A COMPARATIVE STUDY OF SURROGATE MUSCULOSKELETAL MODELS USING VARIOUS NEURAL NETWORK CONFIGURATIONS

A THESIS IN

Electrical Engineering

Presented to the Faculty of the University of
Missouri Kansas City in partial fulfillment
of the requirements for the degree

MASTERS OF SCIENCE

by

PALGUN REDDY PULASANI

Kansas City, Missouri

2013

A COMPARATIVE STUDY OF SURROGATE MUSCULOSKELETAL MODELS USING VARIOUS NEURAL NETWORK CONFIGURATIONS

Palgun Reddy Pulasani, Candidate for the Master of Science Degree

University of Missouri-Kansas City, 2013

ABSTRACT

The central idea in musculoskeletal modeling is to be able to predict body-level (e.g. muscle forces) as well as tissue-level information (tissue-level stress, strain, etc.). To develop computationally efficient techniques to analyze such models, surrogate models have been introduced which concurrently predict both body-level and tissue-level information using multi-body and finite-element analysis, respectively. However, this kind of surrogate model is not an optimum solution as it involves the usage of finite element models which are computation intensive and involve complex meshing methods especially during real-time movement simulations. An alternative surrogate modeling method is the use of artificial neural networks in place of finite-element models.

The ultimate objective of this research is to predict tissue-level stresses experienced by the cartilage and ligaments during movement and achieve concurrent simulation of muscle force and tissue stress using various surrogate neural network models, where stresses obtained from finite-element models provide the frame of reference. Over the last decade, neural networks have been successfully implemented in several biomechanical modeling applications. Their adaptive ability to learn from examples, simple implementation techniques, and fast simulation times make neural

networks versatile and robust when compared to other techniques. The neural network models are trained with reaction forces from multi-body models and stresses from finite element models obtained at the interested elements. Several configurations of static and dynamic neural networks are modeled, and accuracies close to 93% were achieved, where the correlation coefficient is the chosen measure of goodness. Using neural networks, the simulation time was reduced nearly 40,000 times when compared to the finite-element models. This study also confirms theoretical concepts that special network configurations--including average committee, stacked generalization, and negative correlation learning--provide considerably better results when compared to individual networks themselves.

APPROVAL

The faculty listed below, appointed by the Dean of the School of Computing and engineering, have examined a thesis titled "A Comparative Study of Surrogate Musculoskeletal Models Using Various Neural Network Configurations" by Palgun Reddy Pulasani, candidate for the Master of Science degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Reza R. Derakhshani, Ph.D., (Committee Chair)
Department of Electrical and Computer Engineering

Trent M. Guess, Ph.D.
Department of Civil and Mechanical Engineering

Ghulam M. Chaudhry, Ph.D.
Department of Electrical and Computer Engineering

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

ACKNOWLEDGEMENTS

I heartily thank my advisor, Dr. Reza Derakhshani, for his continued assistance and immense trust in me throughout my Masters career, and for giving me a chance to prove my expertise. Besides my advisor, I sincerely thank my committee members, Dr. Trent Guess and Dr. Ghulam Chaudhry, for their encouragement and insightful comments. If it were not for their timely support, this project wouldn't have been possible.

I wish to express my special gratitude towards Dr. Trent Guess and Dr. Yunkai Lu for assisting me in the bio-mechanical portion of the project whenever required and for providing the multi-body and finite element analysis datasets which were the frame of reference to the neural networks.

I am deeply indebted to my parents Mr. Anjani Kumar Reddy and Mrs. Pranitha Reddy for always standing by me with extreme love and affection. Thanks Mom for all your everlasting efforts and commitment towards my studies; and thanks Dad for your rightful advices and unconditional support at all times though out my life.

I also thank my lab mates for guiding me with rightful advice during my Masters whenever needed. In particular, I am thankful to Mr. Sri Ram Pavan Tankasala for advising me to enter into the world of neural networks in the first place.

Last but not the least, I would like to thank my friends Chetan, Abhinav, Teja, Swaroop and Aadit for ensuring that my entire 2 years of stay here in Kansas City was filled with joyful and memorable moments.

Dedicated to my Mom and Dad

THESIS OUTLINE

This thesis describes the application of various configurations of neural networks for analyzing the performance of musculoskeletal models. Chapter 1 gives an introductory layout of the biomechanics of surrogate musculoskeletal modeling by familiarizing multi-body and finite element modeling techniques. It also mentions several linear and non-linear modeling techniques with special focus on neural networks.

Chapter 2 gives more specifics on the actual modeling techniques implemented. It starts by giving details on how the multi-body and finite element analysis was implemented before starting with neural network modeling and mentions reasons on why finite element, multi-body or neural networks had to be used and discusses the cons of using finite element analysis to predict stresses. It discusses special neural network configurations like committee, stacked generalization and negative correlation learning. It also introduces a new phenomenon of choosing the best committee out of all possible permutations. Finally it presents a new kind of residual analysis.

Chapter 3 illustrates all the results obtained using the above techniques. It portrays the final results and also shows a table which contain the all the best results obtained using different configurations.

1. INTRODUCTION

1.1 Problem Description

In the process of musculoskeletal modeling, multi-body analysis is usually used to predict body level information (e.g. reaction forces). But since the body parts are assume to be rigid and due to usage of simplified representation of joints, multi-body analysis is limited to only provide this information and not beyond. This inability of Multi-body models to provide low-level information (stress, strain, etc.) recommends the need to use finite element models which effectively perform any operation that multi-body analysis can in addition to predicting low-level or tissue-level information which multi-body cannot.

However using finite element analysis is not always the best choice due to highly increased computational time and complexity. To overcome these problems and develop models which can provide realistic details while being computationally efficient, surrogate or multi-scale models are developed such that they effectively use the functionalities of both multi-body and finite-element models. In this process, multi-body models first provide the joint or body level information as input to the finite element models. Finite element models, in turn, use meshes and boundary conditions to calculate internal stresses. The ultimate objective is to formulate real-life movement in real-time.

Nevertheless this kind of surrogate model is still not an optimum solution as it involves the complexity of finite element models. An alternative to this is to use artificial neural networks, built using the reaction forces from multi-body analysis, to predict the

stress information. Over the decade, neural networks have been proved to be universal function approximators and have been successfully implemented in analysis and prediction tasks especially in biomechanical modeling applications. Their adaptive learning ability to learn from examples, simple implementation techniques and fast simulation times make neural networks versatile, durable and robust.

The prime motive of this project is to perform a comparative study of several neural network surrogate models and present them as successful predictors of tissue-level information.

1.2 Musculoskeletal System

The musculoskeletal system is an integrated system formed by bones, muscles, and joints. Since movement is due to muscle forces acting on bones, it is important to understand how muscles, bones, and joints interact. Joints are interconnection areas, where bones join with each other and aid in movement. There are several kinds of joints in a human body: fibrous, cartilaginous, and synovial joints, which aid in no, minimal, and free movement, respectively. The best examples of fibrous joints are the joints in the skull, which produce no movement at all; while the spine joints produce very little movement. However, the hand and knee joints produce free movement and are the best examples of the synovial joints. Synovial joints [33] (or diarthrosis joints) are the places where free movement actually occurs. This is due to the presence of synovial cavities, which aid in free movement, between the articulating bones. These cavities contain synovial fluid secreted by the synovial membrane, which lubricates the joints and reduce

friction. Thus more presence of this fluid (produced in excess by good exercise) eases the stress on the joints and, thus, makes movement free [Gerard J. Tortora et al 2010].

The main internal structures in the human joint that keep it intact are ligaments, tendons, and articular cartilage. Figure 2 shows the anatomy of the human knee. Articular cartilage connects bones together. Its primary purpose is to protect the bones, while allowing the joint to move freely. Since articular cartilage doesn't have any blood supply, it lacks self-repair capability. Absence of cartilage would eventually result in bones grinding against each other, ultimately resulting in wear. Ligaments connect bones to bones at joints, and their ultimate aim is to keep the bones and joints intact and under control.

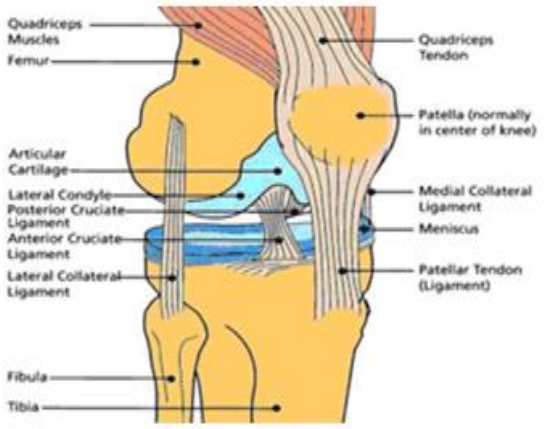


Figure 1-1 Figure 1-2

Images taken from Gerard J. Tortora et al, 2010 [33]

Figure 1-1: Musculoskeletal model of lower limb showing the main muscles and bones

Figure 1-2: Anatomy of human knee

Muscles are connected to the bones via tendons. When muscle contracts to make a movement, tendon acts on the bone accordingly. When muscles produce forces, two kinds of elements – contractile or active and non-contractile or passive elements – contribute. Passive elements include the tendon (which connects bone to muscle), perimysium, endomysium, and epimysium (which are various levels of coverings of the muscle.).

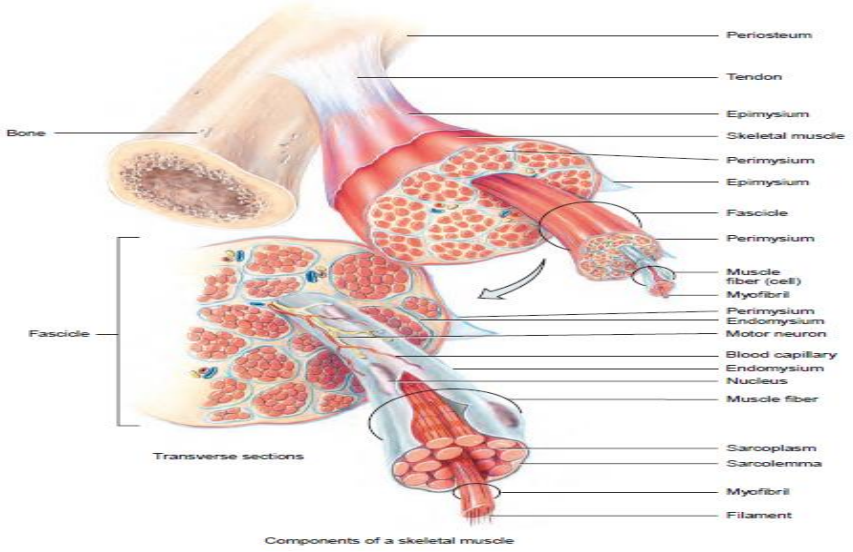
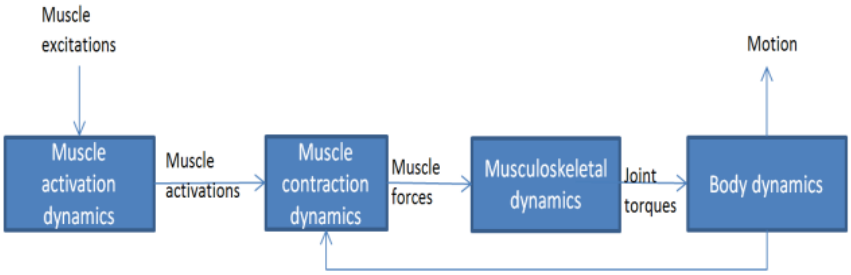


Figure 1-3: Components of muscle

Image taken from Gerard J. Tortora et al, 2010 [33]

1.3 Biomechanics of Human Movement

The process depicted in Figure 1-4 starts from the analysis of muscle excitations which can be observed from electromyography (EMG) signals. These muscle excitations are converted into muscle activations, muscle forces and finally into net joint torques. Each of the steps is explained in following phases



.

Figure 1-4: Biomechanics of human movement

Reference from [16]

1.3.1 Muscle Activation Dynamics

Electromyography (EMG) signals are used to measure electric signals generated during muscle contraction. However, these are obtained using various invasive and non-invasive methods [28], which are not readily usable, due to the presence of unwanted noise, and thus, pre-processing the signal is important. Pre-processing takes place in several steps [3]. Firstly, all the DC offsets are removed. Then, the EMG signal is normalized by dividing it with the peak value. Finally, these rectified signals are low-pass filtered, so that the signal can be correlated with the muscle force. In frequency domain, Fast Fourier Transformation (FFT) of EMG signals aid in determining the frequency spectrum [9], which helps in detecting muscle fatigue [9] and recognition of bent angles in fingers, using feature set [9].

1.3.2 Muscle Contraction Dynamics

A.V. Hill proposed an empirical model of analyzing the muscle contractions using the force-length and force-velocity characteristics. The isometric force-length curve, shown in Figure 1-5, depicts the range of forces generated by the muscle, when held at various

lengths. Active muscle force is a steady force developed on the complete activation of a muscle. It is defined as the change in the amount of force developed when the muscle is become active. In most scenarios, active muscle force is generated in the region $0.5\, l_{om} < l_m < 0.5\, l_{om}$, where l_{om} is the optimal muscle fiber length, and F_{om} is the maximum possible isometric force developed. From the force-velocity curve (Hill type model), it can be inferred that there is a hyperbolic dependency of the force on the velocity.

The main parameters that describe the muscle's force producing properties are F_{om}, l_{om}, v_{max}, muscle activation, a_m (which is derived from activation dynamics),and pennation angle(which is defined as the angle between muscle fibers and tendons when fibers are at optimal length).

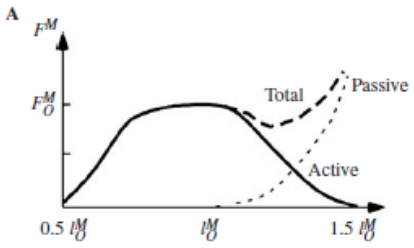


FIGURE 7.3 Force-length curve for muscle. (*a*) Isometric force-length properties when muscle is fully activated.

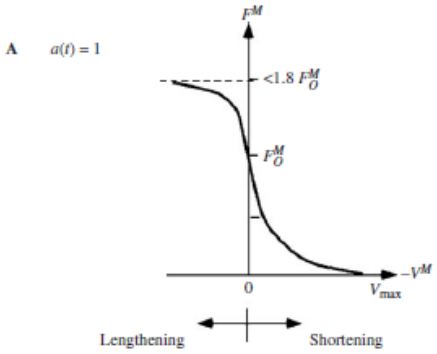


FIGURE 7.4 Force-velocity curve for muscle when (*a*) muscle tissue is fully activated,

Figure 1-5: Force-length-velocity curves

Image taken from Myer Kutz et al, 2009 [16]

1.3.3 Musculoskeletal Dynamics

Biomechanical movements in a human body occur due to a number of forces acting on various bones, which cause them to rotate about the joints due to application of torques. The joint torque is a collection of all the individual muscle forces. These forces come mainly from muscle contractions, and thus, it is vital to understand the force-producing properties of muscles, mainly length and velocity, to analyze motion.

The moment arm of a system is the perpendicular direction to the line of action of force, and hence, it changes with the angle of application of force.

For a given musculotendon force F^{MT} and moment arm r^{MT}, the musculotendon torque T^{MT} is given by:

$$T^{MT} = r^{MT} * F^{MT}$$

In Figure7, r1, r2, r3, and r4 are various moments acting on force F while r2 (the perpendicular distance to the force F) is the moment arm.

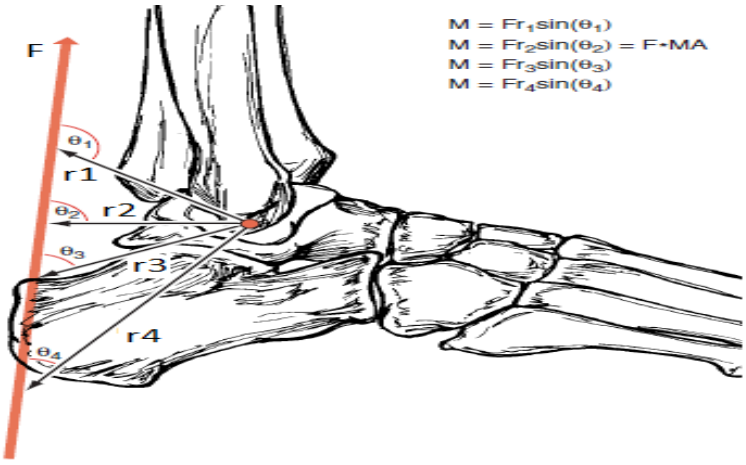


Figure 1-6: Two-dimensional moment analysis.

Image taken from Andrew R. Karduna, 2009 [14]

From the equations of motion, the relationship between movement and muscle forces in a musculoskeletal model can be given by:

$$M(q)\ddot{q} + C(q,\dot{q}) + G(q) + R(q)F^{MT} + E = 0 \tag{1.1}$$

[Taken from Ahmet Erdemir et. al 2006]

where $M(q)$ is the mass matrix, C represents the centrifugal forces, and torques, $G(q)$ is the gravitational loading, E represents the external forces. q and its derivatives refer to the positions, velocities, and accelerations.

Using the definition of musculotendon torque, we have:

$$M(q)\ddot{q} + C(q,\dot{q}) + G(q) + T^{MT} + E = 0 \tag{1.2}$$

[Taken from Ahmet Erdemir et. al 2006]

It is an indeterminate problem to calculate the individual muscle forces from joint torques as the number of possible solutions is infinite. Hence, the primary objective is to first figure the net joint torques and then use optimization methods to get the optimal solution of muscle forces.

1.4 Surrogate Models

1.4.1 Multi-Body Model

Multi-Body models have been used extensively for many decades to understand the behavior of human musculoskeletal models and, specifically, human joints [Thomas Buchanan et. al 2004]. Various parts of human body (for instance the knee, elbow etc.) are modeled according to the requirement, such that body level information can be

garnered. [Otten 2003] mentions several mathematical expressions using Newton-Euler, Lagrange and Featherstone's methods to represent multi-body systems. Inverse and Forward dynamics discussed below are two ways of implementing the Multi-body model. Some notable software includes MSC.ADAMS (MSC Software Corporation, Santa Ana, CA) and AnyBody (AnyBody Technology) where ADAMS is a commercial rigid body dynamic modeling software package and AnyBody is a musculoskeletal modeling software package.

1.4.2 Inverse and Forward Dynamics

Theoretically speaking, analysis of movement can be done in two ways. One way is to predict the movement, based on known internal muscle forces and joint moments (forward dynamics approach), and the other is to calculate the joint moment torques, based on movement data and external forces (inverse dynamics approach). It depends purely on requirement which methodology to choose. If motion is to be estimated, forward dynamics approach is the best bet. To estimate the joint torques from motion and external forces, inverse dynamics would be the best choice [3, 7, 24].

1.4.1.1. Inverse Dynamics

In inverse dynamics, estimates of joint torques are given from movements and known external forces. Implementation of inverse dynamics approach starts with measuring the properties, like mass and inertia, of the body of interest and the external forces acting on it. Using equations of motion, joint torques are computed. Individual muscle forces are then estimated from net torques at the respective joints using static optimization methods [24]. Though mass is easily computable, inertia and other

properties of the body are hard to calculate and hence estimation of joint torques is not trivial. Next is the problem of deriving individual muscle forces from joint torques as this problem is indeterminate. Static optimization methods are used to solve this problem.

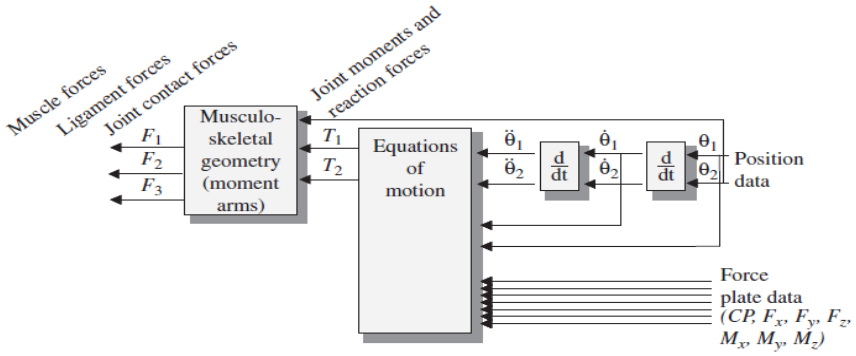


Figure 1-7: Inverse dynamics approach

Image taken from Thomas S. Buchanan et al, 2004 [3]

1.4.1.2. Forward Dynamics

In forward dynamics approach, known internal muscle forces are used to create motion. Neural commands, obtained using EMG signals or by optimization techniques, produce the muscle activations (muscle activation dynamics), and muscle contraction dynamics convert the activations into muscle forces. Forces in the individual muscles contribute to the net joint torques and moments. Multi-joint dynamics then compute the accelerations, velocities, and angles for each joint of interest. Finally the feedback loop updates the neural command based on the musculotendon length and the pennation angles. Figure 1-8 gives high-level information of all of the steps involved in forward dynamics problem.

[Thomas S. Buchanan et. al, 2004] discuss various problems involved with inverse and forward dynamics. Estimation of muscle activation from neural commands and muscle forces from the activations are not easy. However, using optimization methods can directly provide muscle forces [3]. Nevertheless, choosing cost functions for the optimization methods is difficult. The other main limitation is that the difficulty in estimation of joint moments, as a minor inaccuracy might lead to abnormal errors in the movements.

Forward dynamics assisted data tracking [6] is an optimization solution to estimate muscle forces. In this process, a solution is first obtained using default muscle activations, and then, the process is iterated by updating activations, until the performance function is minimized as per requirement

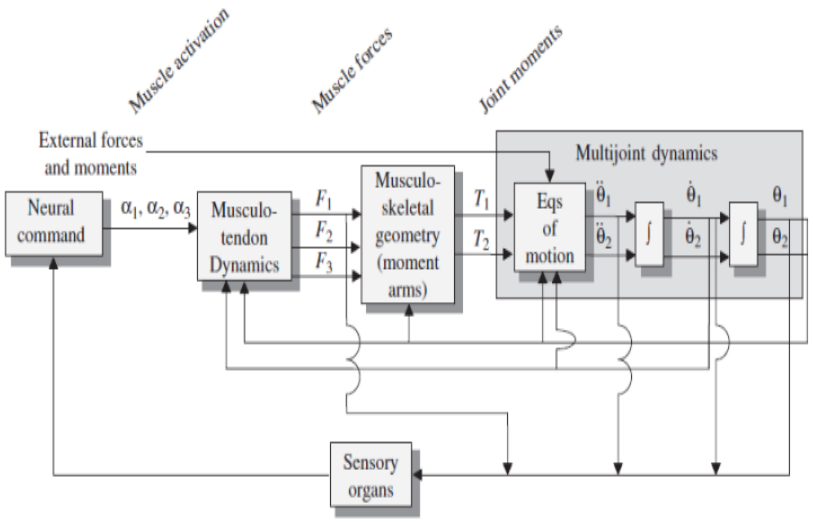


Figure 1-8: Forward dynamics approach

Image taken from Thomas S. Buchanan et al, 2004 [3]

1.4.3 Finite Element Analysis

Finite element analysis is one of the fastest emerging methodologies widely being used in the biomechanical field for computing stresses and strains in cardiovascular, dental, articular cartilage, etc. By definition, FEA is a numerical method for finding solutions to complex problems. Once the problem set is defined, symmetry is applied to reduce the complexity. Then, meshes are used to discretize the model into elements and nodes. Choosing mesh sizes is a known challenge, and the most widely used shapes are tetrahedral or hexahedral for 3D objects. Then, the shape vectors, stiffness matrices, and force vectors are computed. Finally, boundary conditions are applied, using known external applied forces, and rigid body dynamic conditions and stresses are computed.

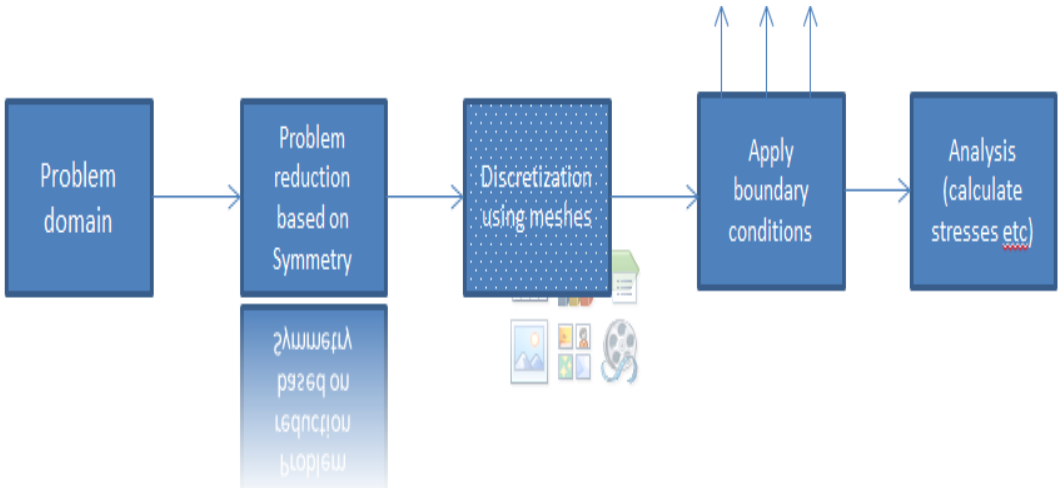


Figure 1-9: Finite element analysis steps

Currently, we have a lot of software that perform finite element analysis for complex problems, which are manually not possible. Some notable software includes ABAQUS (SIMULIA, Dassault Systèmes SolidWorks Corp., S. A., Vélizy, France),

SOLIDWORKS (Dassault Systèmes SolidWorks Corp., S. A,Vélizy, France), LS-DYNA (Livermore Software Technology Corporation), and many more.

1.5 Need for Surrogate Models

Multi-body models have always been a compromise, as they cannot genuinely provide low-level information (tissue-level stress, strain etc.) of a human body. This is due to the usage of simplified representation of joints. Such assumptions limit the usage of MB models, as they do not provide tissue level information for studies like the ligament injuries. On the other hand, Finite Element models can provide tissue-level, organ-level details, but not body-level and are computation intensive. To overcome these problems and develop models, which can provide realistic details while being computationally efficient, multi-scale models are developed such that they effectively use both MB and FE models. Using multi-scale models, detailed low level information and even movement prediction can be obtained. MB models provide the joint or body level information as input to the FE models. FE models, in turn, use meshes and boundary conditions to calculate internal stresses. The ultimate objective is to predict stress on tissue such as cartilage and ligaments during movement and achieve concurrent simulation of muscle force and tissue stress [22].

1.6 Implementation of Surrogate Models

Coupling of MB and FE models can happen in two ways: sequential and concurrent. In the sequential approach, simulated results from the MB model are used as

input to the FE model and ultimately tissue-level stresses are obtained. In the concurrent method, the reaction forces obtained by FE analysis are applied to the MB model iteratively. An application of the non-sequential approach was demonstrated by [Fernandez et al. 2006], where muscle forces derived using computational MB models were used as inputs for FE models to calculate effective stresses [7].However these models did not make use of the coupling and *concurrent* behavior of MB and FE models.

In their research, [Viceconti et al, 2008] presented a Living Human Multi-scale model [34], which can predict the risk of fracture. They discussed various methods to develop subject specific FEM of bones and estimation of internal forces and stresses of musculoskeletal models using optimization methods [6]. [Tahwai et al, 2009] put forward the applications of multi-scale models to musculoskeletal, respiratory and mechano-transduction systems, where the FE models used a feature size field [32] during mesh generation. Feature size fields are robust and computationally efficient and do not require a background grid during startup.

A notable approach by [Halloran et al. 2009]successfully developed an adaptive surrogate multi-scale model using concurrent coupling, which simulated jumping, used the Lazy Learning Toolbox [10] (a locally weighted regression algorithm) written in MATLAB® for surrogate modeling. This approach is different from traditional multi-scale modeling approaches as it trains the surrogate model based on previous FEA runs. These surrogate model outputs are compared with FE outputs, and if the error is in user specified range, the surrogate model output is used for future iterations. This way, the need for FE simulations is minimized, which in turn, increases computational efficiency.

However, this approach is not practically relevant as friction is completely ignored and real-time movements are much more complex and local neighbor search methods (for instance Lazy Learning interpolation) would not be sufficient for a good analysis. In an upgrade to their above work, [Halloran et al. 2010] successfully presented an approach to concurrently couple musculoskeletal and tissue deformation models using strain optimization[11], which aided in lesser number of iterations to attain convergence, rather than local neighbor search algorithms[10]. However, friction was considered globally for musculoskeletal models and not for the FE model specifically, to reduce the complexity, which doesn't make it realistic, as practical systems experience friction globally as well as internally.

Another approach for surrogate modeling is to replace the FE model with several data-driven models [23, 25, 30], which may include various kinds of neural network (NN) models. NN models have proven to be capable of modeling realistic biomechanical interactions [23, 25, 30]. Once trained (usually with internal muscle forces from MB systems), they take very little time to perform the analysis. Inputs to the models are usually the forces and positions from the MB models and outputs are the stresses and strains. The inputs and outputs may vary according to scenario.

1.7 System Identification

Mathematical models depict the behavior of systems using mathematical terms. System Identification is a procedure to create mathematical models of dynamics systems from observed user data. It also gives exemplary information [20] on various tools for effective data fitting using the models. To perform the System Identification process two methods are used – Non-Parametric and Parametric -where the former give high level information about the system like time delays, time constants, gains etc., while the parametric methods involve the process of estimating several parameters to obtain a good model of the system.

1.7.1 White-Box, Grey-Box and Black-Box Modeling

In white-box modeling, models are created using basic laws of dynamics, and therefore, they are very complex to build in practical scenarios, like the study of biomechanics. This technique is mostly used during testing and is generally referred to as theoretical modeling. A good example is electrical circuit analysis, where parameters are known, and the output parameters (voltages et al.) are found.

To overcome the complex nature of white-box systems, two System Identification approaches are used, namely grey-box and black-box, which build models from user inputs and behavior of the system. In Grey-box models, a model is first created using few known parameters, and the unknowns are estimated using various System Identification tools with several assumptions.

Black-box models, unlike grey-box models, have little or no information about the mathematics or the behavior of the system, and various tools in SI toolbox are used

for data fit and parameter estimation. However, they require estimations on the model orders. All the tools discussed below follow the black-box approach.

1.7.2 Steps in System Identification

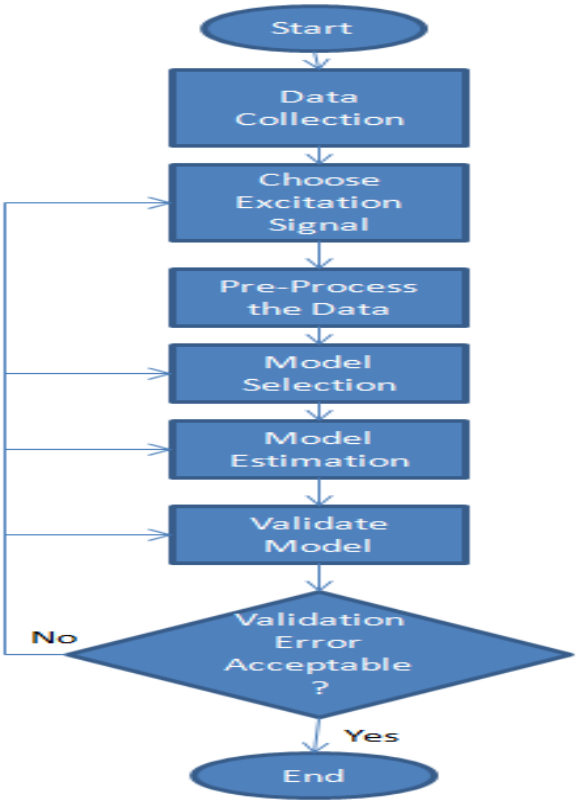


Figure 1-10: System identification Process

1.7.3 Choosing Excitation Signals

One of the foremost and most important steps in the system identification process is the choice of good excitation signals (also referred to as exploration, pilot or stimulus signals) and knowing how the model reacts to various inputs. It is truly mentioned [31] that the kind of excitation depends on the model and analysis chosen. Correctly excited

models provide deeper and much better insight on the system properties and should be able to excite all the relevant frequencies such that maximum information can be garnered. Hence a good excitation signal should be able to portray all the operating points of the system. As discussed in earlier sections, dynamic systems are first analyzed using non-parametric methods (transient analysis, impulse response analysis, frequency analysis) to gain a deeper insight on the system's internal properties like the gain, overshoot, time constant, damping factor and other important parameters.

Normally used excitation signals are the step, impulse, square [1] and triangular waves. Apart from simple excitations like step and impulse used for non-parametric analysis, others forms of excitations are used for parametric methods. Examples are the PRBS, Chirp and Gaussian noise signal which serve as good excitations for parametric models. [Pintelon et. al 2012] discuss about various input signals having different crest factors and signal-to-noise ratios (SNR). Crest factor denotes compactness of the signal [26] and is defined as the ratio of the peak value of the signal to its root mean square (RMS) value. SNR ratio compares the strength of a signal with the disturbances acting on it. High SNR values imply the dominance of the signal over the disturbances and lower values depict relatively high noise content. Other signals include an autoregressive moving average sequence [31], sum of sinusoids [31].

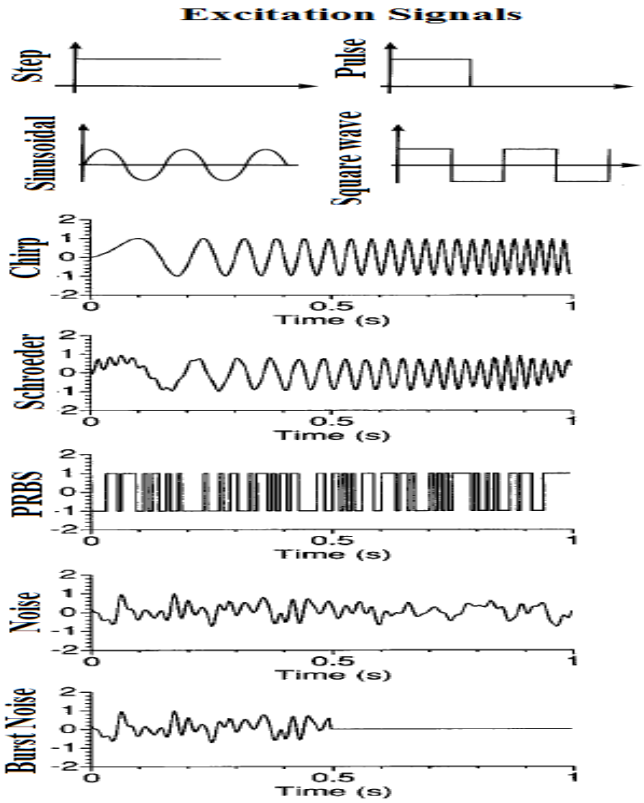


Figure 1-11: Excitation Signals

Image taken from [Isermann et. al 2011 pp. 22] and [Pintelon et. al 2011 pp. 161]

1.7.4 Model Selection

After creating a model using any of the algorithms discussed in the upcoming sections, the next step is to find the simplest system that can ideally fit the system dynamics. This can be done either by comparing the desired and obtained model output (MATLAB® command compare), simulating and predicting the response (MATLAB® commands sim, predict), or by comparison of transient, frequency and impulse response

analysis of various models. Impulse and Step responses aid in model validation by providing characteristics [20] like the peak response, system gain and the settling time, whereas frequency response provides the peak response frequency and stability margins [20]. Analyzing the nonlinear plots gives a deeper insight of nonlinear models on the particular inputs, depicting which range of inputs contribute the most or least in the system response. Also, the flexibility to determine the conditions for validation stop is helpful for good validation. Other important usable performance metrics are discussed below.

1.7.4.1. Distance Metrics

Using various distance metrics, it is possible to know the distance between two datasets (measured vs. actual outputs). Commonly used distance metrics are *Manhattan*, *Euclidean* and *Mahalanobis* distance.

Manhattan distance is one of the simplest distance measures and is calculated as the direct difference between two vectors.

$$\text{Manhattan distance } = \sum_{i=0}^{n} (yi - yi')$$

Euclidean distance is the traditional distance measuring tool having the following formula.

$$\text{Euclidean distance } = \sum_{i=0}^{n} \sqrt{(yi - yi')^{T}(yi - yi')}$$

It is notable from the formulae that all points in Euclidean and Manhattan distances contribute equally to the distance measure but do not consider the deviation of

each point from the average. Mahalanobis distance is another commonly used distance measure which takes into account the covariance of the vectors under consideration and therefore is useful in measuring the dissimilarities between datasets.

$$\text{Mahalanobis distance } = \sum_{i=0}^{n} \sqrt{(yi - yi')^T \in (yi - yi')}$$

where $\in$ is the inverted covariance between y and y'. If y and y' have unit covariance, Mahalanobis distance becomes equal to the Euclidean distance.

1.7.4.2. Mean Squared Error

The simulated outputs (y) can be compared with the desired outputs (y') and the mean squared error calculates the mean of squares of differences between the predicted and actual output.

$$\text{Regular MSE} = \frac{1}{n}\sum_{i=0}^{n} (yi - yi')^2$$

The traditional form of MSE can be extended by taking square root (RMSE) and by normalizing it (NMSE). The combination of both is NRMSE discussed below.

Normalized Root Mean Squared Error (NRMSE)

Regular MSE calculates the total error as a whole, comprising of errors due to all inaccuracies, but does not give information on how the measured output deviates from the desired value. Normalized MSE is obtained by dividing the squared sum of errors with the variance. Normalized Root MSE advances a step by taking square root of NMSE and is given by

$$\mathrm{NRMSE} = \sqrt{\frac{\sum_{i=0}^{n}(yi - yi')^2}{\sum_{i=0}^{n}(yi - yavg)^2}}$$

The main advantage of using root is to get the performance in the same units as the data.

1.7.4.3. Residual Analysis

Residual analysis is an important phase of model validation which deals with analysis of residuals (errors), which are the differences between the measured and one-step-ahead predicted model outputs. Analyzing residual plots gives information on which specific inputs caused the perturbations.

$$\epsilon = yi - yi'$$

1.7.4.4. Correlation Coefficient (R)

Correlation Coefficient (R) is a measure of how well or poorly two data sets are correlated. It is given by the below formula. The denominator is the product of standard deviations of the desired and actual outputs while the numerator is the covariance of the outputs.

$$R = \frac{\sum_{i=0}^{n}(yi - yavg)(yi' - yavg')}{\sqrt{\sum_{i=0}^{n}(yi - yavg)^2 \sum_{i=0}^{n}(yi' - yavg')^2}}$$

where y is the measured output, y' is the desired output and y_{avg}, y_{avg}' are the averages of data points in y and y' respectively. An R value can have value ranging from -1 to 1, where -1 indicates a perfect decreasing linear relation such that all data points of actual output lie on the negative slope as of the data points line of desired output, 0 indicates a null fit and 1 indicates a perfect linear relation.

1.7.4.5. Minimum Description Length (MDL)

Minimum Description Length is a measure of how compactly regularities can be described in a dataset. According to MDL, a model is chosen if it minimizes the data in the model and the parameters governing the data. One implementation of MDL is given by the following expression. A model with lower MDL is considered optimum as it minimizes both the parameter usage for system analysis and the total data points itself.

$$\mathrm{MDL} = -\log \mathrm{P}\left(\frac{\mathrm{k}}{\mathrm{model}}\right) - \log\ \mathrm{P}(\mathrm{model})$$

where $\mathrm{P}\left(\frac{\mathrm{k}}{\mathrm{model}}\right)$ is the probability of parameter usage to describe the system and $\mathrm{P}(\mathrm{model})$ is the number of bits to describe the model

1.7.4.6. Information Criterion

Information criteria are likelihood-based measures which take into account the complexity of the model. Therefore a model which performs good analysis but is computationally complex is not considered the optimum model. Information criteria consider all the information, analyzing capability and complexity, while performing the analysis. Different kinds of information criteria are Akaike Information Criteria (AIC) and Bayesian or Schwarz Information Criteria (BIC). If L(k) is the maximum likelihood of the model, log L(k) is the objective function of maximum likelihood, N is the total number of data points, and k is the total number of parameters actually used in fitting the data, the definitions of AIC and BIC are

$$\mathrm{AIC} = -2\ln \mathrm{L}(\mathrm{k}) + 2\mathrm{k}$$

$$\mathrm{BIC} = -2\ln \mathrm{L}(\mathrm{k}) + \mathrm{k}\log \mathrm{N}$$

When multiple models are compared, models with the lowest values of AIC and/or BIC are the ones which provide an optimum analysis. From the above definitions, the loss function L is an increasing function (logarithm) of the number of parameters influencing the fit and a high value of AIC indicates over fitting. BIC has a larger penalty order of log N for an extra number of parameters when compared to AIC.

1.7.5 Dynamic Models

Dynamical models represent systems with internal dynamics based on the previous state and are time and frequency variant. Therefore, outputs of a dynamic system depend on the inputs at past time instants. Relations can be created using differential equations, discrete-time, or continuous-time data.

1.7.5.1. Linear Analysis

To estimate system dynamics, an analysis should always start linearly. However, if linear models do not provide acceptable results, employing non-linear models is the next step. Estimation using linear models is done using two approaches – non-parametric ,or direct estimation using impulse and frequency response models, and parametric, which includes estimating a set of parameters to identify a model. Non-parametric methods are first employed to determine whether a system is linear, time invariant, and if any noise exists. These experiments are generally considered as pre-processors to the parametric methods as they are evaluated at large number of points and give a good analysis of the system. Various implementations of the latter approach are Transfer

function model, Process model, State-space model, Polynomial model and Grey-box models. A linear system can be represented by the differential equation:

$$\frac{dy^n}{dt^n} + a1\frac{dy^{n-1}}{dt^{n-1}} + \cdots = b0\frac{du^m}{dt^m} + b1\frac{du^{m-1}}{dt^{m-1}} + \cdots$$

In parametric form, the above can be analyzed as

$$A(q)y(t) = B(q)u(t) + H(q)e(t)$$

where $o(t)$ is the final output, $y(t)$ is the definition done below, $e(t)$ is the noise function, q is the delay operator such that $q^{-1}u(t) = u(t-1)$, and A, B, H are the parameters to be estimated during parametric analysis. This is done by minimizing the error function using sum of least squares algorithm. This process is termed prediction error method.

Impulse-Response Models

Impulse responses are the output signals obtained by applying an impulse input. An impulse function or the Dirac Delta function has value of zero at real-time, except at time zero where it is infinite. For finite impulse response models, the impulse function will have a finite value at time zero. For an impulse input u (t), the response obtained is the convolution of the impulse response:

$$y(t) = \sum_{k=1}^{n} gk\, u(t-k)$$

where g_k is the impulse response, $y(t)$ is the system output, and $u(t-k)$ is the shifted impulse input. The ultimate goal is to find the g_k values using correlation analysis (linear least squares method).

Frequency-Response Models and Spectral Analysis

Frequency response describes how a model responds to sinusoidal inputs. Frequency response models are obtained by applying Laplace transforms [20] of the impulse response discussed before or by evaluating the transfer function $G(z)$ on a unit circle [20]. One approach is by computing the Fast Fourier Transform (FFT) of the signal or by analyzing the bode plot. Change in amplitude and phase shift are two vital variables which govern the frequency response characteristics.

$$Y(z) = G(z)U(z)$$

SI Toolbox has three functions to compute the frequency response – etfe, which computes the empirical transfer function (ratio of the Fourier transform of the input and output), and the rest are discussed below. As per spectral analysis [20], any function can be analyzed using sine and cosine waves. Hence, $G(z)$above, can be written as $G(ejw)$. Frequency resolution is the smallest allowable frequency at which frequency response and spectral analysis can be performed. MATLAB® command spa estimates the transfer function using Blackman-Tukey [20] spectral analysis for a fixed frequency resolution and spafdr [20] allows specifying variable frequency resolution for estimating the frequency response.

Transfer Function Models

Transfer functions models are mathematical models obtained by taking the ratio of the output and the input polynomials, and the model order is the order of the input signal. They are different from the polynomial models, as they provide deterministic

analysis, rather than stochastic analysis that is given by the latter. Transfer function models can be effectively used to model single and multiple input systems.

Applying Laplace transforms to (1), (ignoring the error component)

$$s^nY(s) + s^{n-1}a1Y(s) + s^{n-2}a2Y(s) + \cdots = s^nU(s) + s^{n-1}b1U(s) + s^{n-2}b2U(s) + \cdots$$

The above can be represented as

$$Y(s) = G(s)U(s)$$

where G(s) is the ratio of two polynomials and it comes in the form

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^n + s^{n-1}b1 + s^{n-2}b2 + \cdots}{s^n + s^{n-1}a1 + s^{n-2}a2 + \cdots}$$

Y(s) and U(s) are the Laplace transforms of the output, input, and the error signals.

This can be solved by partial fractions by dividing the block into two parts as shown in the below block diagram below (f and r represent polynomials in s).

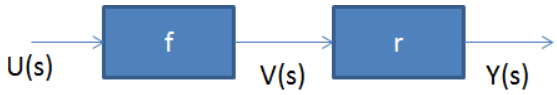


Figure 1-12: Block diagram representation of Transfer function models

$$G(s) = \frac{Y(s)}{U(s)} = \frac{Y(s)}{V(s)}\frac{V(s)}{U(s)}$$

It is evident from the above representations that for complex systems, the factorization of numerator and denominator polynomials (f and r) becomes complicated and obtaining partial fractions is a difficult job.

MATLAB® command, etfe, estimates the empirical transfer function, G, from above equation. Flexibility is also given to include spectral analysis with (spafdr) and

without (spa) varying frequency resolution. Once the transfer function is known, system analysis becomes trivial, as the output response can be studied over a varied range of inputs.

Process Models (Low Order Transfer Function Models)

Process models are simple systems used to estimate system dynamics in terms of system gains or the transport delays [20]. Upgrading the model is feasible, as it only involves modifying the poles, zeros, or the delays. A process model describes the system dynamics in terms of system gains, delays, and other time constants (shown in below equation). It can estimate up to third order transfer function models. Owing to their low order, they are easy to develop, and parameters are easy to estimate. Another advantage of these models is that they support transport delays [20].

Polynomial Models

In Polynomial models the relation between input and outputs depend on transfer functions. If $y(t)$ is the system output, $u(t)$is the system input, q-1 is the delay operator, and e(t) is the error function polynomial models are generally of the following form:

$$y(t)A(q) = \sum_{i=1}^{n} u(t-i)\frac{Bi(q)}{Fi(q)} + e(t)\frac{Ci(q)}{Di(q)}$$

Image taken from [28], pp.2.23 (3)

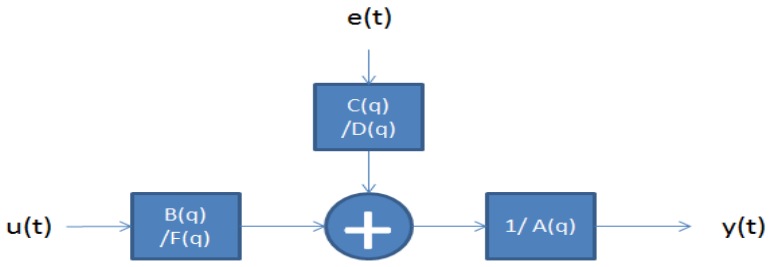


Figure 1-13: Block diagram representation of polynomial models

These polynomial models are obtained by simplifying the original equation (setting either of A, B, C, D, E, and F to 1).All the polynomial models discussed below are basically the linear regression technique and can be analyzed using the least squares mechanism.

Auto Regressive (AR) models

The AR model is used when outputs are only dependent on the previous outputs and inputs are not known. An AR can be defined by the following equation (n is the model order)

$$y(t) = \sum_{q=1}^{n} y(t-i)A(q) + e(t)$$

The problem in AR model is to find the best possible values of the parameter "a" which best describe the series. The most widely used technique to achieve this is to use the least squares estimation.

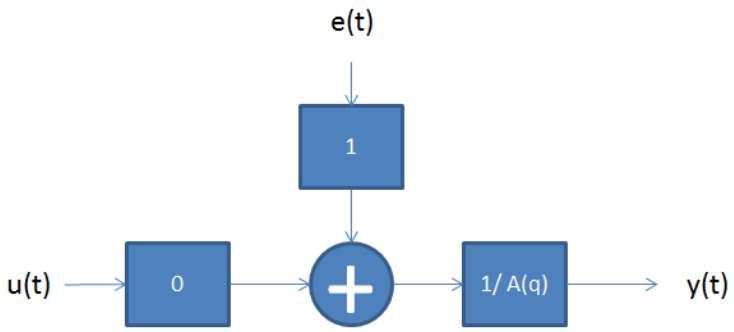


Figure 1-14: Block diagram representation of AR models

Auto Regressive with Exogenous Inputs (ARX) models

An ARX model implements the least squares estimation and is used to obtain model analysis without providing much flexibility to the noise, as they do not model dynamic disturbances that might occur in the model. In addition, modeling is not very flexible as poles (A(q)) of the dynamic system and noise coincide [20], which is upgraded in BJ and OE models.

$$y(t)A(q) = \sum_{q=1}^{n} u(t-i)B(q) + e(t)$$

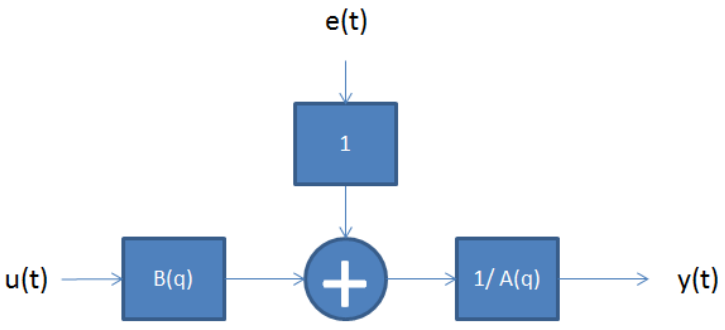


Figure 1-15: Block diagram representation of ARX models

Auto Regressive and Moving Average with Exogenous inputs (ARMAX) models

Unlike AR and ARX models, ARMAX uses a new modeling technique called Moving Average which is another form of linear regression but the estimation of parameters becomes a difficult task to be achieved using least squares estimation due to the extra focus on the error dynamics. ARMAX extends the flexibility by modeling noise separately using the C(q) term.

$$y(t)A(q) = \sum_{q=1}^{n} u(t-i)B(q) + C(q)e(t)$$

ARMAX can be used when unavoidable disturbances [20] occur at the input. There are two more models, ARIX and ARIMAX, which add additional integrators [20] to the noise $e(t)$. Adding additional integrators help in cases where the perturbations are dynamic.

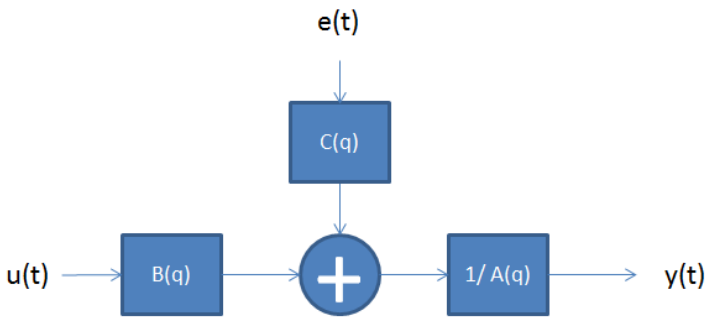


Figure 1-16: Block diagram representation of ARMAX models

Output Error (OE) models

Output Error model, like ARX, doesn't give much flexibility to the noise analysis, but gives additional emphasis on parameterizing the dynamics evident from B(q) and F(q) terms. Therefore these models would be preferable in cases where the noise properties are not given much importance and the sole focus is on the system dynamics.

$$y(t) = \sum_{i=1}^{n} u(t-i)\frac{Bi(q)}{Fi(q)} + e(t)$$

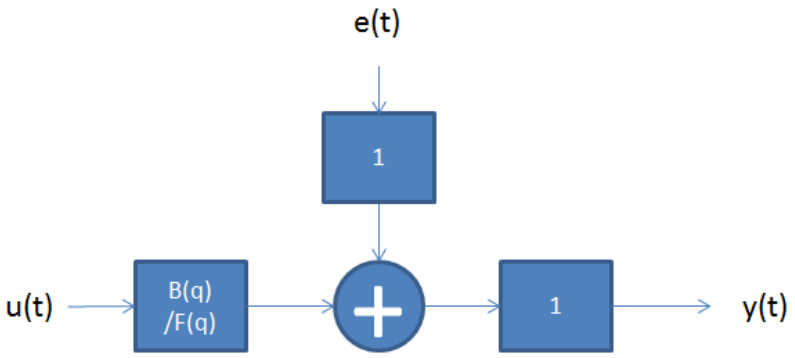


Figure 1-17: Block diagram representation of OE models

In their research on Output error models, [Urban Forssel, Lennart Ljung 1997] provided means to model alternative techniques [8] of Output error and Box Jenkins models (discussed below) to analyze linear and unstable systems. They discuss reasons as to when to choose an Output error model over Box Jenkins and vice versa.

Box Jenkins (BJ) model

Box Jenkins model is an improvement over ARX and ARMAX and separates the system dynamics and disturbances. It extends the properties of both AR and MA models,

and also increases the parameterization. However it requires a lot of data in discrete-time series, without any missing values [15].

$$y(t) = \sum_{i=1}^{n} u(t-i)\frac{Bi(q)}{Fi(q)} + e(t)\frac{Ci(q)}{Di(q)}$$

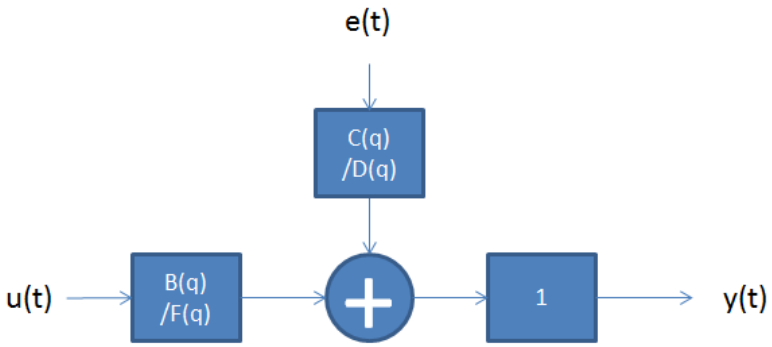


Figure 1-18: Block diagram representation of BJ models

All the above models (except ARX) can be developed using a single command polyest in MATLAB, which can estimate any polynomial model using iterative least square algorithm. These polynomial models provide more information stochastically but require excessive computation time. The first step is to estimate the model orders and delays. The command combination struc – arxstruc – selstruc gives the estimated model order and delay in MATLAB. In this process, different combinations of model orders and delays are used to estimate the models, and the best fit is finalized as the optimal model order and delay.

State-Space Models

State-space models are commonly used to describe linear relationships between inputs and the outputs, and are usually well-suitable for most of the systems, especially the Multi Input Multi Output (MIMO) systems. They introduce variables called state variables, which can be estimated using the input-output data. These are formed either using differential equations or from the transfer function notation but not measured directly. The advantage of using these models is that they require only the model order to be estimated beforehand. In the dynamical equation below, y(t) and u(t) are the output and inputs of the system, respectively. If the derivatives of y are replaced by state variables x(t), we get equations of the following form:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y \\ dy/dt \\ \vdots \\ d^{n-1}y/dt^{n-1} \end{bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \\ 0 & & & & 1 \\ -a_n & -a_{n-1} & \dots & & -a_1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} b_n & b_{n-1} & \dots & b_1 \end{bmatrix} x + du.$$

Figure 1-19: State Space equations

Image taken from [21]

Replacing the matrix terms with variables, we can represent the state space model as

$$x(t+1) = Fx(t) + Gu(t)$$

$$y(t) = Hx(t) + Du(t)$$

The state space models are useful, when compared to other parametric models, as they can model more complex systems with high orders, where minimizing a performance function using the least squares doesn't provide acceptable results. Though state space models are efficient for MIMO systems, if the model order is too high or the data too large, the computation becomes slow and requires lot of memory. An ARX model is preferable in such cases.

[Mohsin et. al] provide a good example to model a tall structure building using ARX, ARMAX and OE models to find the best model and analyze them using auto and cross correlation analysis, frequency analysis, and pole-zero analysis [13]. Even by using all the above linear models, if the results are not acceptable due to bad fits between actual and desired outputs, the next step would be to check for nonlinearities.

1.7.5.2. Non-Linear Analysis

Linear models discussed in the previous section give a good analysis of dynamic systems. However, when a linear model is not good enough in predicting the system dynamics, the need to switch over to non-linear models arises. If the data is found to be weakly non-linear from preliminary analysis [20], its linear model can be coupled with non-linear estimation, which will explain the non-linear parameters more clearly than a linear model would. When the internal non-linear dynamics are known beforehand, they

can be modeled using Non-linear Grey Box modeling. Non-linear models usually require more data for a good analysis when compared to linear models.

Hammerstein-Wiener Models

The Hammerstein-Wiener model is a block-oriented approach [22], which represents a set of non-linear systems surrounded by a linear model. The Hammerstein model consists of a cascade connection of static nonlinearity and linear dynamic systems, while the Wiener model uses reverse coupling, such as the Hammerstein approach. Using this Hammerstein-Wiener model, the system dynamics are first represented using a linear model (transfer function for instance), and then, the nonlinearities are captured using various non-linear functions.

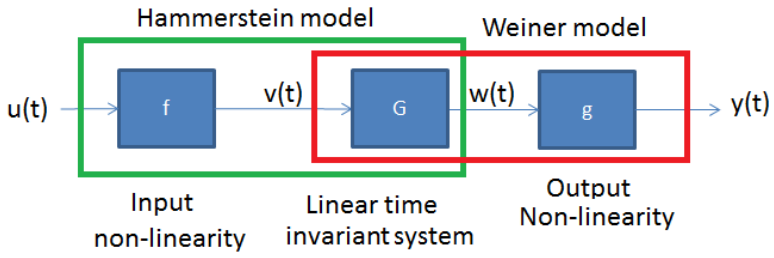


Figure 1-20: Block diagram representation of Hammerstein-Wiener models

Linear models can be represented in the form

$$y(t) = G(q)\big(v(t)\big) + e(t)$$

while Non-linear blocks take the form

$$y(t) = g\big(w(t)\big) \text{ and } v(t) = f\big(u(t)\big)$$

From the above equations, we get

$$y(t) = g(G(f(u(t)))) + e(t)$$

The parameters g, G and f can be found using iterative training algorithms like gradient-descent, generalized mean squared error, scaled conjugate gradient, Levenberg–Marquardt, etc. The process [20] starts with estimating the function f(u) using any of the above nonlinear estimators and sent to the linear block G. The output y(t) is obtained by applying appropriate non-linearity to the w(t) signal, which is in turn, the output of a linear dynamic system.

Nonlinear ARX Models

Nonlinear ARX models provide additional non-linear analysis to the linear ARX models discussed previously. A dynamic model is given by the below equation, and the current output $y(t)$ is a function of the past outputs and inputs. Liner model estimates the parameters a, b.

$$y(t) + a1\, y(t-1) + a2y(t-2) + \cdots = b0u(t) + b1\, u(t-1) + b2u(t-2) + \cdots + e(t)$$

To extend this process, nonlinear models are introduced, which go a step further by replacing the above dependency on past outputs and inputs, with a complex nonlinear function as given below (f is any nonlinear function)

$$y(t) = f(y(t-1), y(t-2), \dots, u(t), u(t-1), u(t-2) \dots + e(t)$$

Non-linear models can also be created from linear models. First, a linear model is created (using polynomial model for instance), and then, this model and the input data are provided as inputs to create a non-linear model. This approach provides better fit and analysis than the linear model.

Nonlinearity estimators

The Nonlinear models discussed above use the below estimators to find the best possible relationship between the system outputs and inputs. It takes a matter of trial-and-error as to know which estimator works the best as per scenario. The commonly used estimators are piecewise linear approximation, wave-net approximation, neural net approximation, sigmoid-net approximation, etc. These estimators are used to obtain the model outputs and can be validated (discussed in a separate section later) to obtain an appropriate model for the scenario.

1.7.6 Neural Networks

Neural Networks are computational models used for solving a variety of non-linear, static, dynamic, and pattern recognition problems and have attained success in many such areas due to their versatility in adaptive learning. The properties of a neural network like universal approximation [Luenberger 1969, Cybenko 1989], ability to learn from examples and generalize well on test data with very fast simulation times (usually in seconds), non-linear behavior and simple implementation techniques make them versatile and robust.

The main principle of working of a neural network is to learn from the inputs and predict future states or outputs. It is evident that the quality of the correctness depends on how well the network has been trained. A neuron [27] or simply referred to as a node is the basic building block of a neural network and can be defined as a processing element with a set of inputs and outputs implementing a weighted non-linear sum. A simple neuron is represented in the below figure.

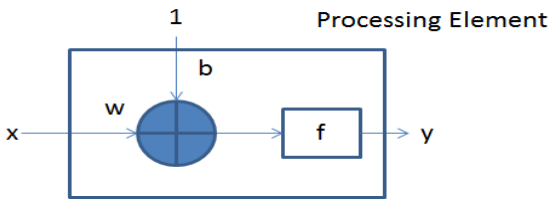
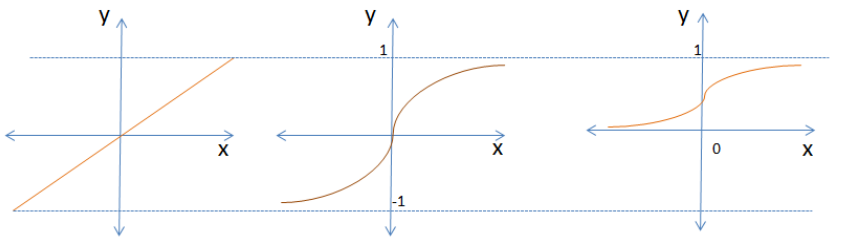


Figure 1-21: A Simple Neuron

where x, y, w and b are the input, output, weight and bias respectively. The above processing element is evaluated as $y = f(w * x + b)$ where f can be any transfer function (linear or sigmoidal). Every layer of a neural network contains three important components – combination, activation and error functions. A combination function performs an inner product of the layer weights and the layer inputs. A linear activation is usually used in the output layer to perform linear regression analysis (fitting the data linearly). A sigmoidal function is usually used in hidden layers so that the layer output is differentiable. The error function determines the error between desired and actual outputs per layer and updates the weights such that during the next iteration the final error would reduce.



$y = x$ $\qquad$ $y = \frac{e^{-x} - e^{x}}{e^{-x} + e^{x}}$ $\qquad$ $y = \frac{1}{1 + e^{-x}}$

Figure 1-22: Various Transfer Functions

Linear classifiers don't always provide an acceptable solution. The best example is the XOR problem where classification using a single line (regression) is not possible and that is where non-linear classifiers enter the scenario.

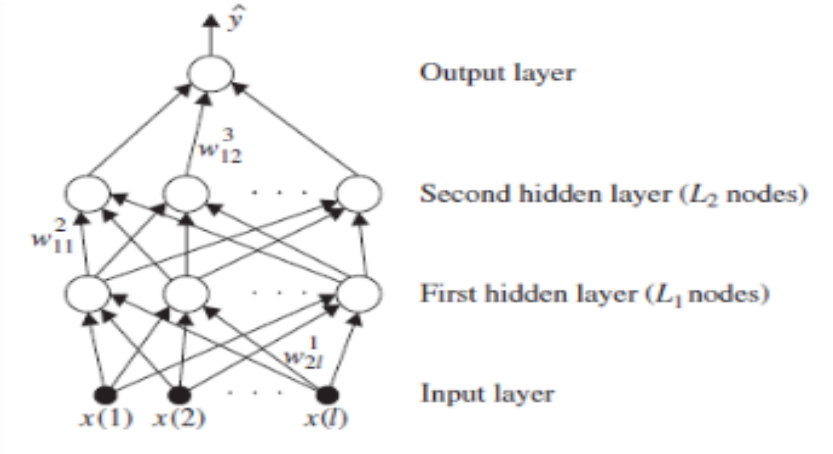


Figure 1-23: A Three layer network

[Image taken from 27]

1.7.6.1. Static Neural Networks

Feed-Forward Architecture

Static neural networks are unidirectional (feed-forward) and contain a set of neurons which process the data. It has three kinds of layers – input, output and hidden. Input layers are directly connected to the inputs and contain a set of processing elements which modify the inputs depending on the weights. Output layers do similar processing and produce the final outputs. Usually an input and output layer is not enough to attain a good analysis or classification. Hence a minimum of one hidden layer is included which

does further processing because of the additional processing elements in the layer. This configuration is also referred to as a Multi-Layered Perceptron.

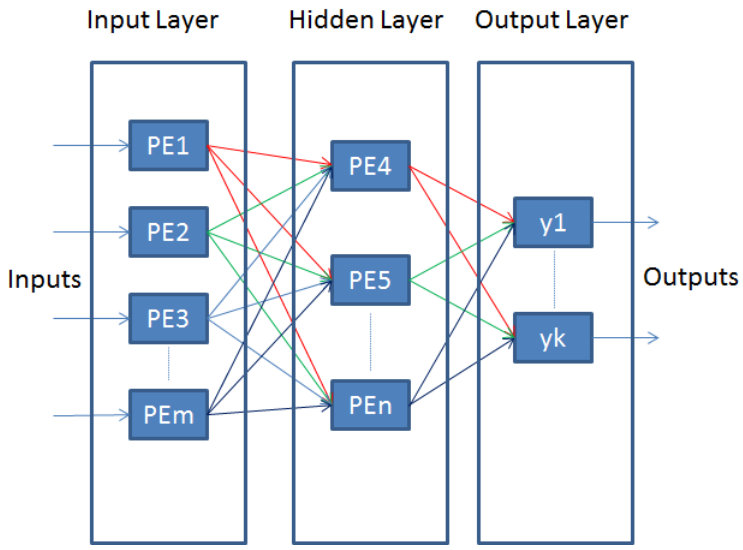


Figure 1-24: Feed-forward Network (Multi-Layered Perceptron)

If $y_k(t)$ is the output, w denotes the weights of different nodes, n is the total number of nodes per layer, i denotes the layers and $x_k(t)$ is the current input, the output expression can be given as

$$y_k(t) = f\left\{\sum_{k=1}^{n} w_{ik}(t)x_k(t)\right\}$$

Feedback Architecture

An iterative procedure named back-propagation is used to reduce the mean square error by recursively updating the weights and bias, which is done in the feedback loop. The feedback loop updates the weights and new outputs are obtained from every layer. This process first trains the network using the input data multiplied by flexible weights and calculates the error by comparing the computed output with the desired output. If the

error is larger than desired, a feedback is performed and the process is run again with updated weights. This iterative process continues until the error lowers to a desired value.

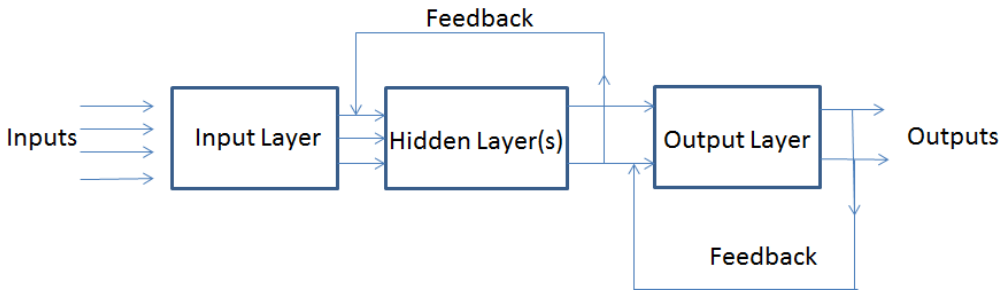


Figure 1-25: Back-Propagation algorithm

The weight update is governed by the following formula, whereαrefers to the learning rate and$\frac{\delta e}{\delta w_{ik}}$ is the change in error with respect to the weights. Choosing the optimum learning rate that in turn produces optimum weights is the ultimate goal of the algorithm. In momentum learning the weights also depend on their previous state. If the search direction suddenly goes flat, a push down-hill is still given owing to the previous state. μ refers to the momentum constant which is usually 0.5.

$$w_{ij}(n+1) = w_{ij}(n) + \alpha\nabla J(w_{ij}) \qquad \text{Gradient Descent algorithm}$$

$$w_{ij}(n+1) = w_{ij}(n) + \alpha\nabla J(w_{ij}) + \mu(w_{ij}(n) - w_{ij}(n-1)) \qquad \text{Momentum Learning}$$

Radial Basis Functions (RBF)

RBFs measure the distance between the input vectors and the reference vectors in contrast to the feed-forward networks, which calculate the summation of dot product of the weights and inputs at each phase. By using RBFs the inputs are transformed using a radial basis function using a distance measure. The distance measure can be Euclidean or

Mahalanobis depending on choice. There are two main configurations of RBF – exact and alternate. In the exact version, the RBF produces a neural network which has zero training error. On the other hand, the alternate configuration, the RBF iteratively creates a neuron for every time instant and this process continues until the error is acceptable.

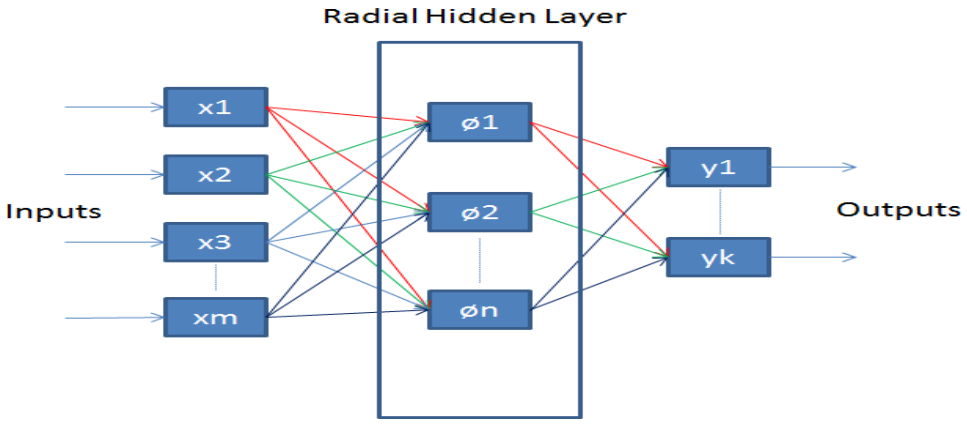


Figure 1-26: Radial Basis Function Network

$$y_k(t) = f\left\{\sum_{k=1}^{n} w_{ik}(t)\text{ø}k(t)\right\}$$

The activation function $\text{ø}(t)$is usually Gaussian and is given by

$$\text{ø}(t) = \exp\left[\left(\frac{-1}{2}\right)(x - xi)' \sum^{-1} (x - xi)\right]$$

1.7.6.2. Dynamic Neural Networks

Dynamic neural networks stand as good classifiers and predictors owing to their time variant feature [27], which enables them to use outputs at hidden/output layer(s) as feedback(s) to the system in-turn aiding in better performance, a phenomenon not inherent in static networks. They can be represented by the following model

$$y_i(t) = f\left\{\sum_{i=1}^{n}\sum_{k=1}^{d} w_{ik}(t)y_i(t-k)\right\}$$

where k is the delay operator

Placement of the delays depends on the model types which can be focused TDNNs, NARX or LRNs discussed below.

Time Delay Neural Networks (TDNN)

For regular TDNNs the dynamics (represented by the delay operator) occur only at the inputs.

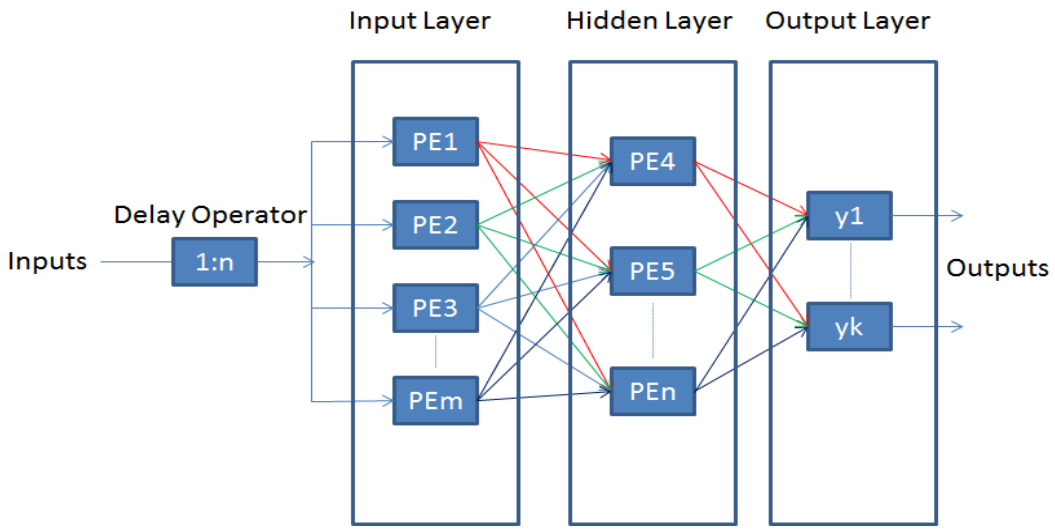


Figure 1-27: Time Delay Neural Network

NARX Networks

The next level to regular TDNNs are the NARX networks which are similar to the regular TDNNs but they additionally cover dynamics in the feedback loop.

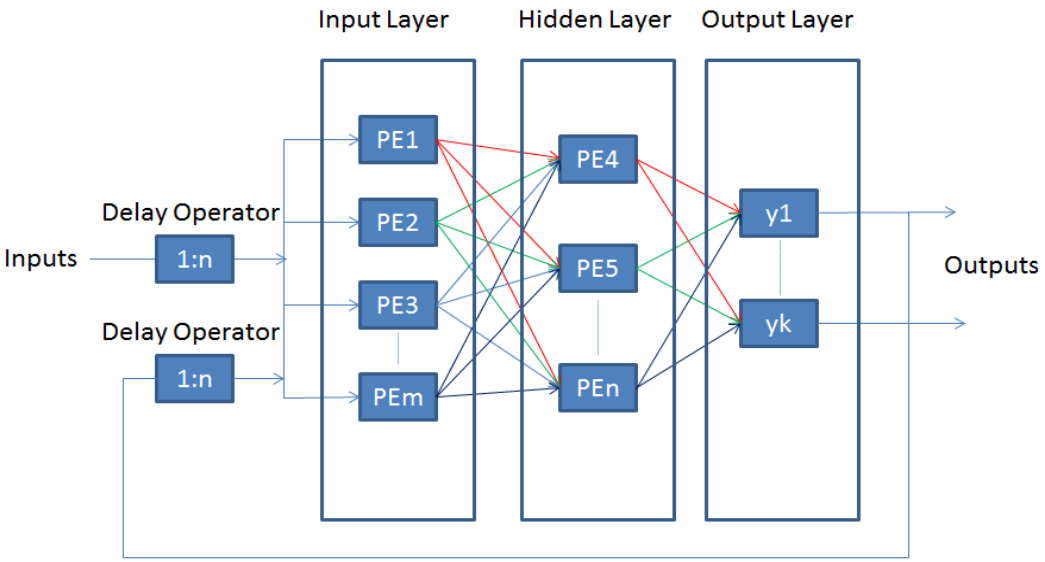


Figure 1-28: NARX Network

Layer Recurrent Networks (LRN)

LRNs are the most complex types of dynamic networks which analyze dynamics in every layer of the network.

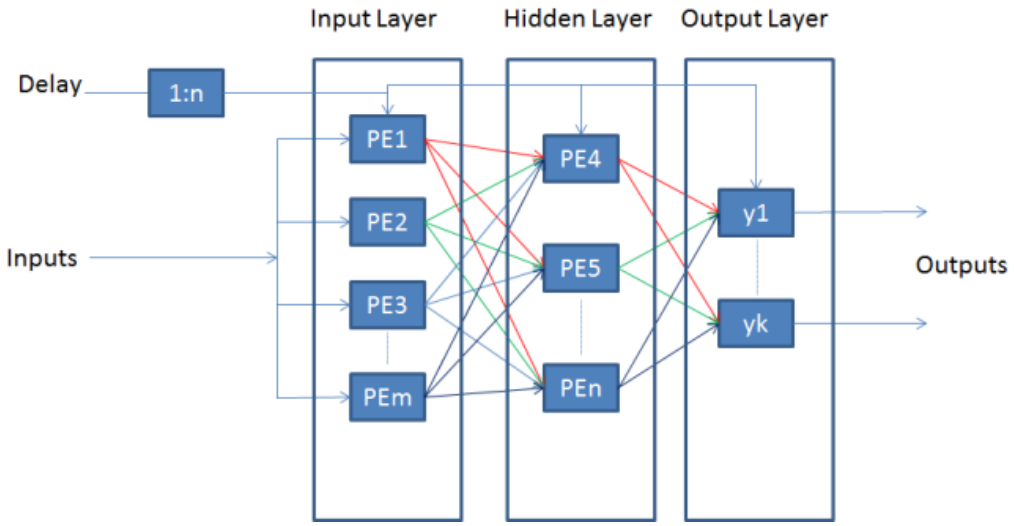


Figure 1-29: Layer Recurrent Network

1.7.6.3. Prioritizing the Models

Once all the models are at hand, the next step is to be able to guess which model is suitable for the dataset. If the system dynamics are simple enough with a low model order, transfer function models should suffice. However, if the model complexity increases, transfer function models do not give acceptable solutions and the need to switch over to more robust linear models (polynomial or state space) arises. If linear models do not give good analysis, non-linear models and ultimately neural networks are the next best bets. Table B-1 depicts all the models discussed above with their respective pros and cons and model representations.

2. METHODS

2.1 Multi-body and Finite Element Analysis

Simplified models of the tibio-femoral joint were developed [22] and multi-body (MB) and finite element (FE) analysis were performed on the model. For this process, a $20 * 20$ mesh [22] was generated with a $1\ mm * 1\ mm$ cross-section, and each element in the mesh is connected to the bone of the surrogate model [22]. Five different forces were applied to the dynamic condyle in five different 2D motion paths (top-views of the paths are shown in Figure: 2-1 (b)). A contact was introduced between the condyle and the tibia cartilage and 3D reaction forces were then found in X, Y and Z directions for these elements, and these served as the inputs to the neural network models. Finite Element analysis was performed to generate the VON-MISES stresses at these mesh

elements which are used as outputs in the neural network model. The main reason for the application of different levels of forces was to have enough training samples which are a vital prerequisite for the neural networks. A detailed description of the surrogate models developed is given in [Yunkai et. al 2013] [22].

The process of musculoskeletal modeling is started with a single "big" cell that has a fixed joint attached to it and multi-body analysis (or rigid-body simulation) is performed to extract the 3D reaction forces at each joint. For multi-body analysis, the cell doesn't have to be split and can be used as a whole. However multi-body analysis does not provide stress or strain information, and hence finite element analysis is used. Since finite element analysis does not produce acceptable results when used on a single “element”, meshes are designed which split the cell into finite elements. An optimum mesh needs to be checked for convergence. The performance (maximum displacement) of two meshes, one with 400 and the other with 1600 elements, was compared. A variance in performance up to 10% is acceptable but if the variance is more than 10% the mesh needs to be refined further. Finally, the mesh with 400 elements turned out to be acceptable and was used for finite element analysis. Once the mesh was selected, the cell was split into 400 elements and rigid body simulations on these elements provided reaction forces at the fixed joints attached to these elements. Finite element analysis on these elements gave stress information. Finite element analysis can perform any operation that multi-body analysis does (reaction force information) in addition to extra information (stress, strain, etc.) which multi-body cannot, but at the cost of increased computational time and

complexity. Hence neural networks were used to predict the stress information, given the reaction forces from multi-body analysis.

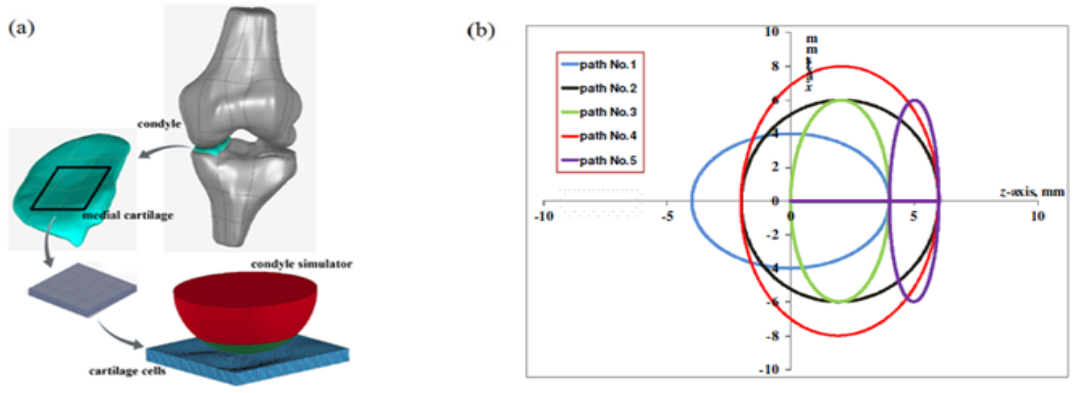


Figure 2-1: (a) Model Geometries (b) Top view of the motion paths used

Images taken from [Yunkai, Guess et. al 2013] [22]

2.2 Neural Network Modeling

Neural networks were trained with reaction forces from multi-body as inputs and VON-MISES stress from finite-element as outputs where the input and output dimensionality was 1200 and 400 respectively. 21 out of 25 MB-FE simulation datasets were used for NN training and the remaining were left for testing to test the generalization capabilities of the network. Out of the 21 simulations left aside for training, 30% was used for validation to avoid over-training. Each NN simulation was run 20 times and the overall simulation time was found by averaging the simulation times of 10th to 20th runs. The

first 9 runs were discarded to ignore the CPU startup times which are usually high during the first few runs.

The main challenge in neural network modeling is to choose the correct network parameters like hidden layer size and tap delay lengths which are found using trial-and-error approach. Once the optimum topology is estimated, various, training, and performance functions are studied and best suitable functions for the scenario are obtained. . The networks are run with different parameters and the topology which gives the best performance is chosen.

2.2.1 Choosing the Network Topology

The LEVENBERG-MARQUADT training method, which uses the *Gauss-Newton* for Hessian approximation [27], is one of the widely used neural network training algorithm primarily because of its ability of faster convergence when compared to other algorithms. However since it requires more memory to run [2] as its gradient is proportional to the square of number of weights and this makes the algorithm unsuitable for huge datasets, it was not chosen as the training function for this dataset. To overcome the memory issue, the scaled conjugate method can be used as its gradient is proportional to number of weights and it also aids in fast convergence. Bayesian regularization training uses even lesser memory than scaled conjugate training but takes more number of iterations to converge [2]. From the results obtained in Table2-1, *scaled conjugate training* emerged to be the best performer, and hence is used as the default training method for all the networks in the forthcoming network configurations.

Table 2-1: Comparison of Training Functions

Training Type	HN	Train R	Validation R	Test R	All R	Best Validation MSE	Epochs	Time (min)
Scaled Conjugate	60	0.9175	0.83331	0.8269	0.8921	0.074841	884	81
Cyclic	60	0.9328	0.82981	0.8263	0.8969	24.1232	4084	642
One-step secant back-propagation	60	0.1487	0.79278	0.7565	0.1416	0.081509	941	83
Resilient	60	0.9189	0.76887	0.6899	0.8615	0.1238	95	5

Of the transfer functions mentioned in Section 1.6.4, logarithmic sigmoidal proved to be a better transfer function over the regular tangent sigmoidal. This check was carried over a set of networks with different hidden nodes mentioned in Table 2-2.

Table 2-2: Comparison of Transfer Functions

Network Type	Transfer Function	Hidden nodes	Test Correlation Coefficient	Best Validation MSE	Epochs
Function Fitting Network	tansig-purelin	30	0.59412	0.36352	981
Function Fitting Network	tansig-purelin	30	0.75482	0.1828	1387
Feed-forward Network	tansig-purelin	20	0.67617	0.033624	1673
Function Fitting Network	logsig-purelin	20	0.84907	0.12283	834
Feed-forward Network	logsig-purelin	20	0.86287	0.078679	520
Feed-forward Network	logsig-purelin	30	0.87911	0.064857	561

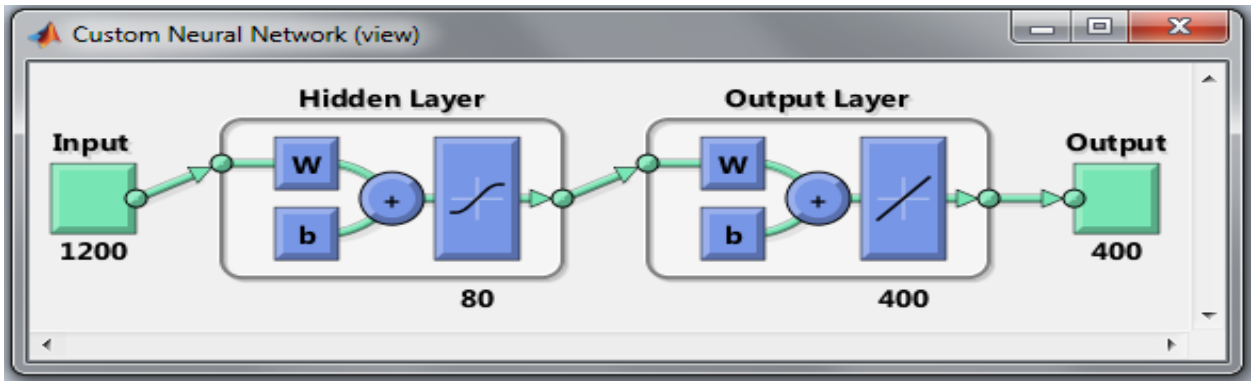


Figure 2-2: Static Neural Network with 80 Hidden Nodes

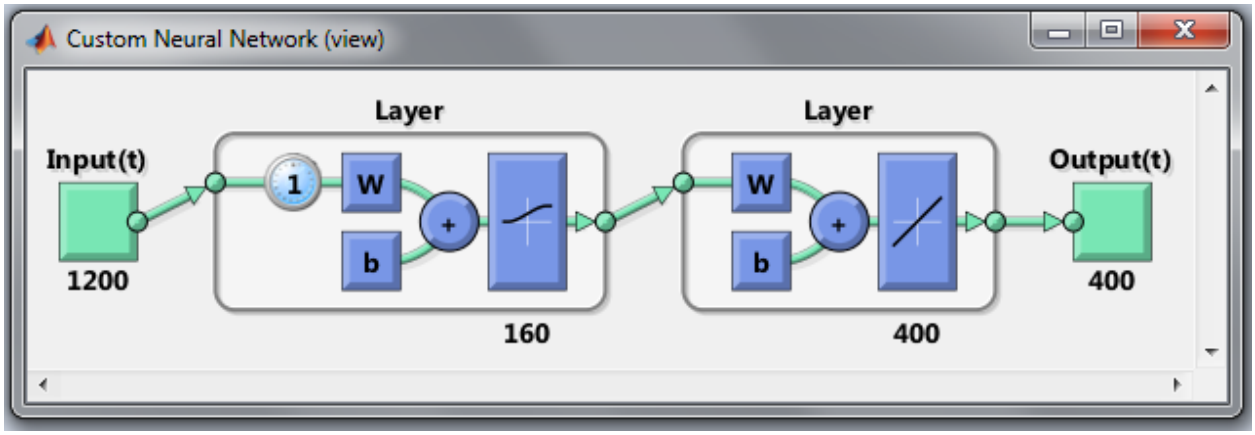


Figure 2-3: Dynamic Neural Network with 160 Hidden Nodes and Tap Delay Length 1

The below parameters were used for NN training -

- Train, Validation, Test ratio = {0.58, 0.26, 0.16}
- Training Function = Scaled Conjugate Gradient with Validation based early stopping
- Learning function = Gradient Descent with momentum learning
- Performance Function = Mean Squared Error with Regularization (regularization ratio = 0.9)

2.3 Training Methods

Once the network configuration is chosen (hidden nodes, network type etc.) the next step is to train the neural network. To do this a training style should be chosen. There are two possible styles of trainings– Batch and Incremental. In batch training, weights are updated once all the training data are presented and this process continues in every epoch. Because of their quickness and the ability to use the whole dataset for every epoch to determine changes, batch training is generally considered more efficient and is used in most of the real-time scenarios. On the other hand, adaptive or incremental training processes one training data per iteration and updates the weights after every iteration. Incremental learning is used in cases when the data is not fixed and when network needs to be trained time to time with the updated data.

In addition to the individual network connections (static or dynamic), there are a few well-known configurations, which when used effectively, produce considerably better results than individual networks,

2.3.1 Committee

Grouping networks by averaging their outputs produce better results than the individual networks themselves. This happens due to possible cancellation of uncorrelated errors. This phenomenon of averaging the networks is called the averaged committee of networks. Since this process involves operations on correlated errors, the prime motive here is to make the errors *uncorrelated* by minimizing the inter-dependencies on the individual network errors.

The averaging can either be *regular* averaging or *weighted* averaging.

$$\text{Regular Committee} = (O_1 + O_2 + O_3 + \cdots + O_n) \,/\, n$$

$$\text{Weighted Committee} = \frac{(O_1 * acc_1 + O_2 * acc_2 + O_3 * acc_3 + \cdots + O_n * acc_n)}{(acc_1 + acc_2 + .. + acc_n)}$$

$$acc_n = \frac{1}{\text{validation error}_n}$$

where n is the number of individual networks under consideration and O_n is the individual network output for network n.

While forming committees, the usual practice is to average the outputs of best networks. However using this approach might not be favorable in practical scenarios where a set of networks, which may have the best outputs themselves, might not perform better as a committee. In such cases, a broader set of networks can be taken into consideration. Out of a set of n best individual networks, this approach would select the best committee out of 2^n possible combinations. This approach was applied for both static and dynamic networks, and results are shown in the results section.

2.3.2 Stacked Generalization

Stacked Generalization is a special configuration of networks where outputs of the best individual networks are fused together and are used as an input to a *new* neural network. The target output of this network would be the same as the desired output of the individual networks. This network is trained similar to the individual networks. The Stack generalization networks were trained with outputs of the best static and dynamic networks using *scaled conjugate training*.

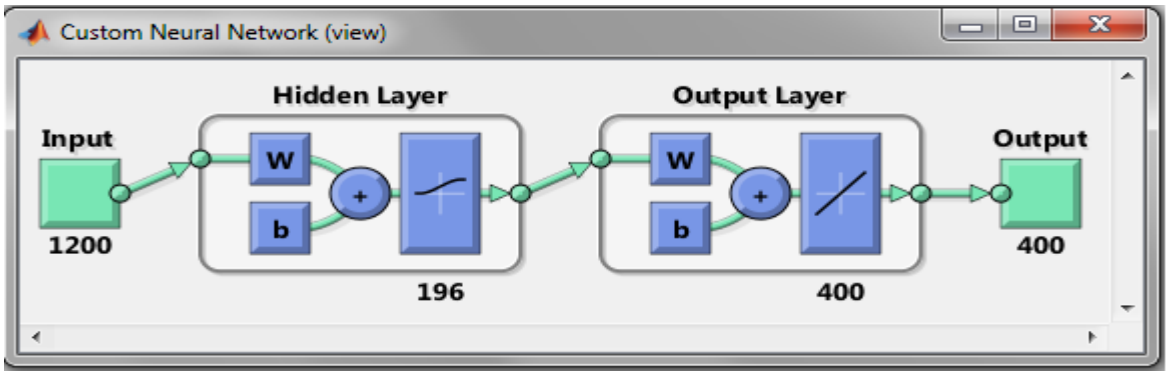


Figure 2-4: Stack Generalized Network with 196 Hidden Nodes

2.3.3 Negative Correlation Learning

The concept of averaged committees, as discussed in the previous section, is to find the individual networks with correlated errors and attempt to minimize the errors and hence achieve better results by averaging the outputs.. However having the best networks beforehand is a tedious process and requires looping through 2^n possible combinations (n is the number of best networks under consideration) as discussed previously. Also after going through this process, there is no guarantee that the errors would be uncorrelated.

Hence a new approach named *Negative Correlation learning* (NCL) was introduced by [17 Yong Liu and Xin Yao] where individual networks are trained such that they produce uncorrelated errors. These network outputs, when averaged, definitely produce better results than the individual networks would as their errors were trained to be uncorrelated. The error function to be minimized in this approach would be the following

$$E = \frac{1}{2}\sum_{i=1}^{n}[(y_i - d)^2 + \lambda * (y_i - y_{committee})^2]$$

$$y_{committee} = \frac{(\mathrm{y_1} * \mathrm{acc_1} + \mathrm{y_2} * \mathrm{acc_2} + \mathrm{y_3} * \mathrm{acc_3} + \cdots + \mathrm{y_n} * \mathrm{acc_n})}{(\mathrm{acc_1} + \mathrm{acc_2} + .. + \mathrm{acc_n})}$$

$$\mathrm{acc_n} = \frac{1}{\text{validation error}_\mathrm{n}}$$

where *n* is the number of individual networks being trained; y_i is the output of the individual network; acc is the accuracy rate of the network; *d* is the desired output; $y_{committee}$ is the averaged committee output of the individual networks and λ is the *penalty operator* [17, 18, 19].

The first term in this performance function is the regular mean squared error (MSE) and the second term is a *penalty function* [17, 18, 19]. When $\lambda = 0$, the error function becomes similar to the MSE. A test was performed to check for the optimum λ value where $0 < \lambda \leq 1$. Four sets of networks with similar topologies except for the varying λ value were trained for 1000 epochs and their behavior (committee train correlation coefficient and maximum residual correlation coefficient) was observed. According to the results obtained in the figure, $\lambda = 0.84$ proved to be the optimum penalty factor value for this dataset. For this value, the reduction rate of maximum residual correlation coefficient from the list of four individual networks is better than that of MSE performance function. Also the committee correlation coefficient is better. Hence

for all the NCL analysis, $\lambda = 0.84$ is the chosen value. The committee is formed by performing w*eighted* average of the individual network output.

For negative correlation learning, since the performance function contains the "committee average output" as one of its terms, incremental training is used. However since incremental learning introduces only 1 input per epoch, the total number of iterations was set to a high number. The actual number of epochs can be calculated by the following formula

$$Actual\ number\ of\ epochs = \frac{Total\ Number\ of\ iterations}{Number\ of\ inputs} \quad .$$

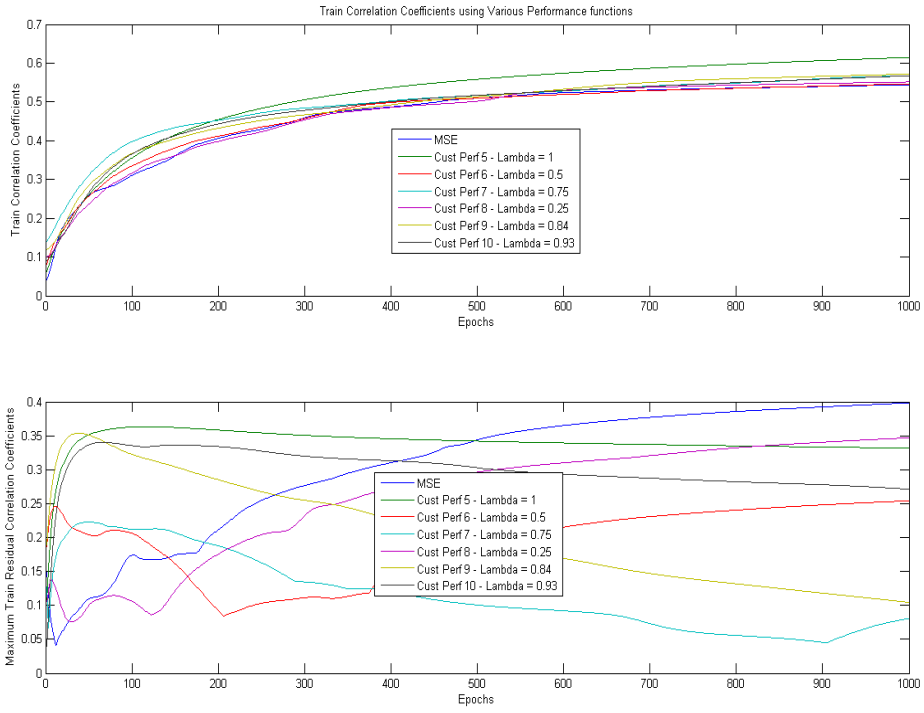


Figure 2-5: Comparison of various networks with different penalty factors

2.4 Error Analysis

Error correlation can be done in two ways - Choosing networks with seemingly uncorrelated errors and averaging their outputs, ultimately minimizing the error and training the individual networks such that their errors turn out uncorrelated once they are fully trained. Using these algorithms, different network outputs and hence different residues (difference between desired vs. actual outputs) are obtained. Out of these a set of networks with acceptably close validation or test errors are chosen and error analysis is performed on these networks. The following steps are followed to obtain the networks with the least correlation

a. Choose models with validation errors close enough (not more than 10% variance). Assume n such models are chosen.
b. Calculate the residues of model n vs. the desired output or ground truth. This step would result in having n different residues.
c. Create an n*n matrix where each entry is the correlation coefficient between residues of two models at a time.
d. From the above matrix, choose the entry with the smallest Correlation Coefficient. This signifies that these two models were trained and ultimately have the least error correlation, and when their outputs are averaged, they produce better results as their errors would have been reduced.

Example: Assume a neural network is trained to predict the sequence of squares. The desired output would be the sequence “1 4 9 16 25 36 49 64 …..” Four such networks were trained with produced the following outputs.

Output1 = 1.5000 6.0000 13.5000 24.0000 37.5000 54.0000 73.5000 96.0000

Output 2 = 0.6000 2.4000 5.4000 9.6000 15.0000 21.6000 29.4000 38.4000

Output 3 = 1.9000 7.6000 17.1000 30.4000 47.5000 68.4000 93.1000 121.6000

Output 4 = 0.3000 1.2000 2.7000 4.8000 7.5000 10.8000 14.7000 19.2000

From the first look its evident that network 2 produced the closest results to the ground truth with the smallest mean squared error. Now if residual analysis is performed on the above the following residuals are obtained

Residue1 = 0.5000 2.0000 4.5000 8.0000 12.5000 18.0000 24.5000 32.0000

Residue2 = -0.4000 -1.6000 -3.6000 -6.4000 -10.0000 -14.4000 -19.6000 -25.6000

Residue3= 0.9000 3.6000 8.1000 14.4000 22.5000 32.4000 44.1000 57.6000

Residue4 = -0.7000 -2.8000 -6.3000 -11.2000 -17.5000 -25.2000 -34.3000 -44.8000

From the figure residues {3, 4} and residues {1, 2} have the least correlations and hence when the outputs of {3, 4} and {1, 2} are averaged, better outputs are obtained. Step 3 in the figure shows the outputs of 1 and 2 averaged and 3 and 4 averaged.

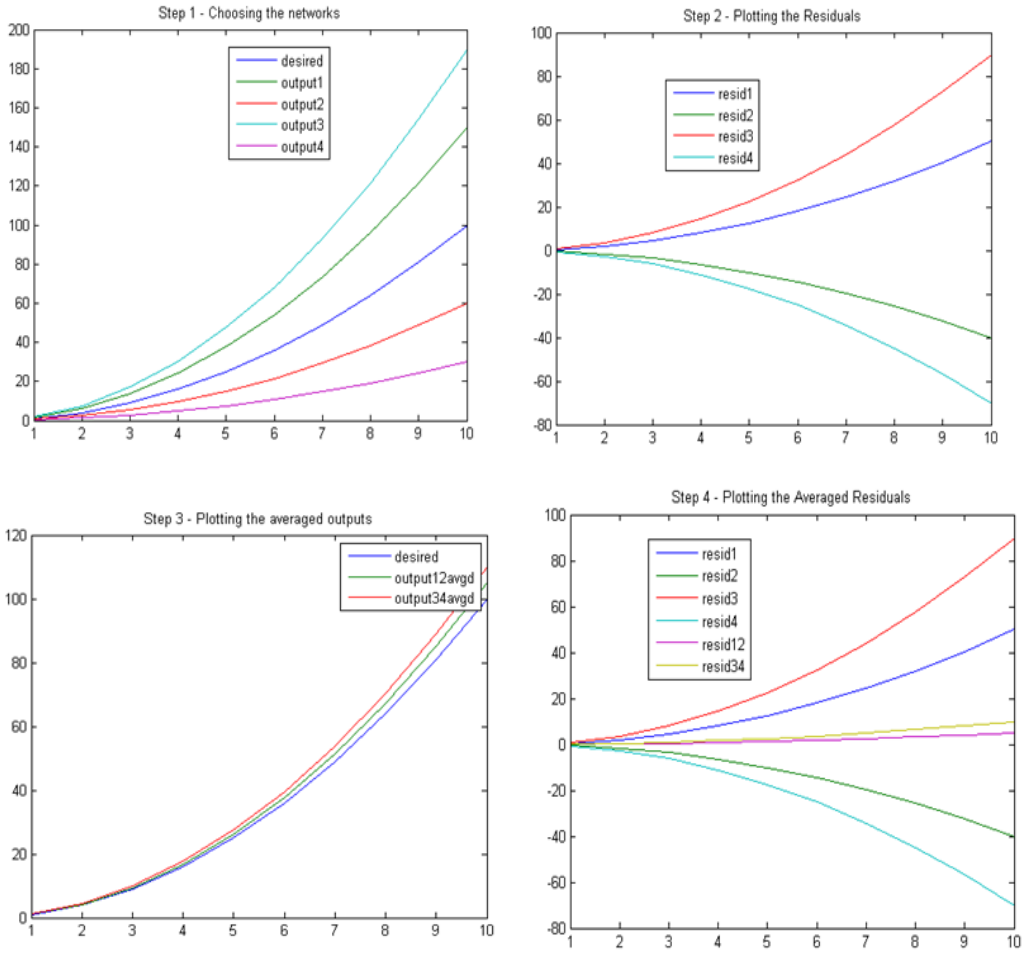


Figure 2-6: Steps in Error Analysis

3. RESULTS

3.1 Individual Neural Network Connections

Table 3-1: Static Neural Networks – Individual Results

S. No	Hidden nodes	Train R	Test R	Train MSE	Test MSE
1	80	0.9184	0.8898	0.1703	0.1919
2	100	0.9158	0.8877	0.173	0.1936
3	160	0.9098	0.8822	0.1789	0.1977
4	110	0.9029	0.8783	0.1851	0.2011
5	150	0.9106	0.8704	0.1781	0.2081
6	60	0.9135	0.8696	0.175	0.2093
7	50	0.8925	0.8687	0.1941	0.2081
8	40	0.9059	0.8663	0.1822	0.2118
9	170	0.8964	0.8647	0.191	0.2156
10	90	0.8838	0.86	0.2017	0.2147
11	180	0.8859	0.8538	0.2	0.2211

Network type = Function Fitting Neural Network

Train, Validation, Test ratio = {0.7, 0.3, 0}

Training Function = Scaled Conjugate Gradient training

Performance Function = Regularized Mean Squared Error

Transfer function – {Logarithmic Sigmoidal for Hidden Layer and Linear for Output Layer}

Max Fail Iterations = 51, Epochs = 1500

Table 3-2: Dynamic Neural Networks – Individual Results

S. No	Hidden nodes	Tap Delay Length	Train R	Test R	Train MSE	Test MSE
1	160	1	0.9402	0.8772	0.1463	0.2035
2	300	1	0.9645	0.8814	0.1134	0.2085
3	240	5	0.9492	0.8717	0.1352	0.2086
4	200	1	0.9544	0.8758	0.1282	0.2092
5	280	1	0.9661	0.8827	0.111	0.2097
6	230	5	0.954	0.8701	0.1289	0.2111
7	250	5	0.948	0.8673	0.1368	0.2117
8	210	5	0.9449	0.8674	0.1406	0.2141
9	220	1	0.9567	0.872	0.1251	0.2157
10	240	1	0.9583	0.8722	0.1228	0.2166
11	282	1	0.9586	0.8914	0.1223	0.1980

Network type = Focused Time Delay Neural Network

Train, Validation, Test ratio = {0.7, 0.3, 0}

Training Function = Scaled Conjugate Gradient training

Performance Function = Regularized Mean Squared Error

Transfer function = {Logarithmic Sigmoidal for hidden layer, Linear for output layer}

Max Fail Iterations = 20,

Epochs = 1500

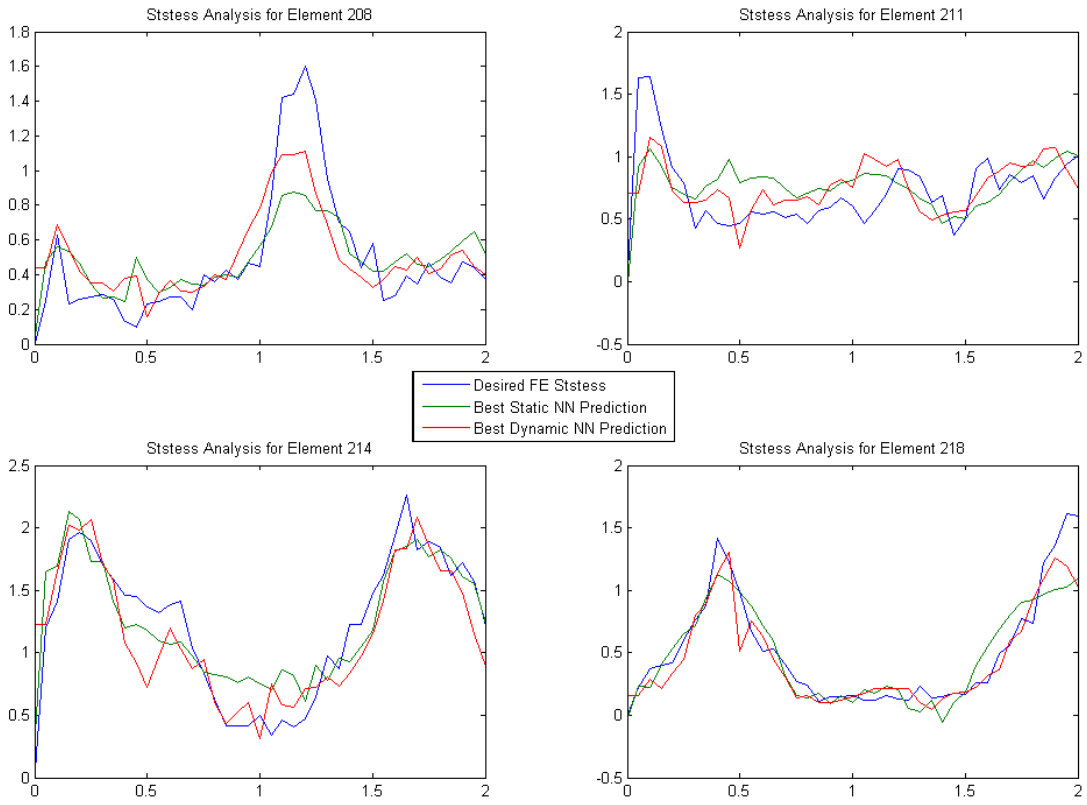


Figure 3-1 : NN stress analysis results

3.2 Weighted average committee of Neural Networks

Table 3-3: Static Neural Networks – Committee Results

S. No	Hidden nodes	Train R	Committee Train R	Test R	Committee Test R
1	80	0.9184	0.9352	0.8898	0.9027
2	100	0.9158		0.8877	
3	160	0.9098		0.8822	

Table 3-4: Dynamic Neural Networks – Committee Results

S. No	Hidden Nodes	Tap Delay Length	Train R	Committee Train R	Test R	Committee Test R
1	300	1	0.9645	0.9673	0.8814	0.9139
2	280	1	0.9661		0.8827	
3	160	1	0.9402		0.8772	
4	230	5	0.954		0.8701	
5	240	5	0.9492		0.8717	

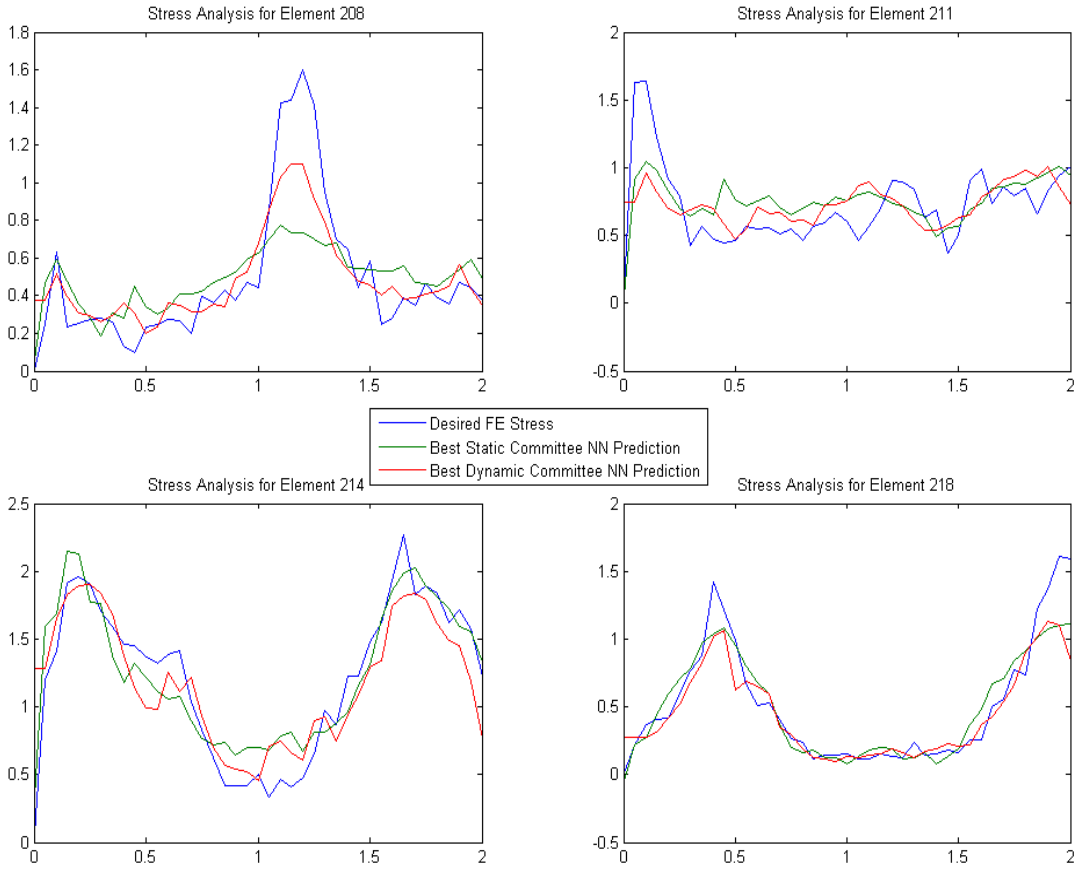


Figure 3-2 : Committee of NNs stress analysis results

3.3 Stacked Generalization

Table 3-5: Stacked Generalization for Static NN Results

S. No	Hidden Nodes	Best Validation MSE	Train MSE	Test MSE	Train R	Test R	Epochs
1	16	0.0297	0.1535	0.1842	0.9341	0.8984	807
2	46	0.0276	0.1324	0.164	0.9517	0.9207	812
3	66	0.0245	0.1291	0.1603	0.9539	0.9239	783
4	86	0.025	0.1277	0.16	0.955	0.9242	616
5	106	0.0313	0.1388	0.1675	0.947	0.9168	481
6	136	0.0579	0.1603	0.1622	0.9306	0.9225	575
7	156	0.0259	0.1277	0.1607	0.9551	0.9237	460
8	176	0.0336	0.1415	0.1657	0.9449	0.9185	355
9	196	0.0389	0.1424	0.1569	0.9447	0.9272	455
10	206	0.021	0.1179	0.1627	0.9618	0.922	499

Table 3-6: Stacked Generalization for Dynamic NN Results

Stacked Generalization for Dynamic NN							
S. No	**Hidden Nodes**	**Best Validation MSE**	**Train MSE**	**Test MSE**	**Train R**	**Test R**	**Epochs**
1	106	0.0296	0.159	0.1962	0.929	0.8945	944
2	126	0.0152	0.1053	0.1986	0.9696	0.892	910
3	166	0.0165	0.1012	0.2277	0.972	0.8693	995
4	186	0.0168	0.1041	0.217	0.9702	0.8747	850
5	206	0.0161	0.102	0.2207	0.9715	0.8745	875

Network used - Function Fitting network.

Data division - 70% for training and 30% for testing

Training function – Scaled Conjugate Gradient Training

Performance function - Mean Squared Error

Transfer function – {Logarithmic Sigmoidal for Hidden Layer and Linear for Output Layer}

Early stopping validation checks – 26

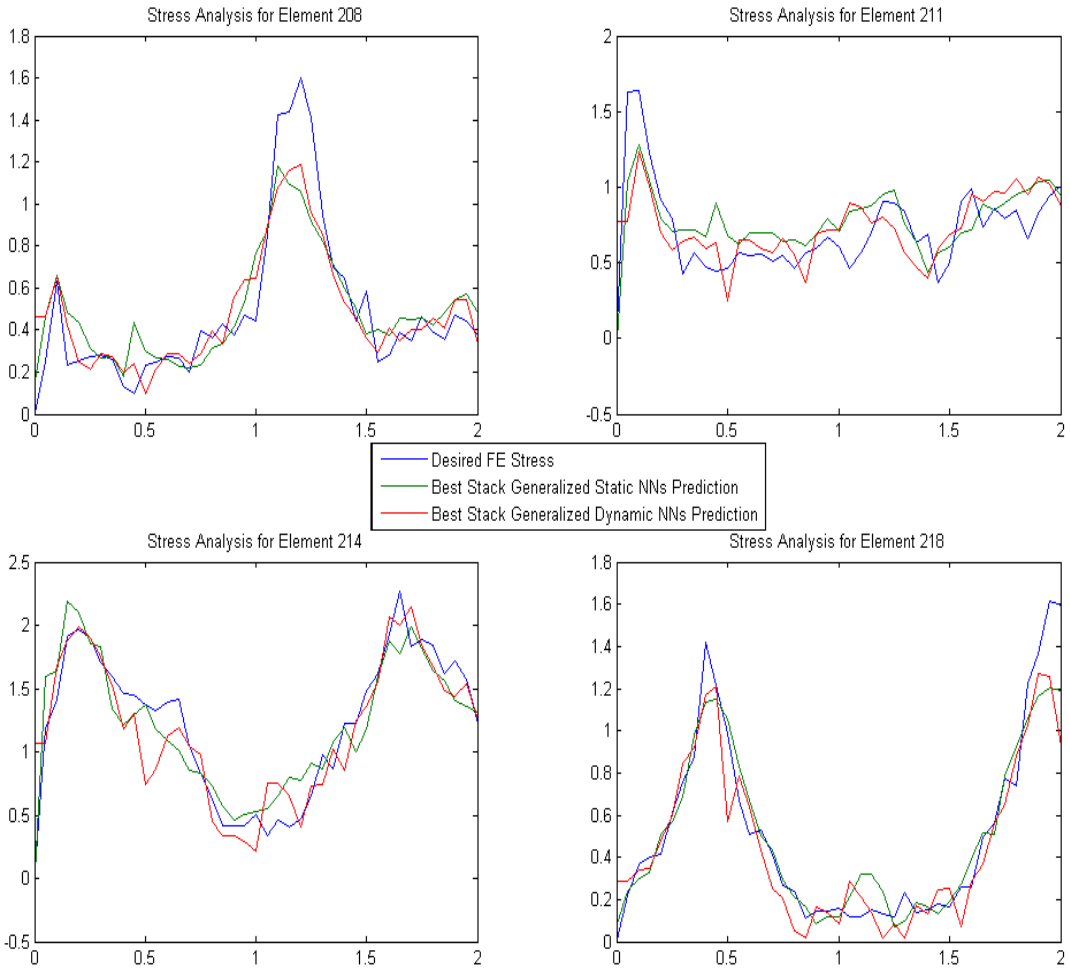


Figure 3-3 : Stacked generalization stress analysis results

3.4 Negative Correlation Learning for static networks

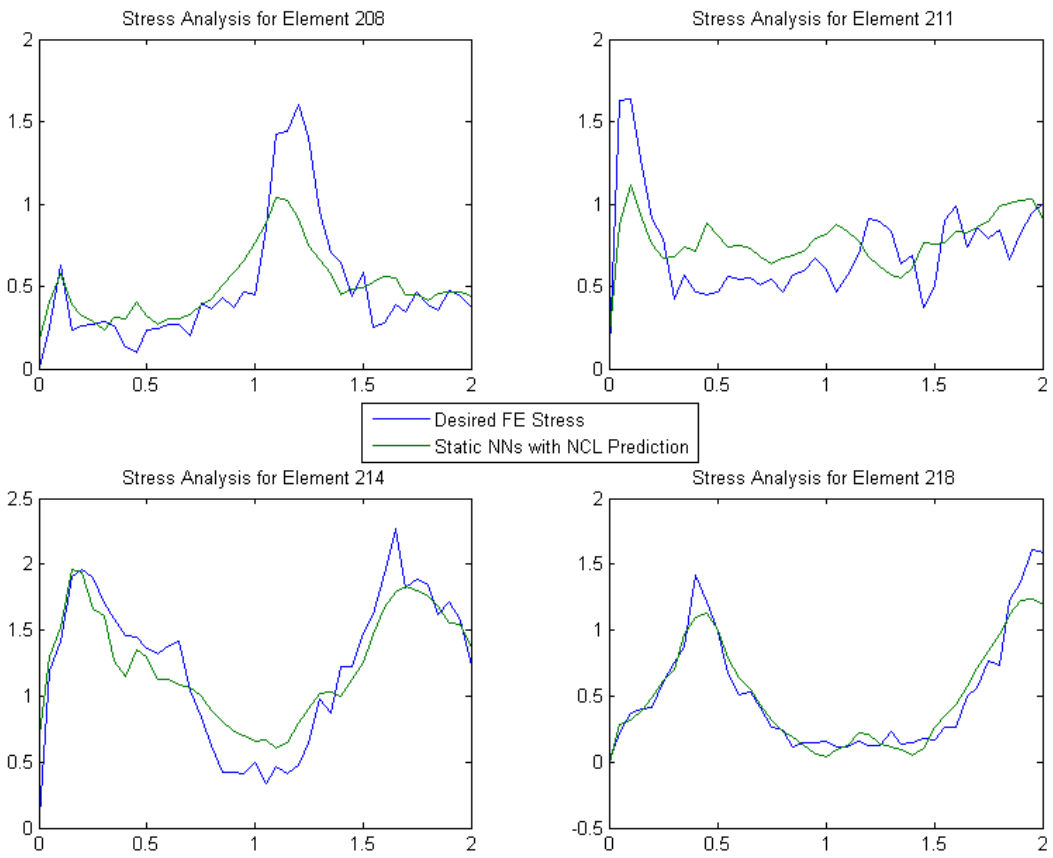


Figure 3-4 : NNs with NCL stress analysis results

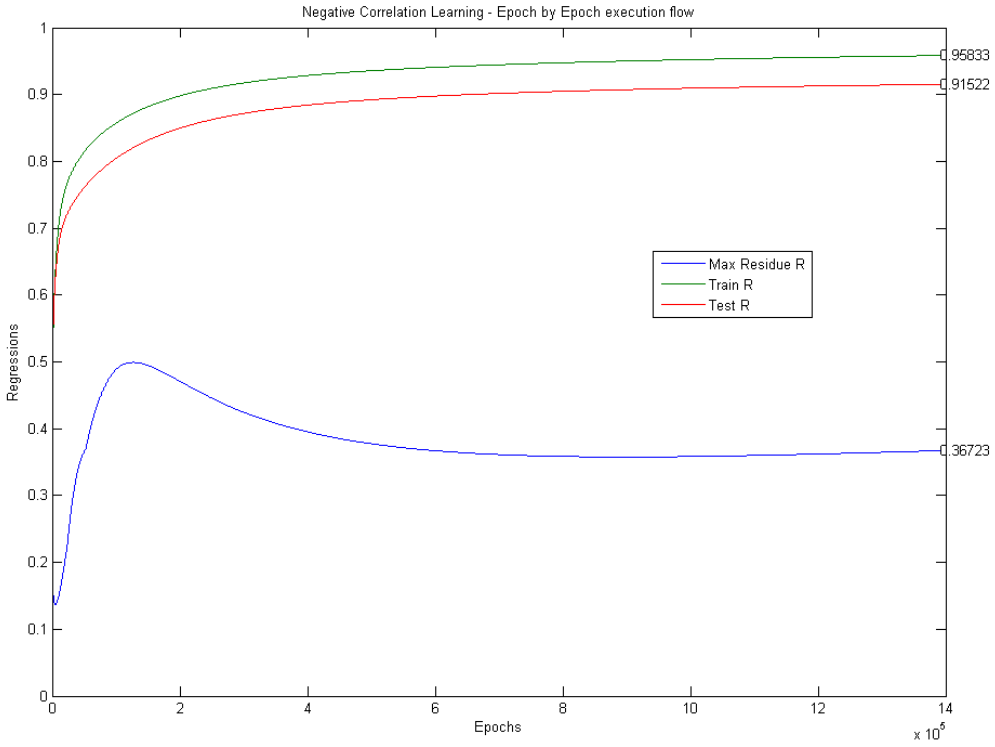


Figure 3-5 : Negative Correlation Learning – Epoch by Epoch execution flow

3.5 Best Configurations

Table 3-7: Best Configurations

Connection Type	**Hidden Nodes**	**Tap Delay length**	**Test MSE**	**Test R**	**Minimum Residual Correlation**	**Epochs**
Dynamic NN Individual	282	1	0.1980	0.8914	NA	1500
Dynamic NN Committee - Weighted Average	{300, 280, 160, 230, 240}	{1, 1, 1, 5, 5}	0.1708	0.9139	0.2853	NA
Stacked Generalization – Dynamic NNs	106	NA	0.1962	0.8945	NA	1064
Static NN Individual	80	NA	0.1919	0.8898	NA	853
Static NN Committee - Weighted Average	{80, 100, 160}	NA	0.1802	0.9027	0.6882	NA
Stacked Generalization – Static NNs	196	NA	0.1569	0.9272	NA	408
Negative Correlation Learning – Static NNs	{80, 100, 140, 180}	NA	0.1686	0.9152	0.1383	1161

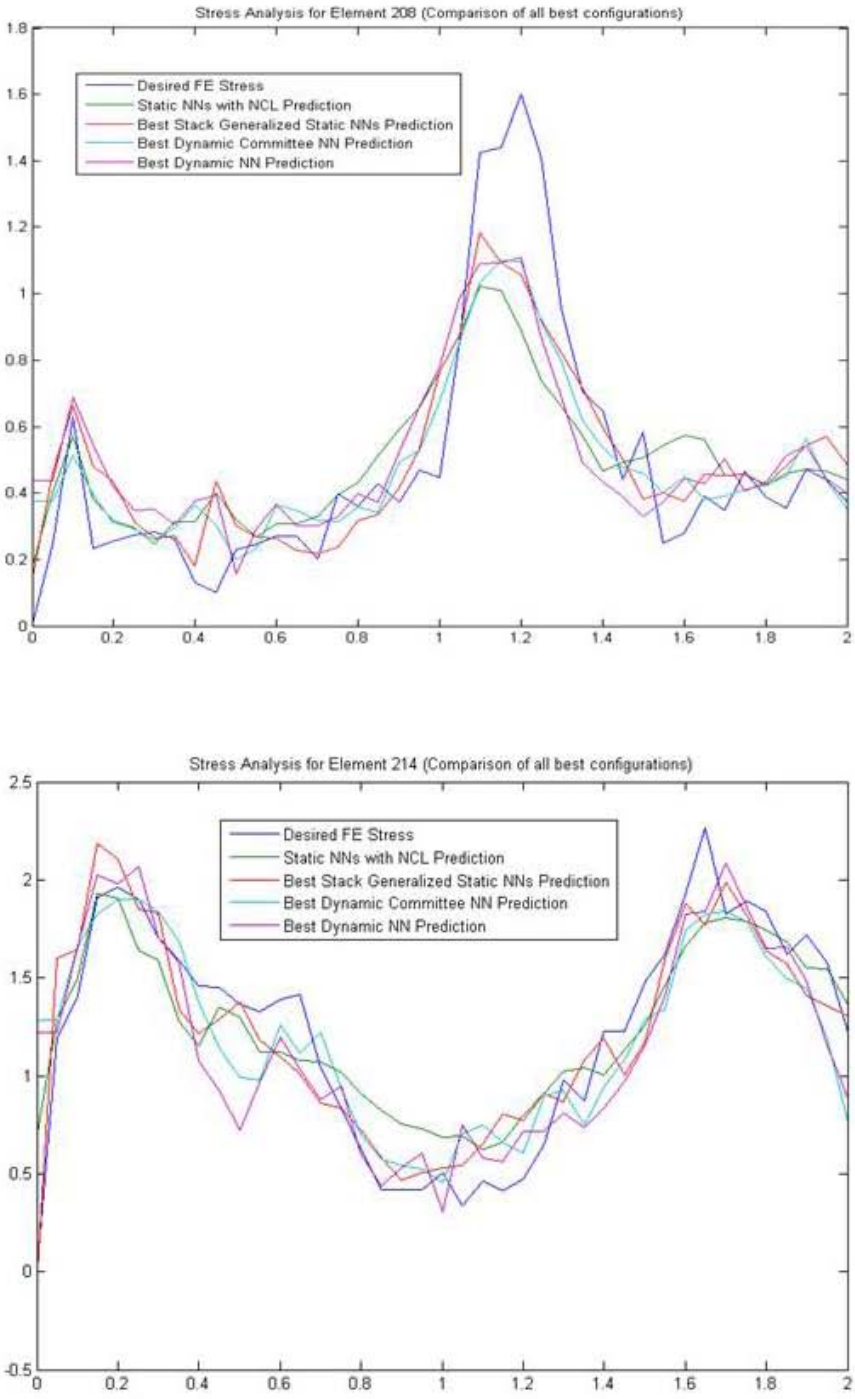


Figure 3-6 : Comparison of Best networks

4. CONCLUSION

Linear models were not used for this study considering their ineffectiveness in approximation and prediction tasks when compared to non-linear models. Various neural network configurations were used to analyze the musculoskeletal knee model. The best individual static network was a function fitting network with 80 hidden nodes (other specifications are mentioned in Table 3.1) and the best individual dynamic network was a focused time delay neural network with 282 hidden nodes and tap delay length of 1 (other specifications are mentioned in Table 3.2). Committee configurations with weighed average were then studied and the best results obtained for static committee (Table 3.3) was a test regression of 0.9027 and for dynamic committee (Table 3.4) was a test regression of 0.9139 which clearly shows an improvement and advantage of cancelling the correlated errors. Then stacked generalization was performed where a static neural network was trained with the outputs of the best individual networks. When used over static components (Table 3.5), this process gave a test regression of 0.9272 for a function fitting network with 196 hidden nodes which is the best result achieved. For dynamic networks (Table 3.6), stacked generalization didn't improve well over the committees. Finally negative correlation learning was implemented over a set of static committees which produced a test regression of 0.9152 and these results (Table 3.9) make NCL stand in the 2nd place right after the stacked generalized network which produced the best test regression of 0.9272 (Table 3.9). This confirms the theory that designing networks

with uncorrelated errors produce better results than cancelling out the correlated errors. This study also confirms theoretical concepts that special neural network configurations like committee, stacked generalization and negative correlation learning provide considerably better results when compared to individual networks themselves. Using neural networks, the simulation time was reduced nearly 40,000 times when compared to the finite-element models.

5. FUTURE WORK

More on regularized mean squared error

The performance function used throughout this research (except for negative correlation learning) is the regularized mean squared error which not only reduces the error between the actual and desired outputs but also includes a penalty function which reduces the network weights and ultimately results in faster convergence. The performance ratio λ determines the amount of weights to be reduced per iteration.

$$MSE\ Regularized = (\lambda) * [\frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2\ ] + (1-\lambda) * [\frac{1}{n}\sum_{i=1}^{n}(w_i)^2]$$

The default value used throughout the research is $\lambda = 0.9$. This value implies a 90% weightage of the mean squared error and 10% weightage of the mean squared weights per iteration. However the regularization term should be looked upon more closely by varying the performance ratio which would result in a flexible weightage of the amount of weights to be reduced per iteration considering the point that a very low λ value will choose networks smaller than the optimum [Principe et.al 1999]. For instance $\lambda =$

0.5 can be implemented which imposes equal weightage on the error as well as the weights.

Reconfirming the authenticity of a model

A neural network model (individual or an ensemble as discussed previously) needs to be trained multiple number of times and its test correlation coefficient needs to be noted after every training. With these results, an analysis has to be made on the variance of these results. If the R values are in the same ball park (with acceptable variance) the model is acceptable as the results it produced are trustworthy over several runs. However if the variance is too much, this would mean that the particular network topology under consideration is unstable and hence not reliable.

APPENDIX

A. Various linear and non-linear modeling techniques

Table A-1: Various linear and non-linear modeling techniques

Priority (low to high)	**Model**	**Model representation**	**Pros**	**Cons**
1	Transfer function	$G(s) = \frac{Y(s)}{U(s)}$ $= \frac{s^n + s^{n-1}\mathrm{b1} + s^{n-2}\mathrm{b2} + \cdots}{s^n + s^{n-1}\mathrm{a1} + s^{n-2}\mathrm{a2} + \cdots}$	For simple systems.	Not suitable for highly complex systems owing to the inability to factorize
2	Auto Regressive (AR)	$y(t) = \sum_{q=1}^{n} y(t-i)A(q) + e(t)$	Outputs depend only on previous outputs	Inputs are not used in modeling

Priority (low to high)	Model	Model representation	Pros	Cons
4	ARX	$y(t)A(q) = \sum_{q=1}^{n} u(t-i)B(q) + e(t)$	Provides a unique solution using linear regression analysis. Preferable when model order is high.	Disturbances are modeled as part of system dynamics. Dynamic disturbances are ignored
5	ARMAX	$y(t)A(q) = \sum_{q=1}^{n} u(t-i)B(q) + C(q)e(t)$	More flexibility in handling the disturbances (The "C" term). Useful when disturbances enter early in the process (Ex: inputs)	Lesser flexibility in disturbance modeling compared to Box Jenkins model.

Priority (low to high)	Model	Model representation	Pros	Cons
6	Box Jenkins	$y(t) = \sum_{i=1}^{n} \mathrm{u(t-i)}\frac{Bi(q)}{Fi(q)} + \mathrm{e(t)}\frac{Ci(q)}{Di(q)}$	Completely distinct System dynamics and disturbance models. Useful when disturbances enter late in the process (inputs)	Requires lot of data.
7	State Space	$x(t+1) = Fx(t) + Gu(t)$ $y(t) = Hx(t) + Du(t)$	Provides better analysis of the system than polynomial models. Only the model order needs to be user-defined.	For high order complex models, State space modeling is expensive and takes lot of memory.

Priority (low to high)	Model	Model representation	Pros	Cons
9	Hammerstein – Wiener	$y(t) = g(G(f(u(t)))) + e(t)$ Where g and f are non-linear functions while G is a linear time invariant system.	Better analysis than NARX due to increased parameterization and extra use due to cascade connection of non-linearity and linearities.	Higher computational time and memory usage.
10	Multi Layered Perceptron	$y_k(t) = f\left\{\sum_{k=1}^{n} w_{ik}(t)x_k(t)\right\}$ Where x is the input vector, w is the weight vector and y is the output vector	Better approximators than RBFs	Complex due to extra usage of weights and more training samples compared to RBFs. Doesn't cover dynamics

Priority (low to high)	Model	Model representation	Pros	Cons
12	Radial Basis Functions	$y_k(t) = f\left\{\sum_{k=1}^{n} w_{ik}(t)øk(t)\right\}$ Where ø is the radial base function vector, w is the weight vector and y is the output vector	Uses simple local approximations which require fewer weights and training samples making the algorithm fast.	Simulation of RBFs is time consuming when compared to feed-forward MLPs. Doesn't cover dynamics
13	Time Delay Neural Networks	$y_i(t) = f\left\{\sum_{i=1}^{n}\sum_{k=1}^{d} w_{ik}(t)y_i(t - k)\right\}$ Where k is the delay operator	Covers dynamics at the input or feedback layer or all layers depending on the model type (tdnn, narx or lrn).	Computationally complex and not accurate compared to static NNs.

B. Error Analysis

A set of Dynamic and Static networks were considered for performing error analysis and operating on 2 individual networks (of same type – static or dynamic) at a time, the residual coefficients were computed and the network-duo which produced the least residual coefficient was noted. These entries denote the two networks which have the least correlated errors. Networks under consideration are given in Table B-1 and the $10 * 10$ matrices which show the duo residual regressions are in Table B-2. From this table, it is evident that dynamic networks 3 and 10 have the least error correlation; while for static networks do not have an error correlation less than 0.7. Hence negative correlation learning for static networks was implemented and produced better error correlation as discussed in the results section.

Table B-2: Error Analysis

	FTDNN Models Under Consideration					Static NN Models Under Consideration			
Model	**Hidden Nodes**	**TDL**	**Test MSE**	**Test R**	**Model**	**Hidden Nodes**	**TDL**	**Test MSE**	**Test R**
1	320	1	0.21667	0.88	**1**	80	NA	0.1919	0.89
2	312	1	0.21112	0.88	**2**	100	NA	0.1936	0.89
3	260	1	0.2311	0.87	**3**	60	NA	0.2093	0.87
4	180	1	0.23524	0.85	**4**	150	NA	0.2081	0.87
5	250	10	0.24615	0.83	**5**	170	NA	0.2156	0.87
6	240	10	0.2465	0.83	**6**	90	NA	0.2147	0.86
7	240	10	0.25073	0.82	**7**	180	NA	0.2211	0.85
8	260	10	0.25285	0.81	**8**	70	NA	0.2248	0.84
9	270	10	0.25524	0.81	**9**	120	NA	0.2293	0.84
10	250	10	0.25846	0.81	10	10	NA	0.2406	0.82

Table B-3: FTDNN Residual Correlation Coefficients

FTDNN Residual Correlation Coefficients										
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1	0.822	0.7902	0.7515	0.346	0.3211	0.331	0.333	0.344	0.32
2	0.822	1	0.7876	0.7497	0.37	0.334	0.342	0.341	0.352	0.32
3	0.7902	0.788	1	0.7835	0.316	0.2925	0.306	0.293	0.317	0.29
4	0.7515	0.75	0.7835	1	0.358	0.3248	0.361	0.351	0.362	0.35
5	0.346	0.37	0.3156	0.3584	1	0.8222	0.829	0.839	0.83	0.83
6	0.3211	0.334	0.2925	0.3248	0.822	1	0.837	0.837	0.841	0.83
7	0.3305	0.342	0.3057	0.361	0.829	0.8367	1	0.86	0.848	0.86
8	0.3328	0.341	0.2929	0.3514	0.839	0.8366	0.86	1	0.85	0.86
9	0.3441	0.352	0.3172	0.3623	0.83	0.841	0.848	0.85	1	0.86
10	0.321	0.324	0.2853	0.348	0.829	0.8321	0.863	0.855	0.865	1

Table B-4: NN Residual Correlation Coefficients

NN Residual Correlation Coefficients										
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1	0.79	0.7881	0.7814	0.721	0.7757	0.732	0.722	0.743	0.71
2	0.7901	1	0.7901	0.7955	0.757	0.7706	0.732	0.725	0.731	0.71
3	0.7881	0.79	1	0.7985	0.752	0.7656	0.726	0.733	0.719	0.71
4	0.7814	0.796	0.7985	1	0.762	0.777	0.743	0.74	0.74	0.72
5	0.7207	0.757	0.7515	0.7617	1	0.7261	0.695	0.703	0.7	0.69
6	0.7757	0.771	0.7656	0.777	0.726	1	0.768	0.821	0.805	0.82
7	0.7323	0.732	0.7257	0.7429	0.695	0.7684	1	0.752	0.751	0.75
8	0.7223	0.725	0.7327	0.7396	0.703	0.8208	0.752	1	0.825	0.88
9	0.7427	0.731	0.7186	0.7395	0.7	0.8051	0.751	0.825	1	0.83
10	0.7082	0.714	0.7101	0.7243	0.688	0.82	0.754	0.882	0.83	1

REFERENCES

1. Annuset, P. Simple Signals for System Identification, *Fourier Transform – Signal Processing*, Chapter 11, Intech, 2012.
2. Bealem, M. H., Hagan, M. T., Demuth, H. W., MATLAB Neural Network Toolbox™ User's Guide R2011b, *The Mathworks Inc*, 2011.
3. Buchanan, T., Lloyd, D., Manal, K., Besier, T., Neuro-musculoskeletal modeling: Estimation of muscle forces and joint moments and movements from measurements of neural command, *J Appl Biomech* 2004; 20(4): 367 – 395.
4. Crama, P. Hammerstein-Wiener system estimator initialization, *Proceedings of ISMA* 2002; 3: 1169 – 1176.
5. El-Shafie, A. Dynamic versus static neural network model for rainfall forecasting, *Hydrological Earth Syst. Sci. Discuss*.; 8: 2011, 6489 – 6532.
6. Erdemir, A. Model-based estimation of muscle forces exerted during movements. *Elsevier's Clinical Biomechanics* 2006; 22(2): 131 – 154.
7. Fernandez, J. Integrating modeling and experiments to assess dynamic musculoskeletal function in humans, *Experimental Physiology* 2006; 91: 371 – 382.
8. Forssell, U., Ljung, L. Identification of unstable systems using Output Error and Box-Jenkins model structures, Report No LiTH-ISY-R-1988, *IEEE Transactions on Automatic Control* 1997: 137 – 141.
9. Guler, N. F., Kocer, S. Classification of EMG signals using PCA and FFT, *Journal of Medical Systems* 2005, 241 – 250.

10. Halloran, J., Ackermann, M., Erdemir. A., van den Bogert, A. Concurrent musculoskeletal dynamics and finite element analysis predicts altered gait patterns to reduce foot tissue loading, *Journal of Biomechanical Engineering* 2010; 43: 2810 – 2815.
11. Halloran, J., Erdemir. A., van den Bogert, A. Adaptive surrogate modeling for efficient coupling of musculoskeletal control and tissue deformation models, *Journal of Biomechanical Engineering* 2009; 131(1): 11 - 14.
12. Isermann, R., Munchhof, M. *Identification of Dynamic Systems: An Introduction with Applications*, Springer, 2011.
13. Jamil, M., Sharkh, S., Hussain, B. *Identification of Dynamic Systems & Selection of Suitable Model,* Automation and Robotics, Chapter 7, Intech, 2008.
14. Karduna, A. R. Introduction to Biomechanical Analysis, *Kinesiology: Mechanics and Pathomechanics of Human Motion* 2009, Lippincott Williams and Wilkins, 2nd edition.
15. Knudsen, M. *Experimental modeling of dynamic systems,* Simulation and Experimental Modeling v 0.2, Department of Control Engineering, Aalborg University, 2004.
16. Kutz, M. *Biomedical Engineering and Design Handbook,* 2nd Edition, McGraw Hill, 2009.
17. Liu, Y. How to generate different neural networks, *Studies in Computational Intelligence* 2009; 35: 225-240.

18. Liu, Y., Yao, X. Negatively correlated neural networks can produce best ensembles, *Australian Journal of Intelligent Information Processing System* 1997: 176 – 185.
19. Liu, Y., Yao, X. Simultaneous training of negatively correlated neural networks in an Ensemble. *IEEE Trans. On Systems, Man, and Cybernetics, Part B: Cybernetics* 1999: 716 – 725.
20. Ljung, L. System identification toolbox user's guide, Version 8.1, *The MathWorks Inc*, 2012.
21. Ljung, L. System identification toolbox for use with MATLAB® , Version 5, 3.1 – 3.35, *The MathWorks, Inc*, 2002.
22. Lu, Y., Pulasani, P., Derakhshani, R., Guess, T. Application of neural networks for the prediction of cartilage stress in a musculoskeletal system, *Elsevier's Biomedical Signal Processing and Control* 2013; 8(6): 475 – 482.
23. Mishra, M., Derakhshani, R., Paiva, G., Guess, T. G., Nonlinear surrogate modeling of tibio-femoral joint interactions, *Biomedical Signal Processing and Control* 2010; 6: 164 – 174.
24. Otten, J. Inverse and forward dynamics: Models of multi-body systems, The Royal Society, *Phil. Trans. R. Soc. Lond. B Biol Sci* 2003; 358(1437): 1493 - 1500.
25. Paiva, G., Bhashyam, S., Thiagarajan, G., Derakhshani, R., Guess, T. A data-driven surrogate model to connect scales between multi-domain biomechanics simulations, *Conference Proceedings IEEE* 2012: 3077-3080.
26. Pintelon, R., Schoukens, J. *System Identification - A Frequency Domain Approach*, Second Edition, John Wiley & Sons, 2012.

27. Principe, J., Euliano, N., Lefebvre, C. *Neural and Adaptive Systems: Fundamentals through Simulations.* New York: Wiley, 1999.
28. Reaz, M. B. I., Hussain, S. Techniques of EMG signal analysis: Detection, processing, classification and applications, *Biol. Proced. Online* 2006; 8(1): 11 - 35.
29. Sinha, N. K., Gupta, M. M., Rao, D. H. Dynamic neural networks: An overview, *Proceedings of IEEE International Conference* 2000: 491 – 496.
30. Sherwood, J., Derakhshani, R., Guess, T. A comparative study of linear and nonlinear data-driven surrogate models of human joints, *IEEE Region 5* 2008*:* 1 – 6.
31. Soderstrom, T., Stoica, P. *System Identification*, London: Prentice Hall, 1989.
32. Tawhai, M., Bischoff, J., Einstein, D., Erdemir, A., Guess, T., Reinbolt, J. Multiscale modeling in computational biomechanics, *Engineering in Medicine and Biology Magazine* 2009; 28: 41 – 49.
33. Tortora, G., Derrickson, B. *Introduction to the Human Body*, 8th edition, John Wiley & Sons, 2010.
34. Viceconti, M., Taddei, F., Jan, S.V., Leardini, A. *et al*. Multiscale modeling of the skeleton for the prediction of the risk of fracture, *Clinical Biomechanics* 2008; 23: 845 – 852.
35. Yaffee, R., McGee, M. *Introduction to Time Series Analysis and Forecasting*, Chapter 3, 69 - 72, Academic Press, San Diego, 2000.

VITA

Palgun Reddy Pulasani was born on March 1, 1987, in Hyderabad, India. He received his Bachelor of Technology degree in Electrical and Electronics engineering from Gokaraju Rangaraju Institute of Engineering and Technology, affiliated to Jawaharlal Nehru Technological University, Hyderabad, India. After working at Tata Consultancy Services, the largest multinational IT firm in India, for over two years, Mr. Pulasani moved to the United States to pursue his master's program in Electrical Engineering at University of Missouri Kansas City where he secured a full grade point average (GPA) of 4.0. Whilst pursuing his master's, he worked as a Graduate Teaching Assistant for one year teaching Embedded System and Microcomputer laboratories. In 2013, he was awarded with lifetime membership in Golden Key International Honor Society which is the world's largest college honor society and is only given to students in the top 1% of the university. He also holds lifetime membership in other prestigious honor societies including Omicron Delta Kappa Leadership, Phi Kappa Phi and Eta Kappa Nu. His research interests include neural network modeling and machine learning.