibliography

- R.M. Jones. *Mechanics of Composite Materials (Second Edition)*. Taylor & Francis, 1999.
- H. Imai, C.-B. Yun, O. Maruyama, and M. Shinozuka. Fundamentals of system identification in structural dynamics. *Probabilistic Engineering Mechanics*, 4(4): 162–173, 1989.
- C.G. Koh, Y.F. Chen, and C.-Y. Liaw. A hybrid computational strategy for identification of structural parameters. *Computers Structures*, 81(2):107–117, 2003.
- I.H. Witten, E. Frank, and M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2011.
- M. Crawford, T. Khoshgoftaar, J.D. Prusa, A.N. Richter, and H.A. Najada. Survey of review spam detection using machine learning techniques. *Journal of Big Data*, 2:1–24, 2015.
- J. Patel, S. Shah, P. Thakkar, and K. Kotecha. Predicting stock market index using fusion of machine learning techniques. *Expert Systems with Applications*, 42(4): 2162–2172, 2015.
- A. Lavecchia. Machine-learning approaches in drug discovery: methods and applications. *Drug Discovery Today*, 20(3):318–331, 2015.

- P.A. Sankar, R. Machavaram, and K. Shankar. System identification of a composite plate using hybrid response surface methodology and particle swarm optimization in time domain. *Measurement*, 55:499–511, 2014.
- S. Ye, B. Li, Q. Li, H.-P. Zhao, and X.-Q. Feng. Deep neural network method for predicting the mechanical properties of composites. *Applied Physics Letters*, 115(16):161901, 2019.
- D.W. Abueidda, M. Almasri, R. Ammourah, U. Ravaioli, I.M. Jasiuk, and N.A. Sobh. Prediction and optimization of mechanical properties of composites using convolutional neural networks. *Composite Structures*, 227:111264, 2019.
- G. Pillonetto, F. Dinuzzo, T. Chen, G.D. Nicolao, and L. Ljung. Kernel methods in system identification, machine learning and function estimation: A survey. *Automatica*, 50(3):657–682, 2014.
- K.J. Åström and B. Wittenmark. *Adaptive Control (Second Edition)*. Addison-Wesley Publishing Company, Reading, MA, 1995.
- L. Ljung. *System Identification: Theory for the User*. P T R Prentice-Hall, Englewood Cliffs, New Jersey, 1987.
- D. McHenry. A lattice analogy for the solution of stress problems. *Journal of the institution of civil engineers*, 21(2):59–82, 1943.
- A. Hrennikoff. Solution of problems of elasticity by the framework method. *Journal of Applied Mechanics*, 8(4):A169–A175, 1941.
- N.M. Newmark. Numerical methods of analysis of bars, plates, and elastic bodies. In *L.E. Grinter, editor, Numerical Methods in Analysis in Engineering*. Macmillan, New York, 1949.
- R.V. Southwell. *Relaxation Methods in Theoretical Physics*. Oxford University Press, London, 1946.

- R.W. Clough. The finite element method in plane stress analysis. In *Proceedings, Second Conference on Electronic Computation, ASCE, Pittsburgh, PA*, 1960.
- N.M. Newmark. A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division*, 85(3):67–94, 1959.
- K.J. Bathe. *Finite Element Procedures (First Edition)*. Prentice Hall, Pearson Education, Inc., 2006.
- M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.
- D. Raj. *Sampling Theory*. McGraw-Hill, 1968.
- W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
- F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington DC, 1962.
- T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
- C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006.
- B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.
- T. Tieleman and G. Hinton. Lecture 6.5 (rmsprop): Divide the gradient by a running average of its recent magnitude. *Coursera: Neural Networks for Machine Learning*, 2012.
- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

- M. Minsky and S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.
- C.A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.
- M.J.D. Powell. Radial basis functions for multivariable interpolation: a review. *Algorithms for approximation*, page 143–167, 1987.
- D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.