Spring 2015

New neural network for real-time human dynamic motion prediction

Mohammad Hindi Bataineh
University of Iowa

NEW NEURAL NETWORK FOR REAL-TIME HUMAN DYNAMIC MOTION PREDICTION

by

Mohammad Hindi Bataineh

A thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Biomedical Engineering
in the Graduate College of
The University of Iowa

May 2015

Thesis Supervisors: Professor Karim Abdel-Malek
Adjunct Associate Professor Timothy Marler

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

PH.D. THESIS

_____

This is to certify that the Ph.D. thesis of

Mohammad Hindi Bataineh

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Biomedical Engineering at the May 2015 graduation.

Thesis Committee: _____
Karim Abdel-Malek, Thesis Supervisor

_____
Timothy Marler, Thesis Supervisor

_____
Jasbir Arora

_____
Nick Street

_____
Salam Rahmatalla

_____
Thomas Schnell

To Yasmeen, Alma, my family, and friends

Life is not about finding yourself. Life is about creating yourself

George Bernard Shaw

ACKNOWLEDGMENTS

I would like to thank my research advisor, Dr. Timothy Marler, for his endless directions and contributions to present my ideas fruitfully and clearly. I am especially grateful to my academic advisor and mentor, Professor Karim Abdel-Malek for his enthusiastic support and directions to work on exciting and appropriate topics. In addition, I would like to thank my other thesis committee members for their time and valuable feedback. I would also like to thank Melanie Laverman for the help in editing my thesis chapters, and all the group at the Virtual Soldier Research program for their efforts. Finally, above all, I am grateful to the God (firstly and lastly) and to my family for their endless smiles, confidence, and support though the past five years of my life.

ABSTRACT

Artificial neural networks (ANNs) have been used successfully in various practical problems. Though extensive improvements on different types of ANNs have been made to improve their performance, each ANN design still experiences its own limitations. The existing digital human models are mature enough to provide accurate and useful results for different tasks and scenarios under various conditions. There is, however, a critical need for these models to run in real time, especially those with large-scale problems like motion prediction which can be computationally demanding. For even small changes to the task conditions, the motion simulation needs to run for a relatively long time (minutes to tens of minutes). Thus, there can be a limited number of training cases due to the computational time and cost associated with collecting training data. In addition, the motion problem is relatively large with respect to the number of outputs, where there are hundreds of outputs (between 500-700 outputs) to predict for a single problem. Therefore, the aforementioned necessities in motion problems lead to the use of tools like the ANN in this work.

This work introduces new algorithms for the design of the radial-basis network (RBN) for problems with minimal available training data. The new RBN design incorporates new training stages with approaches to facilitate proper setting of necessary network parameters. The use of training algorithms with minimal heuristics allows the new RBN design to produce results with quality that none of the competing methods have achieved. The new RBN design, called Opt_RBN, is tested on experimental and practical problems, and the results outperform those produced from standard regression and ANN models. In general, the Opt_RBN shows stable and robust performance for a given set of training cases.

When the Opt_RBN is applied on the large-scale motion prediction application, the network experiences a CPU memory issue when performing the optimization step in the training process. Therefore, new algorithms are introduced to modify some steps of the new Opt_RBN training process to address the memory issue. The modified steps should only be used for large-scale applications similar to the motion problem. The new RBN design proposes an ANN that is capable of improved learning without needing more training data. Although the new design is driven by its use with motion prediction problems, the consequent ANN design can be used with a broad range of large-scale problems in various engineering and industrial fields that experience delay issues when running computational tools that require a massive number of procedures and a great deal of CPU memory.

The results of evaluating the modified Opt_RBN design on two motion problems are promising, with relatively small errors obtained when predicting approximately 500-700 outputs. In addition, new methods for constraint implementation within the new RBN design are introduced. Moreover, the new RBN design and its associated parameters are used as a tool for simulated task analysis. This work initiates the idea that output weights ($\boldsymbol{W}$) can be used to determine the most critical basis functions that cause the greatest reduction in the network test error. Then, the critical basis functions can specify the most significant training cases that are responsible for the proper performance achieved by the network. The inputs with the most change in value can be extracted from the basis function centers ($\boldsymbol{U}$) in order to determine the dominant inputs. The outputs with the most change in value and their corresponding key body degrees-of-freedom for a motion task can also be specified using the training cases that are used to create the network's basis functions.

PUBLIC ABSTRACT

Research in the field of human simulation has led to significant advancement in quality, time, and cost management for products like military and athletic equipment and vehicles. There is, however, a critical need for human simulation models to run in real time, especially those with large-scale problems like motion prediction (a single motion problem involves prediction of between 500-700 outputs). Hence, this thesis addresses that need by developing a new design of artificial neural network (ANN) that is capable of providing real-time motion results with maximum accuracy and minimal training. The success of the new ANN design is proven for the intended problem of motion simulation and other experimental and real-world problems. In addition, the design creates a new tool for the analysis of the task being simulated. The new implemented ANN algorithms will open a new area of advancement and capability in the digital human modeling field. Although the new ANN design is driven by its use with motion prediction problems, the consequent ANN design can be used with a broad range of large-scale problems. The motion problem is simply a well-studied example problem for the proposed developments. The new ANN design can be populated to be used for applications in various large-scale engineering and industrial fields that experience delay issues when running computational tools that require a massive number of procedures and a great deal of CPU memory.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	Artificial neural network
DHM	Digital human model
RBN	Radial-basis network
DOFs	Degrees of freedoms
PD	Predictive dynamic
FFN	Feed-forward network
SVM	Support vector machine
GRN	General Regression neural network
KBNN	Knowledge-based neural network
ELM	Extreme learning machine
ROM	Range of motion
SSE	Sum-squared error
MOO	Multi-objective optimization
OLS	Orthogonal least square
RMSD	Root-mean square distance
MSE	Mean square error
Opt_RBN	The new proposed optimized radial-basis network
RMSE	Root mean square error
MAE	Mean absolute error
CND	Constrained network design
LANO	Locally adaptive network outputs
NOIG	Network output as initial guess
IG	Initial guess
ZMP	Zero moment point

LIST OF SYMBOLS

$\boldsymbol{x}$	I-dimensional input vector ($\boldsymbol{x} \in R^I$)
$\boldsymbol{y}$	N-dimensional network output vector ($\boldsymbol{y} \in R^N$)
h_q	The q^{th} basis-function output
$\boldsymbol{h}$	Vector of the outputs from Q basis-functions
$\boldsymbol{w}_n$	Vector of weighting factors for the n^{th} network output
$\boldsymbol{W}$	Output weights matrix
r_q	The squared Euclidean distance for $\boldsymbol{x}$ from the q^{th} neuron's center
$\boldsymbol{u}_q$	The q^{th} basis-function center
$\boldsymbol{U}$	The basis-function centers
y_n	The n^{th} network output
σ_q	Spread (or Gaussian width) of the q^{th} basis-function
$\boldsymbol{\sigma}$	The basis-function spreads
Q	The number of basis-functions in the network
M	The number of training cases
$\boldsymbol{t}$	True output matrix
t_m	The true output value for the m^{th} training case
x_i^m	The i^{th} component of the input in the m^{th} training case
$x_{i_{max}}$	The maximum absolute value of the i^{th} component in the input vector
$\bar{x}_i^m$	Standardized value of the i^{th} component of the input in the m^{th} training case
σ_j°	Preliminary spread value of the j^{th} basis-function
$\boldsymbol{H}$	Basis-functions' outputs matrix
$\boldsymbol{Z}$	Set of orthogonal basis vectors (orthogonal matrix)
$\boldsymbol{z}_i$	The i^{th} orthogonal column in the orthogonal matrix
$\boldsymbol{A}$	Upper triangular matrix
$\boldsymbol{e}$	Error matrix

$\boldsymbol{g}$	Least square solution vector (scores of the all training case)
g_i	Least square solution of the i^{th} training case (score of the i^{th} training case)
$[err]_i$	Error reduction ratio
ε	The tolerance value
ε_1	The tolerance for error reduction ratio
ε_2	The tolerance for the MSE_k in the current iteration
MSE_k	Mean-square error at the current k^{th} iteration
MSE_{k-1}	Mean-square error at the previous iteration
$\boldsymbol{W}^o$	The network's preliminary outputs weight matrix
α_{jk}^i	The k^{th} element in the j^{th} row in the matrix $\boldsymbol{A}$
M_s	The number of selected significant number of basis-functions
x_1	The first input (variable) in the mathematical simulation example
x_2	The second input (variable) in the mathematical simulation example
y_1	The first output in the mathematical simulation equation
y_2	The second output in the mathematical simulation equation
y_3	The third output in the mathematical simulation equation
$\boldsymbol{q}$	Design variables (joint profiles)
$\boldsymbol{q}_{MoCap}$	Joint profiles provided by motion capture system
$\boldsymbol{q}_{NN}$	Joint profiles (outputs) produced by the neural network
$f(\boldsymbol{q})$	Human performance measure
G	The number of optimization problems to be solved (the number of degrees-of-freedom)
$\boldsymbol{W}_g$	The connection weight matrix that corresponds to the g^{th} group of outputs
$\boldsymbol{w}_{g,d}$	The output connection weight vector that corresponds to the d^{th} output in the g^{th} group of optimization
D	The number of outputs in each g^{th} group
λ_i	Weighting factor for the i^{th} cost function in multi-objective optimization
$f_i(\boldsymbol{x})$	The i^{th} cost function in multi-objective optimization

$f_i^{max}(\boldsymbol{x})$	The maximum value of the i^{th} cost function in multi-objective optimization
$f_i^N(\boldsymbol{x})$	The normalized i^{th} cost function in multi-objective optimization
$f(\boldsymbol{W})$	Cost function
$\Phi(\boldsymbol{W}, r)$	Composite function
r	Scalar penalty parameter ($r > 0$)
$P(\boldsymbol{h}(\boldsymbol{W}), r)$	Penalty function
$\boldsymbol{h}(\boldsymbol{W})$	Set of constraints
$h_i^+(\boldsymbol{W})$	Maximum constraints violation
x_i^{range}	The calculated range for the i^{th} input in the neural network
Δy_n	The change in value for the n^{th} output in the network
ΔDOF_d	The change in value for a DOF (group of network outputs)

CHAPTER I

INTRODUCTION

Artificial neural networks (ANNs) have been used successfully in various practical problems. Fundamentally, ANNs provide a means of modeling large sets of data. Conceptually, they provide a computational representation of how one takes in and processes data—of how one learns. Though extensive improvements in different types of ANNs have been made to improve their performance, each ANN design still experiences its own limitations. As with learning in general, there are many facets to neural networks. There are many ingredients, so to speak. Often, the challenge is not simply in applying a network to a particular problem, but in determining the most appropriate components for a particular problem in order to maximize computational speed and accuracy. In keeping with the analogy of mimicking cognitive performance, there is a distinct need for an ANN with improved performance, and responding to this need yields a tool with potential application to a broad range of problems like those in digital human modeling.

The existing digital human models (DHMs) are mature enough to provide accurate and useful results for different tasks and scenarios under various conditions. There is, however, a critical need for these models to run in real time, especially those with large-scale problems like motion prediction. Predicting motion problems can be computationally demanding. It takes time (minutes to hours) to predict a single task, even with small changes in the task conditions. Furthermore, simulations can be sensitive to changes in task parameters that extend beyond anticipated bounds. This in turn presents a case where accumulating a large amount of data (training cases from which to learn) can

be difficult. Hence, this work addresses that need using a new radial-basis network (RBN) design that is capable not only of providing highly accurate real-time motion results, but of providing them with minimal training.

The RBN is a powerful type of ANN to be investigated for predicting highly complicated problems like DHM motion. The RBN has been selected in this work because of its advantages when predicting regression problems, which will be discussed later, and its fast and successful training process, especially in large-scale applications. The RBN design, however, has some limitations. Therefore, this work presents a new RBN design for real-time prediction of large-scale motion problems with improved performance. Unlike traditional ANN designs, the new design's performance is preserved even with a reduced number of training cases. The design incorporates multiple training stages with approaches to facilitate automatic selection of the necessary network parameters in the training process with minimal heuristics. Along with the successful implementation of the new RBN design, methods for constraint implementation within the new design are also provided in this work. In addition, the design is used as a novel tool to perform biomechanical analysis for the simulated DHM problems.

The field of DHM has been of great interest to recent scholars. The field is full of potential impact on industry, the military, and healthcare in terms of saving time and money. Current DHM software can provide answers and valuable approximations for problems that are risky and difficult to test on a human subject. For example, calculating the maximum load and torque carried by the human spine is difficult without hurting the spine. This field also became important in many industrial, health, sport, and military areas because of the advancement in computer-based designs and software, which need

robust DHM models to test their prototypes. Instead of spending time and money building a new car prototype to test driver discomfort, engineers can use DHM software like Santos™ to have a virtual human with realistic anthropometries sit inside the prototype in the computer. Santos is physics-based DHM software with a realistic full body of 55 degrees of freedom (DOFs) that was developed by the Virtual Soldier Research (VSR) program at the University of Iowa. The software can provide posture and motion predictions for various tasks and scenarios. Then, information like joint angle and torque and compression load on each DOF is provided, as well as other measurements like ground reaction forces, balance, and energy consumption.

In physics-based DHMs like Santos, powerful and complicated methods are incorporated in the software to solve posture and motion prediction problems by optimization. Even though posture prediction is not a simple problem to solve, its optimization problem runs faster than the motion prediction one. The running time in posture prediction is in the fractions of a second, while the running time in motion prediction is in minutes. Motion prediction is more complicated, especially for a full DHM, because it requires dynamic prediction for all the body's DOFs under many constraints. The motion prediction method used in Santos, which is called predictive dynamics (PD) (Xiang et al., 2010), is flexible in showing the cause and effect of a predicted task(s). Computational speed, however, is the main limitation to making PD work in real time. Due to the size of the problem, which includes hundreds of design variables and thousands of constraints, the PD algorithm needs a long time, averaging in minutes, to run and provide the motion for a task. The calculations take time even for simple tasks or for rerunning a task with minor input changes. Hence, there is a need for

real-time motion prediction to allow the user to see immediate results for any changes in task inputs.

Successful implementation of ANNs in DHM motion problems will open new areas for the use of ANNs in DHM applications, which is still an active research topic with many issues to be resolved. Eventually, handling the complicated motion problems for instant predictions within a single ANN design should reveal more advancements and capabilities in the DHM field. The RBN design can be modified to handle large-scale problems accurately with a minimal number of training cases, and with no training or memory issues. The new RBN design depends on and adds to the wealth of previous research on ANN designs, which will be discussed in the next section, to implement multiple training approaches for maximum performance. The RBN, which is considered unconstrained optimization, can also involve a method for constraint satisfaction by either imposing the constraints within its training process, or by modifying the network output(s) to satisfy the constraints. Furthermore, the new RBN model can be used to extract useful feedback about the predicted task(s) like the relationships between different network inputs and their corresponding outputs, the most effective input(s) and key outputs, and the most important training case(s) to help the task validation and development processes.

1.1. Artificial neural network background

This section provides an overview of the technical aspects of ANNs. The reasons for selecting the ANN from among other data mining tools and statistical methods are

also illustrated. This in turn provides a foundation for discussing the state of the art and recently developed methods in the next section.

1.1.1. Biological analogy

The human brain is a decision system that has millions of units, called neurons, connected in a complicated way. The brain is more powerful and faster than any computer processor in handling complicated problems and providing a specific proper response for any mental, physical, or psychological situation. The brain is capable of doing so because it depends on memorizing and experiencing various situations a person faces to perform the relevant tasks. The human brain has multiple layers of neurons that interact with each other in parallel. This parallel interaction means that each neuron receives input lines from various neurons in the previous layer and sends different output lines to many neurons in the next layer. The neuron also sends values to the previous layer and receives values from the next layer. In addition, a neuron can still receive and transmit signals from and to other neurons even when some neurons' lines stop passing these signals because of the parallel connections. As in any other system, the received signal is called "input," and the transmitted signal is called "output." For a task to be learned and memorized by the brain, the signals received and sent to and from the neurons are set for the task to provide the proper decision.

The brain's powerful abilities include, but are not limited to, making proper decisions when solving various tasks like motions, postures, and mathematical calculations. This ability is gained by experiencing previous exact or similar examples for the solved tasks. A simple example is a baby who recognizes fire as something that hurts

and stays away from it after the first time he/she touches it. Based on the brain's underlying units' (neurons') functionalities in memorizing tasks by training (experience), studies have mimicked that architecture to duplicate the brain's ability to solve various complex practical problems efficiently. The new mimicked design is called an artificial neural network (ANN) (Figure 1.1). As the multiple connected neurons (i.e., a synapse connection) in the human brain, the ANN includes units, usually called neurons, distributed in different layers and all connected together. The left side of the figure shows a brain's neuron, while the right side shows an ANN with a single neuron.

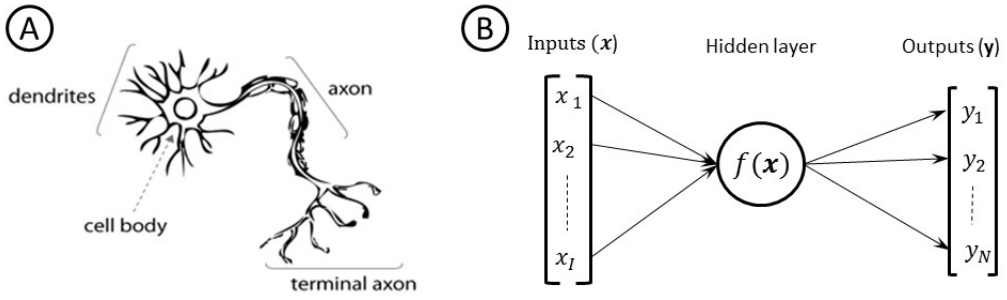

Figure 1.1: Schemes for human brain neuron and an artificial neural network (ANN).

An ANN is a mathematical approach consisting of multiple units, with parameters, connected together. The units' parameters are adapted to predict a system output(s) using data drawn from that system. Like human neurons, an ANN has three main parts: 1) network input(s) (***x***), which represent the dendrites in part A in Figure 1.1, 2) a hidden layer with multi-neurons (i.e., the neurons represent the network basis functions), in which each represents the cell body in part A, and 3) network output(s) (***y***), which represent the axon terminals in part A. Inside the hidden layer neurons, the main mathematical calculations occur to process the inputs and provide the proper outputs. The

axon, shown in part A in Figure 1.1, has a threshold value (i.e., weight) at which the neuron provides output only when the signal that goes to the axon is larger than its threshold. Like the axon, there are lines (connections) between the ANN neurons, called connection weights, where the values received and sent to and from the neuron differ depending on the weight value of the line. The weight value in each line is decided by a learning process that is performed in order to remember a task.

1.1.2. Artificial neural network as a data mining tool

Because of ANNs' powerful ability to predict the output of complicated problems, there are many successful applications for their use as pattern recognition models. Pattern recognition is the discovery of specific relationships (patterns) between various system inputs to provide proper output(s) correspondingly. The ability of ANNs has been pointed out in various sciences and industrial fields (Chakraborty, Mehrotra, Mohan, & Ranka, 1992; Fausett, 1994; He & Jagannathan, 2007; Lapedes & Farber, 1987; Patterson, 1998; Stinchcombe & White, 1989; Trippi & Turban, 1992; White, 1989; Widrow, Rumelhart, & Lehr, 1994; G. Zhang, Eddy Patuwo, & Y Hu, 1998). Depending on the type of problem and its accuracy requirements, different types of ANNs are used. An ANN is generally used in predicting all types of problems in data mining, which is the process of building a model that is useful, understandable, and usable from data by discovering its patterns.

There is a major difference between the focus and purpose of using ANNs, and all data mining models in general, and the traditional statistical tools in data discoveries. Data mining is more oriented toward applications than toward the basic nature of the

underlying phenomena of the data. For example, uncovering the nature of the underlying functions or the specific types of interactive, multivariate dependencies between variables is not the main goal of data mining. Instead, the focus is on producing a solution that can generate useful predictions. Therefore, data mining accepts techniques like the ANN to generate valid powerful predictions, but data mining is usually not capable of identifying the specific nature of the interrelations between the variables (i.e., a larger amount of feedback that can be drawn from statistics).

Data mining problems are divided into three main categories: 1) clustering, 2) classification, and 3) regression. Data clustering creates relationships between fed inputs and separates them into different clusters based on their similarities. Data classification assigns inputs to their classes from among different classes. Data regression means creating a curve that passes and fits between training data sets. More details about data mining concepts and techniques are provided in the literature (Han, Kamber, & Pei, 2006).

There are different types of ANNs to be used for the prediction of data mining problems. The Kohonen self-organizing network (Kohonen, 1990, 2001) is used in clustering problems. Regression and classification problems are usually predicted using the feed-forward back-propagation network (FFN) (Hecht-Nielsen, 1989; Rumelhart, Hinton, & Williams, 1985) and radial-basis network (RBN) (Broomhead & Lowe, 1988; Park & Sandberg, 1991), both of which have other subtypes under different names. The same type of network can be used to predict both regression and classification problems by using different transfer functions in its design, specifically in the neurons of the output layer. The typical option for classification problems is the sigmoidal function, while the

linear function is used for regression problems. The recurrent neural network (Pearlmutter, 1989; Williams & Zipser, 1989) is used for time series problems, which can also be considered regression problems. More details about various types of ANNs are provided in the literature (Bishop & Nasrabadi, 2006; Hagan, Demuth, & Beale, 1996).

Among data mining techniques, the ANN is preferred in many applications, including those in this work, for the following reasons:

1. The ANN has a relatively simple design and a fast training process, given the powerful results. Compared to other competitive data mining techniques like support vector machines (SVMs), which require the transfer function to be determined by the user depending on the desired model complexity and optimization with a large number of parameters, the ANN design is simpler to implement and train. That makes the ANN the first option for many scholars. ANN is also especially easy and flexible to implement in hardware (Mathia & Clark, 2002; Moerland & Fiesler, 1997).
2. The ANN can learn various types of problems using the same transfer function (basis function). The transfer function is the mapping function that is used to transfer the input space to the output space (i.e., it is the function that creates the relationships between the inputs and outputs). Other data mining methods require the user to determine the proper transfer function to be used for each problem in advance, which is hard to determine. Furthermore, even with simple transfer functions, methods like SVM require optimization to be run for a large number of parameters, which produces issues for many optimization tools (Collobert, Bengio, & Bengio, 2002).
3. The ANN has been proven to outperform competing data mining techniques in many practical applications, as will be shown by the work of many scholars presented later

in this section. Flexible ANN design with adaptable complexity to various problems allows for handling various problems successfully.

1.1.3. Regression with artificial neural networks

Since this research focuses on DHM motion prediction, which is a regression problem, the ANN details are narrowed to discuss the design of ANNs for regression problems. The types of ANNs that are typically used in regression problems, which mainly include FFN and RBN, act like a multi-dimensional curve-fitting model (i.e., regression curve). The statistical term "curve fitting" refers to the process of constructing a curve (i.e., a mathematical function) that has the best fit for a group of data. These fitted curves are constructed to be used for 1) aiding in data visualization, 2) providing values of a function where no data are available, and 3) extracting and analyzing the relationships among various inputs (i.e., variables). More details about regression analysis and curve-fitting topics are covered in the literature (Deming, 1944; Kleinbaum, Kupper, Nizam, & Rosenberg, 2013).

The following specifically summarizes the reasons for using an ANN in predicting regression problems rather than traditional statistical regression methods:

1. The ANN runs faster than the conventional curve-fitting tools (Bishop & Roach, 1992). This is especially true when the curves are nonlinear, which is the case for most practical problems. The speed of ANN calculations is due to the parallel calculations for its hidden units.
2. The user in regression models should have some knowledge about the complexity of the problem (the type or order of the curve required). That knowledge is not available

or accurate in most cases. On the other hand, the ANN fits the problem without these assumptions and determines the level of complexity needed to properly predict the problem. Moreover, the adapted ANN parameters to fit the problem might imply practical significance and insights about the problem.

3. Most traditional curve-fitting tools require building a separate curve (model) for each problem's predicted output. On the other hand, a single ANN model can be used successfully to predict a relatively large number of outputs, in the hundreds (M. Bataineh, Marler, & Abdel-Malek, 2012).

In general, when an ANN is designed, its complexity depends on the problem to be solved. The complexity of the problem is proportional to the number of inputs that the network needs to handle, the inter-relationships of the inputs, and the number of outputs the network needs to predict. Building an efficient and accurate ANN requires careful study of issues related to the network, including: (1) a universal function approximation capability (i.e., ability to generalize the problem prediction), (2) resistance to noise or missing data (i.e., filtering the noise from data and extracting the features' relationships properly), (3) accommodation of multiple nonlinear variables for unknown interactions, and 4) choosing the proper type of ANN for the best problem prediction (Geman, Bienenstock, & Doursat, 1992; Twomey & Smith, 1998).

1.2. Literature review

This section reviews the state of the art in the field of ANNs and their applications, with a focus on the applications related to dynamics and DHM. The section is divided into subsections to illustrate detailed information about different categories

related to ANN design developments and applications. The deficiencies and limitations in each category are also presented. In addition, many definitions and terminologies are provided throughout the section.

1.2.1. Techniques in artificial neural network design

This section reviews the current state of the art with respect to the major developments in ANN designs and training approaches. Separate subsections are presented for the well-known approaches that have been introduced to improve the prediction capabilities of ANNs and other data mining techniques. Basically, most presented approaches experience limited accuracy (performance) capabilities related to either poor training with heuristic settings, training algorithms that are specialized for limited types of problems, or a combination of the two. Other techniques experience training difficulties when applied for large-scale problems. Therefore, such approaches cannot be used in designing an ANN model for the large-scale application of DHM motion problems.

In addition to its powerful ability in problem prediction relative to other data mining and statistical approaches, the ANN has an advantage in computational speed. When a powerful computational tool is needed for fast prediction of a complicated system's output, many scholars have presented the ANN as a method that provides real-time and stable performance over the slowly converged optimization approaches (Cochocki & Unbehauen, 1993; Kennedy & Chua, 1988; Maa & Shanblatt, 1992; Xia, 1996; S. Zhang & Constantinides, 1992). In one study (Xia & Wang, 1998), for example, the authors proposed a methodology for globally convergent optimization of ANNs and

tested it on theoretical examples. Comprehensive testing for these proposed ANN-based methods, however, was not done on practical complicated problems. Moreover, the accuracy and generalization ability of the outputs from those methods were not investigated.

The originally introduced ANN design, which is the FFN and its variants like those presented in the literature (Huang, 2003; Reed & Marks, 1998), has been modified and applied to work for various types of classification and regression applications. Some of the state of the art for the currently available types of ANNs and their modified versions are presented in the literature (G. Zhang et al., 1998). As an example for such design modification, Zhang and Wu (2008) proposed an improved optimization approach for solving the local optimum problem in FFN design. The network's connection weights in that approach are first assigned to random numbers between -1 and 1. Then, the bacterial chemotaxis optimization approach is used to find the optimum weight values in the network. Although the proposed method was tested on some examples, there is still a computational issue with the use of the FFN in practical large applications that have a relatively large number of design variables. Moreover, design improvements should be focused more on types like the RBN because it already provides global solutions. The RBN can be trained without local minima issues, as shown in the literature (Bianchini, Frasconi, & Gori, 1995; Haykin & Network, 2004), because the used cost function is local minima free with respect to all the weights (i.e., the design variables) However, its parameters need to be selected properly to achieve proper performance.

Other scholars have been performing modifications on the traditional ANN models to meet some requirements and/or accuracy for the problems being solved. Platt

(1991) presented a dynamically changed ANN design to allocate new hidden units and adjust the parameters of the existing ones when poor network performance is obtained for new inputs. The proposed network, however, might experience stability issues because its architecture is not fixed. Poor performance could happen when testing new inputs with the modified network's parameters. As an alternative to the random or heuristic methods in assigning the Gaussian basis-functions' widths in RBN, scholars have assigned the widths to be found by the distance to the nearest center (Moody & Darken, 1988; Moody & Darken, 1989) or by the k-means method (Lloyd, 1982; MacQueen, 1967). Although the presented designs generally show faster learning processes with a similar number of hidden units, the accuracy improvements are neither noted nor investigated. The fast learning process is only necessary for an online training process like time series applications. The main goal in most regression and classification applications is to obtain the best possible accuracy, since the produced trained ANN will already run quickly.

One of the known special ANN designs is the general regression network (GRN) (Specht, 1991). That design was proposed as a fast curve-fitting tool that provides its output as a weighted sum of the cases used in the training process. As an example of a successful GRN application, it was used in large-scale motion prediction problems with a limited number of training cases (Mohammad Bataineh, 2012). That work illustrates that it is possible to use an ANN to provide acceptably accurate results for a large problem using few training cases. The GRN was designed to predict 330 outputs using 52 training cases. The GRN, however, has limited generalization capability because it has no underlying training process in its design. The GRN design experiences over-fitting issues, which occur because the ANN is specialized in predicting the training data but poorly

generalized for prediction of any other cases. In addition, the GRN might experience unstable performance when the available training data is too limited.

Different approaches are proposed to help solve the limited performance of ANNs and other models in the pattern recognition field. Basically, these approaches focus on assisting more generalized network performance. With more generalization, the network has less error when predicting its output(s) for new test cases of the problem being predicted. The main and most popular approaches are summarized in the following subsections.

1.2.1.1 Ensemble

The *ensemble method* was introduced to resolve poor generalization in the ANN and other data mining tools and has been applied in various ANN designs and applications. The idea of the ensemble ANN is to design a combination of different networks (i.e., predictors) that are trained individually. Then, the prediction (i.e., decision) in the ensemble is performed by majority voting in classification and averaging their outputs in regression. A weighted combination of networks is also used.

In general, the complexity of the ANN design is chosen to reflect the complexity of the predicted system, which is unknown in most cases. Therefore, a more complex ANN is not always necessary for better problem generalization. It is also hard to find the optimal needed network design because most traditional ANNs depend mainly on evaluating the training error and some testing cases for the predicted problem. Consequently, ensembles of ANNs are used to improve the predictions. The ensembles have been investigated by several scholars in the neural networks community (Hansen & Salamon, 1990; Krogh & Vedelsby, 1995; Perrone & Cooper, 1992; Wolpert, 1992). The

most common types of ensembles are boosting (Freund & Schapire, 1995; Schapire, 1990) and bagging (Breiman, 1996). The methods differ in the way the training process is performed and the training sets are generated.

The ANN ensembles have been used successfully with superior performance in many applications (Cherkauer, 1996; Drucker, Schapire, & Simard, 1993; Hansen, Liisberg, & Salamon, 1992; Sharkey, 1999; Zhou, Jiang, Yang, & Chen, 2002). The ensembles, however, experience over-fitting and poor generalization problems in other applications. Scholars have shown that a basic single ANN or an ensemble ANN with fewer networks may perform better than one that includes all ANNs (Zhou, Wu, & Tang, 2002). That work employed an approach called GASEN to show that using few ANNs in the ensembles is better than using all of them. The study showed that the same results apply in both regression and classification problems. Ting and Witten (2011) also addressed some issues in the stacked generalization method, which is a type of ensemble proposed in the literature (Wolpert, 1992) to produce more appropriate performance.

1.2.1.2 Knowledge-based neural network

Knowledge-based neural network (KBNN) is a hybrid learning method for an ANN that uses two algorithms, a theoretical knowledge-based algorithm and data-based algorithm (Towell & Shavlik, 1994). Both algorithms are combined in an integrated training process that should lead to more generalized performance than that obtained using a single algorithm. The knowledge-based algorithm depends on understanding and defining the closest theory of the problem's solution domain to provide the initial network model. The data-based algorithm corrects the initial network to provide the final trained network. The data-based algorithm is similar to any typical ANN learning process

that depends on the available training data. The resulting KBNN is a mixture of both algorithms to reach the most proper prediction. The KBNN, when both algorithms are initiated properly, outperforms other typical learning methods.

On the other hand, the KBNN needs prior knowledge about the predicted problem that might be hard to obtain. The KBNN also causes problems for the typical data-based algorithms because the initial network already has training error around zero. That in turn allows for minor changes to be produced by the second algorithm before setting the final network, since that algorithm also depends on the reduction of the training error. Attempts have been made to address this issue by making major changes to the used data-based algorithms (Hinton, 1989; Watrous, 1988). Although the knowledge-based algorithm can handle some types of problems, the classification types in specific, it typically experiences limited performance when predicting the regression problems (Towell & Shavlik, 1994).

1.2.1.3 Extreme learning machine

The *Extreme learning machine* (ELM) was introduced recently as a new training algorithm that improves the speed of the traditional learning process in ANNs, as well as improving ANN performance (Huang, Zhu, & Siew, 2004). The method replaces the gradient-descent-based optimization with a faster method that depends on calculating the network output weights directly using the hidden outputs. Since its introduction, the method has undergone various modifications and developments (Cao, Lin, Huang, & Liu, 2012; Deng, Zheng, & Chen, 2009; Huang & Chen, 2008; Huang & Siew, 2004; Huang, Zhou, Ding, & Zhang, 2012; Huang, Zhu, & Siew, 2006). The ELM method has been applied in many applications (Liang, Saratchandran, Huang, & Sundararajan, 2006;

Statnikov, Aliferis, Tsamardinos, Hardin, & Levy, 2005; Suresh, Venkatesh Babu, & Kim, 2009; R. Zhang, Huang, Sundararajan, & Saratchandran, 2007). The ensembles idea was also incorporated in the ELM method (Sun, Choi, Au, & Yu, 2008), where the final predicted output is the average of the ELM outputs. The resulting ensemble model had a better generalization performance than that of a single ELM. More work on ensemble ELM was conducted for applications in time series prediction (Lan, Soh, & Huang, 2009) and fast online sequential learning (Van Heeswijk et al., 2009). A detailed state of the art of the method's algorithms, modifications, and future work is provided in the literature (Huang, Wang, & Lan, 2011).

On the other hand, application of ELM has limitations, including the following: 1) its generalization performance is still questionable; the method focuses on obtaining zero training error, which can be achieved easily in any simple learning or optimization method; 2) input weights and biases, or hidden centers and hidden function impacts in RBN, are assigned randomly in the original ELM and its recent modifications, which limits the network performance, leaves some heuristics for the user that might be assigned badly, and produces a network with more unnecessary hidden neurons; 3) the ELM algorithm is not sensitive to the number of hidden neurons compared to the back-propagation algorithm; and 4) newer modifications of the method slow its learning speed because of iterative learning steps (Huang & Chen, 2008). Some of these limitations have been investigated by later scholars (Zhu, Qin, Suganthan, & Huang, 2005) to resolve the randomness in assigning input weights and bias by using differential evolution. Suresh et al. (2010) also introduced the sparse ELM to reduce the computational time in ELM.

1.2.2. Large-scale applications

Although there has been limited work on the use of ANNs with large problems because of training and accuracy issues, ANNs have nonetheless been used to solve some specific types of large-scale problems. The definition and interpretation of large-scale problems are first presented in this section, followed by a review of methods used specifically for such problems. These works have shown promising results.

Typically, when a scholar of ANNs, and data mining in general, refers to application as large-scale problem, it means that the problem has a large number of training cases. That is because most applications have few inputs and outputs (on the order of tens or less), while it is common to have applications with large amounts of training data (on the order of thousands or tens of thousands) to be considered in the training process. On the other hand, applications like DHM motion problems involve a relatively small number of training cases and inputs (on the order of tens or hundreds) and a large number of outputs (on the order of hundreds). Motion problems can also be called large-scale in terms of the ratio of outputs to input or outputs to the number of training cases. The motion large-scale problem is a unique and challenging problem for the effort of developing a new ANN in this work.

Research has shown that some types of ANNs, specifically the FFN, cannot be used for large-scale problems because of memory and poor convergence issues. The FFN requires a massive number of parameters to be calculated and uses a back-propagation training paradigm that can become stuck easily at local poor solutions. Other models of ANNs, such as the RBN, which will be illustrated in Chapter 2, show superior performance in such applications; however, they still take a long time to be trained and

produce a large network that runs slowly. There has been limited work on getting ANNs to work more efficiently for large-scale problems. Most of that work considers only problems with large training cases (sets) because they are the common and typical type of large-scale problems.

Scholars working on large-scale problems and large training sets have focused on the use of the RBN or multi-ANNs as the solution for the memory and poor generalization issues in the training process. For instance, sparse Gaussian methods, which are considered Bayesian approaches, were introduced to solve such problems (Quiñonero-Candela & Rasmussen, 2005; Snelson & Ghahramani, 2006). A mixture of ANNs, specifically the RBN, was presented, in which outputs are a weighted average to provide the final output for regression problems with more than hundreds of thousands of training cases (Lázaro-Gredilla & Figueiras-Vidal, 2010). When compared to the sparse Gaussian process methods, the method showed improved performance and comparable computational cost. Heeswijk et al. (2011) also presented faster ensemble ELMs for large-scale data set regression applications by using multiple computer CPU cores and parallel calculations.

Limited work on the interpretation of a large-scale problem in terms of the ratios of output to input and/or output to the number of training cases has been performed recently. The GRN was used to predict 330 outputs for the DHM motion prediction tasks of walking and jumping on a box using 52 and 24 training cases, respectively (Mohammad Bataineh, 2012; M. Bataineh et al., 2012). The results were relatively accepted, especially for cases with inputs close to the training cases. The network running time was also in the fractions of seconds in both training and testing modes. Despite the

aforementioned advantages of the GRN, its design has some capability limitations, which are provided in Section 1.1.2. In general, the ANN is potentially useful for producing real-time and more accurate results for such large-scale problems.

1.2.3. Constrained problems

With applications like dynamic human motion prediction, it is critical that all constraints, like joint and torque limits, contact points with the ground and other subjects, etc. are satisfied. However, direct use of ANNs can result in unsatisfied constraints and thus unrealistic simulated motion. This section provides some examples from the literature of successful incorporation of constraints when using ANNs for specific applications. Since typical training processes for ANN involve unconstrained optimization, constraints are generally incorporated beyond the network design. General issues are discussed regarding the constraint implementation within ANN designs in the provided literature.

In the pattern recognition community, the ANN is considered a non-constrained optimization tool that runs extremely fast once its training process is completed. The ANN's design and its parameters are all set in the training process, which is illustrated in Chapter 2, to predict a specific task. Then, the network can easily predict any new task inputs because it is set to predict such tasks. Some scholars have investigated the use of the powerful ANN capability in the fast prediction of complex tasks to predict tasks that involve some constraints to be satisfied. Such works are known under different names, including but not limited to adaptive constrained neural network, dynamic ANNs, constrained optimization using ANNs, and the hybrid ANN approach.

Yang and Wang (2000) proposed an adaptive ANN with constraint satisfaction for job-shop scheduling problems. The network adjusts its weights and biases to satisfy the constraints along with considering some heuristics for the presented design. The use of dynamic recurrent neural network has been proposed to solve constraint optimization problems (Bouzerdoum & Pattison, 1993; Xia, Leung, & Wang, 2002; Xia & Wang, 2005; Y. Zhang & Wang, 2002). The proposed models were tested successfully on some numerical and experimental examples, but not on a practical complex application. The speed of computations and performance in such simplified network models is also arguable for general applications. The recurrent network model was, then, expanded to be applied on manipulators that are constrained by physical joints and velocity limits (Y. Zhang, Wang, & Xia, 2003). The work performed distributed parallel computations to obtain fast online outputs for a simple 7- DOF robotic manipulator. Moreover, Zhang and Li (2009) introduced another ANN model to solve online time-varying problems subject to linear equality constraints.

Gholizadeh et al. (2008) presented an ANN to find the optimal weight of structures that are subject to natural frequency constraints using a combination of RBN and genetic algorithm. The study used RBN and wavelet-RBN models to provide approximations for the natural frequencies to improve the computational speed of the proposed optimization algorithm. That work, however, employed other optimization tools to perform the optimization; ANNs were used to provide predictions for the constraint values for calculation speed-up. In general, almost all these works are task-specific; the proposed approaches work only for the intended tasks by updating the network design parameters or complementing optimization tools to satisfy the constraints. In these

special network designs, the networks need to run through extra computationally costly steps to adjust their parameters to satisfy the violated constraints. In addition, even with satisfied constraints, the updated designs might not be guaranteed to provide feasible, successful, and accepted solutions for all input cases. The ability of ANNs to produce fast and reliable solutions becomes vulnerable thereafter.

1.2.4. Dynamics and human simulations

This section focuses on presenting the state of the art for the use of ANNs in dynamics applications, especially those in human and robotics motions. A summary of the existing limitations in the field, specifically those related to results with limited accuracy and prediction of complex problems, is also presented.

There has been limited research in the use of ANNs in the DHM and virtual reality fields because these fields are relatively new. The main use of ANNs has been focused on human model posture prediction (Jung & Park, 1994; B. Zhang, Horváth, Molenbroek, & Snijders, 2010) and motion prediction of robotics and dynamics systems (Frank, Davey, & Hunt, 2001). Motion-related applications include, but are not limited to, robotics and controller system motion, motion analysis, reconstruction of dynamic objects, and time-series dynamic prediction and classification. Recently, researchers have shown more interest in using different types of ANNs as prediction models to handle motion and kinematics problems for the following reasons:

1. Dynamics problems, especially human motion, are complicated to duplicate using simple prediction models. These problems can be modeled using physics-based models (Xiang et al., 2010), but it is computationally costly. Prediction models can be

also designed for the problem using prerecorded data or various combinations of inputs and their corresponding outputs. However, for many problems, that is either costly or unavailable. Hence, ANNs are good candidates to provide prediction for such problems because they have been proven to be successful in providing instant results for complicated problems (Hagan et al., 1996; Looney, 1997). An ANN can provide acceptable results for many problems without understanding the underlying relationships between the problem's variables.

2. Reliable training sources for ANNs are widely available. Recently, more advanced motion capture and video systems have been developed to capture human and object dynamics accurately and naturally. In addition, various humanoid and digital human motions can be obtained from simulation software, physics-based digital models, and gaming modeling.

There are many works that focus on using ANNs in robotics motion and the manipulation of their controller systems. Some scholars consider motion prediction as a time series problem to be predicted; Frank (2001) provided some investigations on ANN performance in that area. Stakem and AlRegib (2008) used an ANN for predicting human arm movement in a virtual environment. The ANN in that work, the FFN, was used as real-time predictor to solve delay problems in the sensor and network that were used to detect the arm motion. The FFN has six inputs, six hidden neurons, and eight outputs, representing 800 ms of arm movements. In a simple task like performing a reaching motion, the predicted movements after 400 ms (the first 4 outputs) were less accurate than those before that time. The prediction was inaccurate after 100 ms when tested in more complex tasks like shoe tying. In the summary, it was suggested that for further

successful predictions, the network should be either retrained with more data or predict less than 200 ms (fewer outputs). The work, however, did not recommend investigating the use of other more accurate types of ANNs like the RBN.

Another dynamics application using ANNs was a robotics controller (Lewis, Yesildirek, & Liu, 1996). A modified type of FFN was proposed where its weights were updated on-line to act as a disturbance in robotic arm motion. The network performance with 10 hidden neurons was tested to adjust the joint angles of a planar two-link arm. It was found that adding the ANN improved the tracking performance of the system significantly. On the other hand, the running time for updating the network weights was slow (around 1 minute). Furthermore, the number of hidden neurons is heuristic, so the model simulation should be repeated many times with different numbers before selecting the final proper network design. In a similar application, Kun and Miller Iii (1996) proposed an adaptive dynamic balance scheme for a robot using an ANN. Three neural networks were used to balance the motion of side to side, forward to back, and preserving foot contact for a two-legged walking machine (Miller III, Glanz, & Kraft III, 1990; Miller, Werbos, & Sutton, 1995). The networks were not involved in the prediction of the actual motion. Kim and Lewis (1999) also proposed the use of two ANNs in controlling a robot manipulator. The system was evaluated successfully on a two-link robot manipulator, demonstrating the ANN's ability to handle the nonlinear unknown parameters in the system manipulator.

Even though the powerful computational ability of ANNs is outstanding, their use within direct and complicated motion prediction problems is still limited. That might be because such problems involve too many input parameters and outputs, which requires

many combinations of cases for the ANN to be trained well. In addition, the typically-used FFN in many applications experiences memory and training issues when used for relatively large-scale problems like motion and DHM applications. Most scholars working on motion predictions, which are relatively new, consider using the traditional FFN type of ANNs. There have been, however, some successful uses of ANNs in predicting a complete motion profile in the DHM field (M. Bataineh et al., 2012). In that work, the GRN was used to predict 330 outputs that represent profile values for a full 55 DOF in the tasks of walking and jumping on a box. Although the GRN has some limitations, the results were promising in terms of the acceptable accuracy achieved and the extremely fast training time.

In addition to their prediction capabilities in dynamics problems, ANNs can be useful in extracting and analyzing some information about the predicted problem. Koike and Kawato (1995) presented the use of surface electromyography signals as inputs for two FFNs to dynamically predict joint torques. Consequently, the predicted torque values were used to reconstruct human arm movement with a forward dynamics model. The experiments were limited to the shoulder and elbow joints, where two networks, each with a single output, were used for predicting the torque of each joint to improve the accuracy. An interesting conclusion was drawn from that work: that ANNs could be helpful in forming some complicated calculations in the human environment like arm stiffness and operations in the central nervous system. Despite the promising results in that work, various joint torques should be combined in one model in order to provide more insight on other calculations like arm stiffness because the shoulder and elbow motions and torques, for instance, are implicitly related to and dependent on each other.

Yoo et al. (2008) also used the FFN, which was trained by gaits of various people, to recognize humans automatically. The network had an architecture of 10 inputs, 28 hidden neurons, and 13 outputs. The work elicited the potential use of ANNs as methods that are able to improve gait analysis by handling the highly nonlinear relationships within gait parameters.

In other motion-related applications, ANNs have been used as an indirect source that led to providing improved motion predictions. Lung tumor motion during respiration was predicted in advance using an FFN (Isaksson, Jalden, & Murphy, 2005). Their network architecture included two hidden neurons and one output. Hong et al. (2002) presented an expression-based real-time face animation using an FFN. When the network is fed with a face expression, it can classify that expression as smiling or sad. Different FFNs were also evaluated in that work for predicting many types of expressions. Ishu et al. (2004) introduced a modified method for setting recurrent neural network parameters using automatic heuristics. The method was tested on the motion identification of an underwater robot. The results show that the proposed method was slow; many runs needed to be performed, and each took around half a day to complete. Moreover, the method was not tested on more complex problems.

Most of the preceding scholars used FFNs with single or few outputs to preserve accuracy. This, in turn, limited the use of ANNs in many more complicated applications because of the FFN's history of producing poor local optimum solutions, as well as experiencing memory issues with large numbers of inputs and/or outputs. Therefore, it is crucial to investigate the performance of other types of ANNs in such applications, especially for the applications related to DHM motions that are studied in this work.

1.3. Summary of literature review and motivation

Although extensive research has been conducted with ANNs, there are gaps in the current state of the art with respect to prediction models for large-scale problems. In this context, the term “large-scale” refers to the ratio between the number of outputs and the number of inputs and/or training cases. Specifically in DHM motion prediction problems, the large-scale problem could be referred to as the number of outputs. There is a critical need, especially in the field of DHM, to develop a single model that is capable of providing instant realistic prediction of whole-body motion. Moreover, the prediction model should be designed in anticipation of minimal available training cases, as producing training cases by using dynamic motion prediction can be computationally expensive. To our knowledge, except for one study (Mohammad Bataineh, 2012), there has not been work toward successful motion prediction of full-body DHM. The results of that work were promising in terms of the accuracy produced for the given complex task and the fast training and running times. Therefore, this work investigates the development of the ANN as a means to predict the real-time and accurate results necessary for DHM motion tasks. In pursuing this investigation, a new RBN design is introduced to overcome the current limitations in the network design. The new design is also used to study and provide insights regarding the inherent parameters in the predicted tasks.

The knowledge-based neural network (KBNN) (Towell & Shavlik, 1994), which is a hybrid learning method for an ANN, is conceptually similar to this work as far as proposing multiple stage training process for more accurate results. However, the KBNN needs prior knowledge about the predicted problem that might be hard to obtain. The KBNN also causes problems for the typical data-based algorithms because the initial

network already has training error around zero. That in turn allows for minor changes to be produced by the second algorithm before setting the final network, since that algorithm also depends on the reduction of the training error. Moreover, although the knowledge-based algorithm can handle classification problems, regression problems are typically difficult to handle (Towell & Shavlik, 1994). In contrast, the new ANN design in this work proposes two data-based approaches that are used to set all network parameters and reduce the training error. The new design is mainly developed to enhance the prediction capabilities of the regression types of problems with application in the DHM field.

The use of typical FFNs and their variants in large-scale applications like motion prediction is limited due to their computational issues and poor performance. As stated, the RBN has been found to be a better alternative for such regression applications, although its design requires some modifications. Extensive improvements and developments were recently introduced to various ANN design and training techniques, but there are still some limitations related to the generalization (i.e., limited accuracy) and applicability to extreme applications like large-scale problems and those with reduced training data. Although some of the limitations like the computational speed have been addressed in new approaches like ELM, such approaches' generalization performance and sensitivity to various network parameters are still questionable. The new RBN design proposed in this work should solve the limitations for regression types of problems, especially by the minimal heuristics design and improved robust performance.

To date, given that ANN is considered a non-constrained optimization tool, there is no specific method that proposes a robust, fast, and accurate ANN design that

incorporates constraints to be completely satisfied within its training process. Almost all the works in the literature that apply ANN with constraints are task-specific. The approaches proposed in the literature work only for the intended tasks by either updating the network design parameters or complementing the network output(s) with an optimization tool for constraint satisfaction. As a result, some of these approaches create new issues for the network performance and slow down production of the final output(s). This work evaluates new approaches for constraint implementation that are conceptually similar to those proposed in the literature, but that might not affect the results produced from the new RBN design. In addition, the new approaches can maintain the speed of ANN calculations. Unlike the existing methods, the new approaches can be applied for any task and any type of constraint.

This work targets a specific type of constraints in DHM motion problems that are difficult to satisfy, even with the highly accurate results from the new RBN design. There are two new approaches to be evaluated in this work for constraint implementation within the new RBN design. Unlike the approaches presented in the literature, the first approach satisfies the constraints within the training process. The network parameters are optimized so that all the constraints are satisfied for the provided training data. The first approach may lead to a network with stable, accurate, and fast performance in the testing phase (i.e., when new input cases are provided). However, this approach cannot guarantee constraint satisfaction at various task conditions. Nonetheless, in practical applications like those in DHM motion, a slight violation in the task constraints is usually allowed. Therefore, the first approach is evaluated toward that goal, as long as the network produced computationally fast and acceptable solutions. The second approach

includes local modification of the network output(s) to satisfy the constraints. With a well-trained network, and after it provides its outputs, they are modified by solving an optimization that satisfies the task constraints. The optimization finds the network outputs (design variable) to minimize a cost function that represents the difference between the network outputs and the final ones and is subject to the constraints.

Moreover, this work focuses on implementing the new RBN in the Santos software environment to be trained and run to provide real-time motion prediction, and to possibly be populated for the prediction of other problems. Some work (Ishu, van Der Zant, Becanovic, & Ploger, 2004) pointed out the need for automatic methods to run the ANN quickly; this requires careful selection and modification of the approach incorporated in the new RBN design. Such requirements are considered in the algorithms used in the new design to facilitate automation of the whole training process to be performed by any user.

Last but not least, other applications and analyses in the DHM predicted motion problems are performed in this work. The new RBN is used to draw biomechanics feedback about the predicted task. The predicted motion is analyzed from the network parameters to address topics including, but not limited to: 1) the relationships between the task inputs and outputs, 2) feedback from the important training case(s) to improve the task development and validation, and 3) the task key conditions (inputs) and joints (outputs).

The aforementioned deficiencies with respect to the performance of various ANN models when are applied for special applications like motion prediction problems arise the need for new ANN design with improved performance. The new design in this work

needs to have special improved performance when simulating the large-scale applications with minimal available training data like the DHM motion prediction. This work is motivated by the following:

1. The FFN, which was the first introduced and is the most commonly used network, experiences poor performance in many applications. The poor performance is especially apparent when it is used in regression problems. The RBN is more powerful for predicting highly complicated regression problems like DHM motion.
2. There is a need for developing a new ANN design that is able to produce robust results for large-scale problems with a reduced number of training cases. The robust design should use approaches with minimal heuristics. Most current neural networks are highly dependent on user-specified heuristic parameters, and performance can be highly sensitive to improper selection of such parameters. Consequently, the use of ANNs with complex applications is either limited or unsuccessful.
3. Physics-based motion prediction is a promising approach for providing useful information about many tasks under various conditions. The approach, however, is still computationally demanding; it takes time (minutes to hours) to predict the motion, even with small changes in the task conditions. That is because the approach involves solving an optimization problem that has hundreds of design variables with thousands of constraints to be satisfied. The problem's design variables represent the motion task outputs, which are the values of the full-body DOFs at different time frames of that task. The constraints represent the range of motion (ROM) for all models' DOFs, torque limits, etc. Therefore, we need to find a model that is able to provide real-time motion prediction as the result of any changes the user may impose.

4. Incorporation of the large number of constraints in motion prediction tasks is challenging for building a model with real-time results. This challenge is especially crucial for ANN design because ANN does not have a direct method to implement constraints within its design.
5. Although ANNs are used extensively, the practical significance of the network parameters is rarely studied in detail and promises to provide insight into the problem being simulated.

1.4. Hypothesis and research objectives

Successful implementation of ANNs in DHM motion problems will open new areas for using the ANNs in DHM applications, which is still an active research topic with many issues to be resolved. Eventually, handling the complicated motion problems for instant predictions within a single ANN design might reveal inherent relationships in the predicted task that are difficult to express physics-based models. Consequently, that could lead to more advancements and capabilities in the DHM field. The RBN design can be modified to handle large-scale problems accurately with a minimal number of training cases, and with no training or memory issues. Specifically, the following hypotheses will be tested:

1. The RBN training process can be modified to provide more accurate results for large-scale problems with a reduced number of training cases. The new RBN design depends on and adds to the wealth of previous research on ANN designs to implement multiple training approaches for maximum performance.

2. The RBN parameters like the number of basis functions (i.e., neurons), basis functions' centers and widths, and output weights need to be set with minimal heuristics using robust approaches in order to optimize accuracy and computational efficiency.
3. With proper enhancement, the RBN can be used to predict human motion with increased computational speed.
4. The RBN, which is considered unconstrained optimization, can involve a method for constraint satisfaction by either imposing the constraints only within its training process, or by modifying the network output(s) to satisfy the constraints.
5. The ANN, as a basis functions- and connections-based model, can be used to extract useful feedback about the predicted task(s). Potential extracted information includes: a) the relationships between different network inputs and their corresponding outputs can determine the most effective input(s), and thus define specific input as the dominant one in changing the major specific output(s); b) extraction of the most important training case(s) from the basis functions' centers and the connection wrights values might help the task validation and development processes by using that case to validate the whole task; and c) significant biomechanics information like the key outputs (i.e., joints) in a task can be drawn and analyzed as those with the most changing values.

The presented hypotheses are tested in this thesis by developing a new RBN design that outperforms other models, especially when used for large-scale problems with fewer training cases. This new design is thoroughly tested and evaluated with basic problems before being used with more complex motion prediction problems. Next, the

design is modified to incorporate motion constraints. Finally, the biomechanics implications of the network parameters are also studied and analyzed. This work pursues the following objectives:

1. Design a new ANN to provide robust and improved performance with large-scale problems with a reduced number of training cases.
2. Develop training approaches that facilitate automation of the whole training process with involvement of minimal heuristics.
3. Use the new ANN design to provide accurate real-time prediction of motion tasks for full DHM under various task conditions.
4. Develop new methods for constraint implementation within the ANN design to improve the results. Implementing contact constraints in motion tasks is especially important because these constraints can be easily violated, even with highly accurate network results.
5. Extract useful information about the predicted tasks from ANN parameters, especially for the large-scale DHM motion problems. For instance, using the network basis-functions and their weights to evaluate the relative importance of the training cases. In addition, the ANN might help in providing significant biomechanics information and the task development and validation processes by drawing some relationships between the task key input(s) and output(s).

1.5. Overview of the Thesis

The introduction chapter provides the current state of the art in ANN designs and their main applications with a focus on those in the dynamics and DHM fields. The

chapter also includes the main aims to be achieved in this work. Chapter 2 provides detailed background about the RBN as the proposed ANN to be used and modified for use in DHM motion problems. In Chapter 3, new approaches for RBN design are provided with detailed formulations. The new design is also tested and evaluated on mathematical and practical problems. Chapter 4 presents new approaches for the successful use of the new RBN design with large-scale predictive dynamic (PD) applications. In addition, the chapter involves performance evaluation for the network design with the new approaches on two PD tasks. Chapter 5 investigates new methods for constraint implementation within the new network design. Different network models with and without the new methods are evaluated and compared against each other. In Chapter 6, simulated task analyses are provided with the main focus on connecting the new network parameters with the different parts of the simulated PD tasks. The chapter works to extract the most critical inputs, outputs, and training cases for the task and network performance from the network parameters. Finally, Chapter 7 provides summary and discussions of the thesis work, as well as ideas for potential future work.

CHAPTER II

RADIAL-BASIS NETWORK (RBN) – BACKGROUND AND ANALYSIS

This chapter presents basic knowledge about the architecture of the radial-basis neural network (RBN) in order to help highlight current deficiencies and to provide a technical platform for discussions of new methods in subsequent chapters. As outlined in Chapter 1, the proposed work is ultimately concerned with applications to human modeling. In particular, optimization-based dynamic motion prediction is classified as a regression problem, which is described in Chapter 1. Consequently, the proposed work is concerned primarily with regression problems. Along with feed-forward back-propagation (FFN), RBN is the artificial neural network (ANN) used for regression problems. As explained in Chapter 1, the FFN network has significant deficiencies that prohibit its use with human modeling. Therefore, this work focuses specifically on RBN networks. The two primary high-level benefits of RBN networks are itemized as follows:

1. The RBN can be adapted to provide highly accurate results for any complex problem. The RBN training-process solution is always a global optimum because the transfer function used in the RBN is quadratic and has one unique solution (i.e., the function is positive definite) (Bishop & Nasrabadi, 2006; Ripley, 2007). Through its local-response radial transfer functions, the RBN is a universal approximation for any nonlinear complex problem (Hartman, Keeler, & Kowalski, 1990; Hornik, 1993; Poggio & Girosi, 1990; Stinchcombe & White, 1989).
2. In addition to being relatively accurate, the training process for the RBN is computationally fast. The RBN usually has no memory or training issues when used

to predict large-scale problems in terms of the number of training cases, inputs, and outputs. This is because the RBN does not have a large number of weights to be optimized in the training process. Hence, the RBN can be successfully trained using a relatively large number (thousands) of training cases to predict multiple outputs within one network model.

Despite the benefits of RBN networks as an overarching construct, there are significant deficiencies in current RBN networks with respect to algorithm details, especially when considered for applications to human modeling. Such applications include accurately handling systems with a relatively high degree of complexity, working with an insufficient or reduced number of training cases, and working with hundreds of outputs like those in motion prediction tasks. This chapter first addresses the basic concepts of the RBN and its architecture. It then describes the RBN design with respect to computational equations, typical training methods, and approaches for setting various network parameters. The final section discusses the RBN design limitations and needs.

2.1. Radial-basis network (RBN) architecture

The structure of RBN consists of three components called "layers." These layers each entail specific inputs, transfer functions, and outputs. Figure 2.1 shows the design of an RBN for an I-dimensional input vector $\boldsymbol{x} = [x_1, x_2, \dots, x_I]$ and an N-dimensional output vector $\boldsymbol{y} = [y_1, y_2, \dots, y_N]$. The middle layer, called the *hidden layer*, consists of units called *hidden neurons*. Each hidden neuron acts as a transfer function called a *basis function*, which maps the input space R^I to the output space R^N ($\boldsymbol{x} \in R^I$ and $\boldsymbol{y} \in R^N$). The vector space defined by the set of input values is often referred to as *feature space*, where

a feature is simply a component of the input vector. Conceptually, however, a feature represents some practical aspect of the input vector. That is, different components might represent very different aspects of the problem, potentially with very different units. Various types of combinations of the basis functions form a regression curve for predicting $\boldsymbol{y}$ for the given $\boldsymbol{x}$. The specific nature of this regression curve is determined during the training process. A full description of the traditional RBN is provided in the literature (Looney, 1997; Wasserman, 1993).

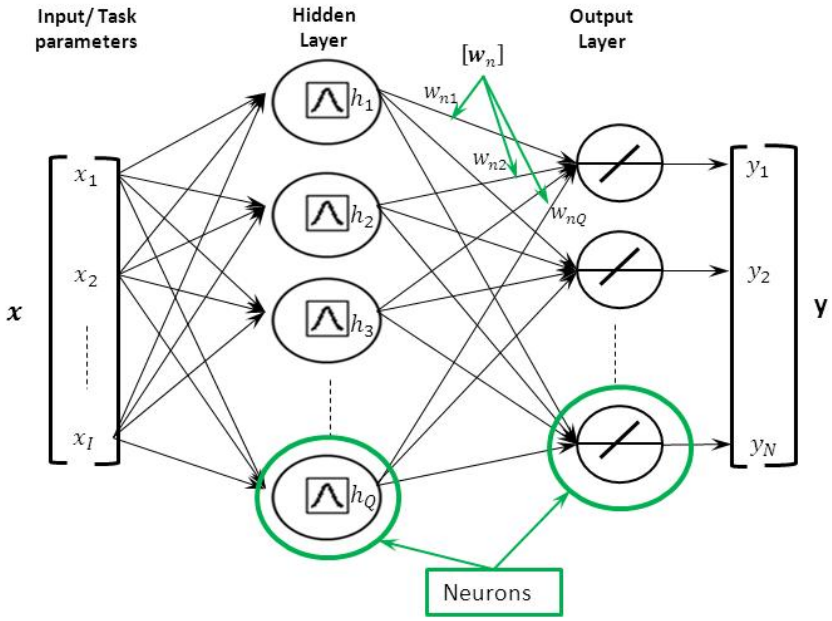

Figure 2.1: Schematic of the radial-basis neural network (RBN).

In Figure 2.1, $\boldsymbol{h} = [h_1, h_2, \dots, h_Q]$ is the vector of the outputs from Q basis functions. $\boldsymbol{w}_n = [w_{1n}, w_{2n}, \dots, w_{Qn}]^T$ is a vector of weighting factors that are determined during the training process for the n^{th} network output. There are N vectors of $\boldsymbol{w}_n$ in the matrix $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_N]$. The weights ultimately represent the relative significance

of their corresponding basis functions. Each weight scales its corresponding basis function. The core of RBN calculations occurs in the neurons of the hidden layer, where the basis functions are evaluated. Each hidden neuron is a radial basis function, usually of the Gaussian function type.

h_q is determined as follows:

$$h_q(r_q) = \exp\left[-r_q/2\sigma_q^2\right] \tag{2.1}$$

h_q is inversely proportional to the distance between $\boldsymbol{x}$ and the q^{th} basis function center ($\boldsymbol{u}_q = \left[u_{q1}\, u_{q2} \ldots u_{qI}\right]$), as shown in Equation 2.2. Its value is also proportional to the radial-basis function width σ_q (i.e., Gaussian spread).

$$r_q = \left\|\boldsymbol{x} - \boldsymbol{u}_q\right\|^2 = \sum_{i=1}^{I}(x_i - u_{qi})^2 \tag{2.2}$$

The value h_q, as a function of r_q, equals 1 at its maximum, when $\boldsymbol{x}$ equals $\boldsymbol{u}_q$, and equals zero at its minimum.

The distance r_q in Equation 2.2 is calculated as the squared Euclidean distance for $\boldsymbol{x}$ from the q^{th} basis function center ($\boldsymbol{u}_q = \left[u_{q1}\, u_{q2} \ldots u_{qR}\right]$). Figure 2.2 illustrates the relationship between r_q and the produced Gaussian-based output h_q for the q^{th} basis function. The figure also shows the contours of the h_q values, which decay as the radius r increases. The value h_q equals 1 at maximum, when $\boldsymbol{x}$ equals $\boldsymbol{u}_q$. The center $\boldsymbol{u}_q$ represents a row in the matrix $\boldsymbol{U} = \left[\boldsymbol{u}_1\, \boldsymbol{u}_2 \ldots \boldsymbol{u}_Q\right]^T$. Thus, r_q indicates the distance of the input vector $\boldsymbol{x}$ from the q^{th} basis function center. If the centers of the basis functions are selected such that the basis functions are far apart, then the corresponding r_q for any given $\boldsymbol{x}$ might be relatively small for one or multiple basis functions. The rest of the basis

functions, however, produce relatively large r_q for that $\boldsymbol{x}$, because they are set at further locations. Consequently, only the basis functions that produce the relatively small r_q contribute in the network final output.

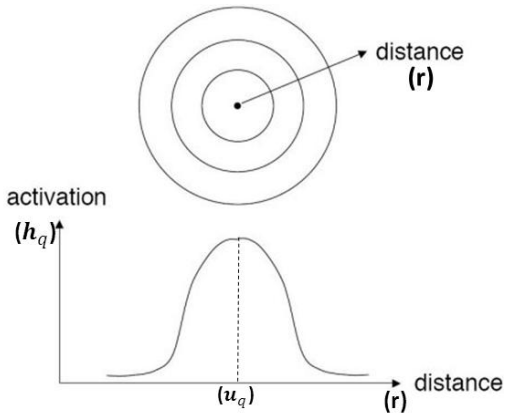

Figure 2.2: Two-dimensional plot and contours for the Gaussian function type of radial basis function.

The function width σ_q (the Gaussian spread) is the primary tuning parameter in RBN along with the number of basis functions (i.e., neurons) in the hidden layer. Together, $\boldsymbol{U}$ and the vector of Gaussian widths $\boldsymbol{\sigma}$ define the basic shape of the response surface that is determined by training the neural network.

An illustrative example for the basis function centers is shown in Figure 2.3. The network in this example has a two-dimensional feature space, x_1 and x_2. The radial functions $\boldsymbol{h}$ are circles with different sizes that decay from their centers $\boldsymbol{U}$ (i.e., contours of larger values when closer to the center). The circles' sizes of a basis function (h_q) are defined by σ_q. Each h_q , with a center at $\boldsymbol{u}_q$, has a maximum value of 1 and almost zero at its largest contour.

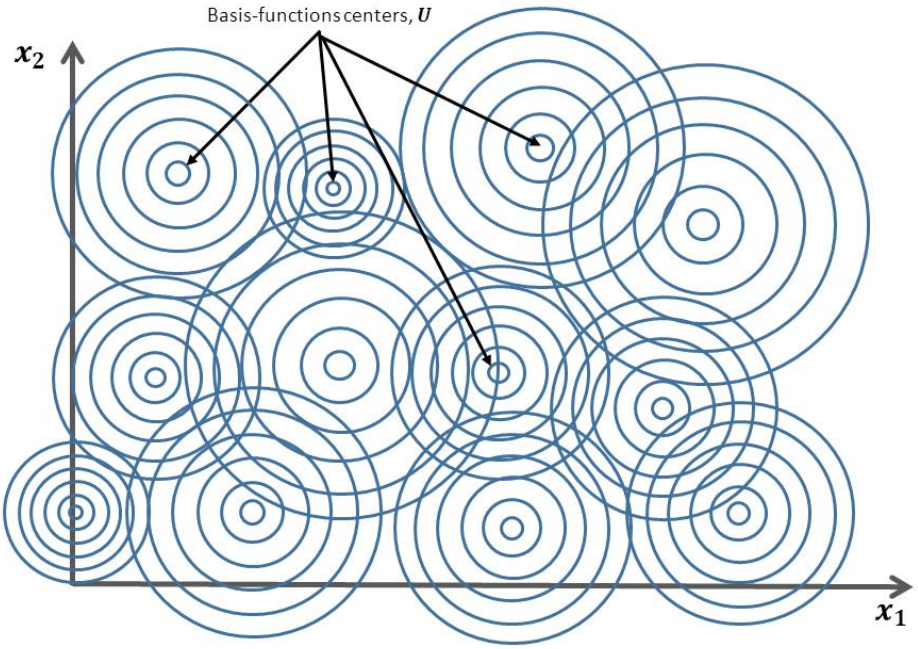

Figure 2.3: Portion of network radial-basis functions curves in two-dimensional feature space (x_1 and x_2 are the inputs).

In the network, when all basis functions outputs ($\boldsymbol{h} = [h_1\ h_2\ \ldots\ldots h_Q]$) are calculated based on Equation 2.1, $\boldsymbol{h}$ enters each one of the output neurons to calculate the final network outputs $\boldsymbol{y}$. Figure 2.4 shows a portion of the RBN, shown in Figure 2.1, that corresponds to the calculations in the n^{th} output neuron to produce the final n^{th} output (y_n), which is calculated in Equation 2.3. The output y_n is a weighted sum produced by multiplying the elements of $\boldsymbol{h}$ by the elements of $\boldsymbol{w}_n$. As shown in the figure, the output neuron has a linear transfer function, as indicated in Chapter 1 for the case of a regression problem. When calculating the *N* outputs of the full network shown in Figure 2.1, there are *N* weight vectors, each corresponding to its associated output.

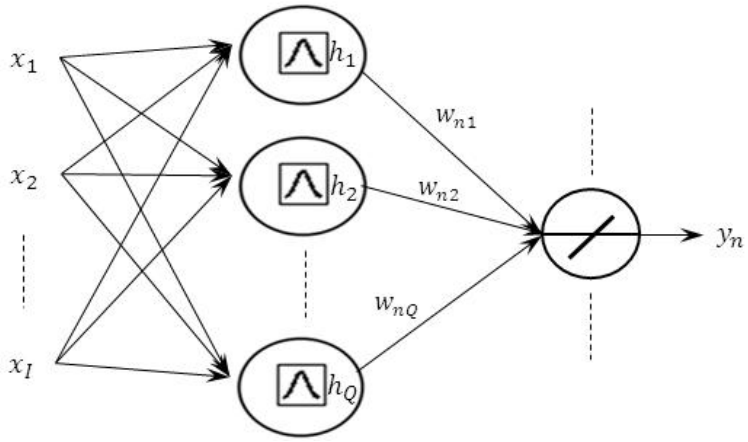

Figure 2.4: Portion of neural network corresponding to the n^{th} output within multi-outputs radial-basis network (RBN).

$$y_n = \boldsymbol{h} \cdot \boldsymbol{w}_n = \sum_{q=1}^{Q} h_q * w_{qn} \tag{2.3}$$

Given the value of $\boldsymbol{h}$, the calculation of the n^{th} output y_n is provided in Equation 2.3. When calculating the *N* outputs of the full network shown in Figure 2.1, there are *N* weight vectors, each corresponding to a single associated output.

In regression problems, when multiple Gaussian functions are used and combined as the network basis functions, the produced network function conceptually acts as a curve (see Figure 2.5). The network output curve, which is represented by y_n in Equation 2.3, results from a weighted combination of different Gaussian functions at different space locations. Figure 2.5 shows an example for output curve y_n produced from three Gaussian functions ($h_1, h_2,$ and h_3). Each function has its own weight w_{qn}, in Equation 2.3, to change the peak value of the function.

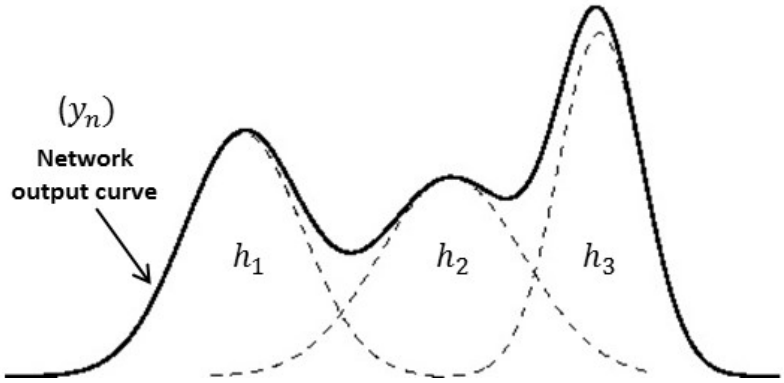

Figure 2.5: RBN output curve resulting from multiple Gaussian functions.

Different radial-basis functions can be used, including the Gaussian function, multi-quadric function, generalized multi-quadric function, inverse multi-quadric function, generalized inverse multi-quadric function, thin plate spline function, and cubic function (Buhmann, 2003; Press, 2007). Choosing the type of radial function is not a critical issue because various functions provide comparable practical results on network performance. One study (Powell, 1987) refers to the Gaussian function and the thin plate spline function as typical options. Others (Moody & Darken, 1989; Poggio & Girosi, 1990) also specified the Gaussian function as the most commonly used radial basis function when solving regression problems. In addition, the Gaussian function is a localized function, meaning its output monotonically decreases with distance from its center (i.e., $h(x) \rightarrow 0$ at $\|\boldsymbol{x} - \boldsymbol{u}_q\| \rightarrow \infty$), while all the other functions, except for the inverse multi-quadric function, monotonically increase where h(x)$\rightarrow \infty$ at $\|\boldsymbol{x} - \boldsymbol{u}_q\| \rightarrow \infty$. Therefore, the Gaussian function provides a reasonable, globally stable, and fast method for interpolation with tracking errors converging to a neighborhood of zero (Sanner & Slotine, 1992). In other words, it is a bounded function that provides output

between 0 and 1, which produces a stable response. Hence, this work will consider the Gaussian function as the typical basis function in all proposed approaches.

In terms of the number of hidden layers, it is better to use one layer and work on changing the number of basis functions (neurons) and/or training data sets until the best performance is achieved. That is because the literature indicates that one hidden layer is capable of handling any practical complex problem (Bishop & Nasrabadi, 2006; Wasserman, 1993). More complex problems need more neurons in the hidden layer and, rarely, more hidden layers. There is currently no theoretical reason to use neural networks with more than one hidden layer. The reasons for not using more than one hidden layer are as follows:

1. Adding more hidden layers makes the network performance unstable and subject to more noise because there are more neurons and connections between the layers. In addition, the resulting model becomes more complex and specialized for the training cases (i.e., more potential for an over-fitting problem). That, in turn, reduces the general network prediction ability for new unseen inputs (i.e., inputs other than the training cases).
2. There is more potential to reach a local optimization solution when the network has two or more hidden layers because there are more connections to be found at different layers. Consequently, the locally optimized network has poor performance.

Given the structure described thus far, the network must be trained in order to determine the following parameters while maximizing the accuracy of the network: 1) the number of basis functions Q, 2) the basis function centers $\boldsymbol{U}$, 3) the basis function widths $\boldsymbol{\sigma} = [\sigma_1\ \sigma_2 \ldots \sigma_Q]$, and 4) the output weight matrix $\boldsymbol{W}$.

2.2. Training techniques in radial-basis network (RBN)

In this section, typical network training processes are presented for setting the values of different network components. The processes are called the *fast-training method* and the *full-training method*. After the discussion of these processes, methods to improve the network generalization during the training process are illustrated. Then, techniques for setting the network parameters are provided in subsections under each parameter's name.

In the training process, known sets of input(s) ($\boldsymbol{x}$) and their corresponding output(s) ($\boldsymbol{t} = [t_1, t_2, \dots, t_M]$) for M training cases are used to set and estimate the network parameters. In the typical training process, all network parameters are either set by optimization or are predefined heuristically. Typically, there are two types of training processes for designing an RBN. The first method has $\boldsymbol{w}$ parameters as the design variables. The second method has $\boldsymbol{w}$, $\boldsymbol{U}$, and $\boldsymbol{\sigma}$ as the design variables. Each one has its own advantages and drawbacks. The number of basis functions Q in both methods is set by the user.

2.2.1. Fast training method

The first type of training process, called the *fast training method*, finds the connection weight ($\boldsymbol{w}$) values (Equation 2.4) (Broomhead & Lowe, 1988; Wasserman, 1993). After the values of $\boldsymbol{w}$ are initially assigned to random small values, usually between -1 and 1, the optimization in Equation 2.4 is solved by adjusting the values of $\boldsymbol{w}$ until the sum-squared error (SSE) reaches the minimum. The optimization has been proven to always find a global solution because of its quadratic-based basis functions

(Broomhead & Lowe, 1988; Looney, 1997). This problem is typically solved using the gradient-based numerical optimization methods because the quadratic cost function (i.e., the error) is at the minimum where the gradient is zero. Given the significance of the initial guess with any gradient-based optimization method, improper selection of initial values for $\boldsymbol{w}$ can result in relatively slow convergence.

$$\text{Minimize: } f(\boldsymbol{w}) = \sum_{m=1}^{M}(t_m - y_m)^2 \qquad (2.4)$$

$$= \sum_{m=1}^{M}(t_m - (\sum_{q=1}^{Q} h_{mq} * w_q^T))^2$$

In Equation 2.4, t_m represents the true output value for the m^{th} training case. The term *true output* is used to distinguish the output provided by preconceived training data from output calculated by a complete and trained neural network. y_m represents the predicted output from the network for the m^{th} training case. Thus, h_{mq} is the q^{th} basis function output for the m^{th} training case. w_q is the weight value for the q^{th} basis function.

For network parameters $\boldsymbol{U}$ and $\boldsymbol{\sigma}$, the user usually defines them heuristically prior to solving Equation 2.4. Basically, the centers in $\boldsymbol{U}$ are set to the inputs of some randomly selected training cases, and the basis functions widths ($\boldsymbol{\sigma}$) (i.e., Gaussian spreads) are heuristically set to a small value (e.g., 0.1). This training method sometimes works well for applications with a relatively small number of training cases because the network could be designed to have all training cases as centers in its design (i.e., $\boldsymbol{u}_q = \boldsymbol{x}_q$). When a large number of training cases exists (thousands of cases or more), the user heuristically sets the centers $\boldsymbol{U}$ that are drawn randomly from a subset of the input training set (Looney, 1997; Wasserman, 1993).

On the other hand, the method generally experiences poor performance due to the heuristic or random selection of the network parameters. Certainly, any algorithm with heuristic parameters offers opportunities for improved effectiveness, if such parameters are optimized in some sense. There is always a problem with the accuracy obtained by random selection of $\boldsymbol{U}$ because the randomly selected centers do not provide enough distribution to cover the whole training space in order to allow proper network performance. In addition, two or more training cases that are overlapping (i.e., located close to each other) might be selected, which will not produce a model with small error. That is because some inputs will be too close to two or more centers at the same time, and the corresponding outputs will be distorted. In other words, some of the selected cases are linearly dependent, and all of them can be replaced by one case. The problem with this condition is also called the *ill-conditioned problem*. In addition, the random selection of the basis functions' centers can inadvertently exclude significant training cases that cover part of the spanned space. The method of selecting the centers of $\boldsymbol{U}$ to equal the inputs of all training cases (i.e., $Q=M$) is not necessarily an effective alternative because it produces an over-fitting issue, which will be discussed in a later subsection. The resulting network predicts the training (on-grid) cases accurately but fails to predict off-grid points with proper accuracy. Furthermore, this approach results in ill-conditioned problems. In summary, the fast-training method has the advantage of computational speed, especially when there is a small number of training cases, but new methods are needed for defining network parameters more effectively.

2.2.2. Full-training method

The second method, called the *full-training* method, finds all network parameters $\boldsymbol{w}$, $\boldsymbol{U}$, and $\boldsymbol{\sigma}$ that minimize the sum-squared error (SSE) (Equation 2.5). Thus, fewer parameters are set heuristically or randomly, as compared to the fast-training method. The method uses the same procedures and optimization formulation as the first training method. All design variables are initially set to random values before optimization is run. The problem is formulated as follows:

$$\text{Minimize: } f(\boldsymbol{w}, \boldsymbol{U}, \boldsymbol{\sigma}) = \sum_{m=1}^{M}(t_m - y_m)^2 \qquad (2.5)$$

$$= \sum_{m=1}^{M}(t_m - (\sum_{q=1}^{Q} h_{mq} * w_q^T))^2$$

Despite the deterministic benefits, this method yields a network that often requires an excessive number of basis functions for proper accuracy. This is due to the random selection of the number of basis functions and their centers, which can result in centers that are located outside the training space or that cover a partial area in the training space. Hence, more basis functions are required to achieve proper training accuracy. This in turn makes the network run slower after the training is completed. In the case of applications with a relatively large number of training cases, a large number of outputs, or both together, the method also experiences training issues and is computationally costly due to the large number of design variables. Consequently, the use of this method in this work for prediction of large-scale problems like the digital human modeling (DHM) motion is not applicable. We could, however, use the fast-training method because of its advantages for applications with relatively small numbers of training cases and its cheap computations. On the other hand, novel approaches will be added to that design in this

work, especially the randomly assigned parameters. All such approaches will be discussed in Chapter 3.

In addition to the aforementioned limitations, both traditional training methods often have over-fitting or under-fitting issues in many applications. Basically, both approaches can be significantly inaccurate. This is because the number of basis functions is heuristically chosen, which leads to more or fewer basis functions than the network needs to predict a problem accurately. A detailed description of the desired network for optimal design, as well as the under- and over-fitting issues, is discussed in the next section, which provides the crux of the proposed new algorithms.

2.2.3. Network generalization

A third approach to network training involves improving the network *generalization*. Generalization is the network's ability to provide proper prediction for new cases that have never been used to train the network. The typical training methods produce models that have poor accuracy due to the use of SSE as the cost function in the optimization problem. If the SSE is minimized for the training cases without monitoring the network performance when predicting new test cases, the resulting trained network will have what is called *poor generalization*. The test cases are cases that are not used in the training process. The network is properly generalized if it predicts the test cases with accepted accuracy. The accepted accuracy value is determined based on the application or on the performance of the network relative to that of other prediction models. The network prediction's accuracy is measured by calculating the error when it predicts the

test cases. This is called the *test erro*r. Acceptable generalization is produced when the test error is minimal (i.e., acceptable for the predicted application).

Figure 2.6 shows a conceptual illustration of three simple network regression curves produced by minimizing the SSE of the training points shown in the figure. Curve (b) represents the optimal one with good generalization; the curve passes smoothly through the training cases with small error. On the other hand, curve (c) presents a network prediction curve with poor generalization. The curve in the Figure 2.6 passes through most of the training points. Consequently, it has minimal on-grid training error, but has spikes and sharp transitions between the training points (large off-grid test error) that result in reduced accuracy. This case is called *over-fitting*, which indicates that the network uses a more complex curve (i.e., higher-order polynomial) than it should. Poor performance is also seen with curve (a), where the regression curve is simpler than needed for the problem. This case is called *under-fitting* because the produced polynomial is below the order required to fit the problem training data (i.e., the system has high bias). This issue of under-fitting happens because either the used model is simple or the available training data are inadequate.

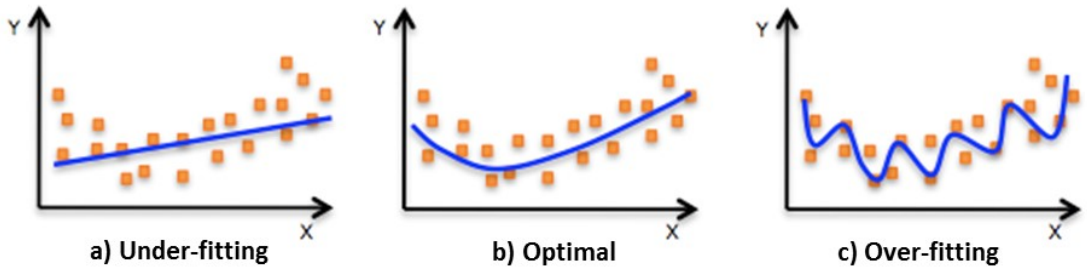

Figure 2.6: Example for fitting training data using three curves that represent cases of a) under-fitting, b) optimal fitting, and c) over-fitting.

Optimal design is difficult to find in many practical applications for one or both of the following reasons: 1) there is an insufficient number of training cases to provide enough insight about the underlying structure of the predicted output, and 2) the prediction model has poorly defined parameters, leading to poor performance. Some applications have a reduced number of training cases that is not enough to be appropriately predicted by typical prediction models. Consequently, researchers have been trying to find the best methods to minimize testing error produced from models like ANN to improve the generalization in application prediction under any given conditions. Hence, many methods have been introduced to reach that goal by improving ANN training processes and/or selecting their parameters. The methods that are concerned with the training optimization problem are summarized in this section, while the methods concerned with the network parameters are presented in the next subsections under different categories.

Well-known methods for improving the network generalization by minimizing the approximated network testing error include: 1) early stopping, 2) cross-validation, and 3) regularization. All methods target reduction of over-fitting in the training process by preventing the SSE from reaching zero, as well as providing the minimum possible estimate for the general test error for the used model by calculating the error produced from the prediction of a few test cases. The early stopping method divides the training data into two parts. One large training part usually includes 60-90% of the original training cases and is used to train the network. The other small test part (also called the *tuning part*) is used to test the network performance during the training process. The training process, using either Equation 2.4 or 2.5, is performed many times, each with

different values for the network parameters (i.e., different $\boldsymbol{U}$ and $\boldsymbol{\sigma}$) and/or numbers of basis functions. Each time the network is created, it is tested using the cases in the tuning part (i.e., the tuning error is calculated). The training process stops when the tuning error does not change, starts increasing, or has only slight changes over successive iterations.

With the cross-validation method, the training data is divided into small portions, usually 5 or 10 groups of equally sized data points. All except one portion are used as training cases, and the remaining portion is used as tuning to evaluate the test error. With the same network parameters setting, every group of data is used once as tuning data while the remaining groups are used to train the network. As in the early stopping method, the training process is repeated many times, each time with different network parameters and/or numbers of basis functions. The selected network is the one with smallest average error over the errors produced when using each group as tuning data. Cross-validation differs from early stopping in that all training points are eventually used in the training and tuning process, while the early stopping method assigns a specific portion of the training cases to always be used to test and tune the network. More details about the early stopping and cross-validation methods are provided in the literature (Duda, Hart, & Stork, 2012).The two methods share some limitations, which include the following: 1) the training process sometimes terminates prematurely with poor network design because the optimization is terminated after the tuning error increases or becomes stuck at some apparently minimum value (i.e., local solution), 2) the methods usually do not provide proper results with applications where limited training data are available because splitting the data to have the tuning part could produce poor trained model with the remaining portion of the data, and 3) the selected portion of tuning cases might not be

a good representation of the actual testing error (the network might be adapted for very small tuning error but provide large error for the actual test cases).

A third method involves adding a regularization part to the objective function (Bishop & Nasrabadi, 2006; Haykin, Haykin, Haykin, & Haykin, 2009), which is also called the *weight decay*. Connection weights $\boldsymbol{w}$ are added to the cost function, as shown in Equation 2.6, where the optimization becomes multi-objective optimization. Bartlett (1998) confirms that the smaller the weight's $\boldsymbol{w}$ values are, the better generalization performance the network tends to have. Therefore, their values are controlled and monitored by adding them as a penalty to the cost function. The resulting network is still flexible enough to handle the predicted problem with relatively small training and testing errors.

$$\text{Minimize: } f(\boldsymbol{w}, \boldsymbol{U}, \boldsymbol{\sigma}) = \sum_{\mathrm{m}=0}^{M}(t_m - y_m)^2 + \frac{1}{2}\sum_{q=1}^{Q}(w_q)^2 \qquad (2.6)$$

This *weight decay* approach, however, sometimes fails to solve the over-fitting problem because the regularization (penalty) part in the cost function does not have enough weight to largely affect the whole cost function in the optimization. As a result, the weights $\boldsymbol{w}$ stay large in the final optimal design. Moreover, the network performance could be poor because of other issues like under-fitting, and thus adding the regularization part has no benefits. Consequently, this work will develop other methods for designing new RBN with optimal performance, and with minimal over-fitting and under-fitting issues. This design will be discussed in Chapter 3.

While the three formulations discussed above address overall approaches to training networks, there are a variety of specific methods for setting method parameters, $\boldsymbol{U}$ and $\boldsymbol{\sigma}$, and these techniques also play a critical role in designing networks.

2.2.4. Techniques for setting network parameters

Despite the drawbacks of the aforementioned training processes, some scholars have introduced methods for finding the critical network parameters. These parameters include the basis functions centers ($\boldsymbol{U}$) and Gaussian widths ($\boldsymbol{\sigma}$). It is important that both parameters be selected carefully for better network performance. Therefore, this section provides a summary of the main techniques introduced in the literature for setting these parameters.

2.2.4.1 Basis functions centers

Besides the methods used for setting $\boldsymbol{U}$ in the aforementioned typical training processes, recent methods have been proposed to find the best possible values of $\boldsymbol{U}$ depending on its physical meaning in the network design. The basis functions centers, which are the rows in matrix ($\boldsymbol{U} = \left[\boldsymbol{u}_1\ \boldsymbol{u}_2 \ldots \boldsymbol{u}_Q\right]^T$), are the locations of the centers of basis functions. Distribution and selection of these centers is crucial in order to cover the whole training space, and in order to provide adequate accuracy. The main methods are summarized as follows:

1. The density estimation method, which sets $\boldsymbol{U}$ in the training space at locations with high densities (i.e., places with more concentrated training cases). Some drawbacks include: a) the selection of centers is not guided by measuring the training SSE, which can lead to large test error, and b) the resulting centers are representative of the feature space density, but do not guarantee the ability to capture the structure that carries discriminatory information for the predicted problem. In other words, the method can leave important and unique points unselected because of their low

densities. More details about this method are provided in the literature (Haykin et al., 2009).

2. The k-means clustering algorithm, which involves clustering the training data into groups. The number of *k* clusters is determined in advance by the user, and the algorithm clusters the similar training points (cases) based on the distances from each point to the cluster centers. Then, the produced clusters are set as ***U***. The k-means method experiences drawbacks similar to those of the density estimation method. More details about this topic are provided in the literature (Haykin et al., 2009).
3. The orthogonal least square (OLS) algorithm, which involves selection of the training cases that contribute the most in the network-predicted output as basis functions centers. The algorithm keeps adding basis functions one at a time until the termination criterion is satisfied. Its algorithm terminates when the sum of the contributions from all added basis functions is close to 1. The method is considered a fully supervised method because it tracks the error reduction in the produced model. It avoids the problems of the first two methods. The OLS method is considered a full-training method that eventually finds ***w*** after ***U*** is set. More details about this topic are provided in the literature (Chen, Billings, & Luo, 1989).

From these methods for setting ***U***, there is an opportunity for the OLS method to be included with some modifications in a new training approach. The advantages of the OLS method allow for successful incorporation in a more robust and generalized RBN design. On the other hand, more innovative procedures for the OLS method termination criterion are needed. That is because the method was originally introduced for applications with large numbers of training cases, on the scale of thousands or hundreds

of thousands, such as signal processing problems. The new approach, which is introduced in Chapter 3, employs the OLS with necessary modifications to facilitate more generalized performance for any application, especially those with limited numbers of training cases.

2.2.4.2 Basis function spreads

The radial-basis function width, σ, (or Gaussian spread) is a property of the basis function that determines the area covered by the function. Usually, the $\boldsymbol{\sigma}$ values are found by optimization, as in the full-training method in Section 2.2.2, or set heuristically to a fixed small number (i.e., 0.1) or using the root-mean square distance (RMSD) between the centers of the basis functions. The values of $\boldsymbol{\sigma}$ represent the function variance, which is essentially a measure of how disperse a set of numbers is. The larger the $\boldsymbol{\sigma}$, the more scattered the function. For example, Figure 2.7 shows three Gaussian functions with different $\boldsymbol{\sigma}$ values. When the function has larger σ, as with the red-colored function, the function will cover more area in the space. That in turn reduces the existence of empty spaces. That is, the function will still provide output at a distance further from its center before it vanishes. On the other hand, such a function could produce less accurate results for some inputs because it covers a large area that could lead to extreme overlapping with other basis function(s) at those inputs. A basis function with smaller σ , like the green- and blue-colored functions in Figure 2.7, provides highly accurate results within ranges close to its center because it covers a small area in the training space where it only responds to inputs located at a distance close to its center. Using such function(s), however, leaves the training space with more uncovered areas, which could produce zero outputs for the inputs that are located at these uncovered areas.

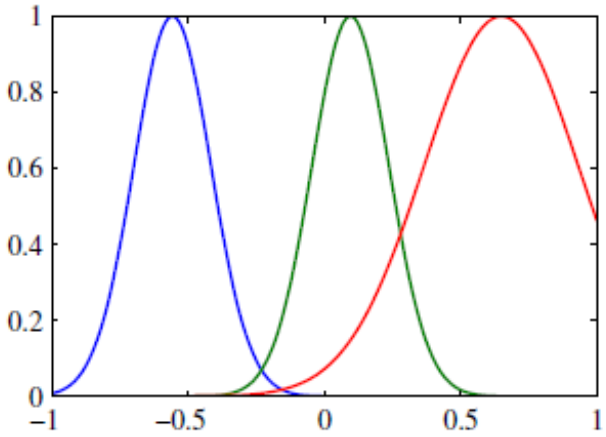

Figure 2.7: Three Gaussian functions, where the red-colored one has larger width σ than the other two functions.

Ideally, when an input is located at a point within one basis function, that basis function is the only one fired (activated), producing the output corresponding to the received input. Practically, many inputs are located in the space at points where two or more basis functions are fired, like the areas in Figure 2.7 with two or more overlap functions. Overlapping between neighboring basis functions is necessary to cover all the training space. That is especially needed when the problem has multi-dimensional inputs, which is the typical case in most problems. In general, the width σ should be neither too large nor too small for the network to cover all the training space while achieving the most appropriate accuracy. In addition, using the same σ value for all basis functions cannot produce highly accurate and appropriate models because each basis function deals with a different segment (space size) of the training space in terms of both the received input and the predicted output.

Many software packages and methods for RBNs use a fixed value for all $\boldsymbol{\sigma}$ in their network designs. Reasons include, but are not limited to: 1) they ignore the

importance of having different values for σ at different basis functions and depend only on setting up other network parameters in designing the prediction model, and 2) it is costly in terms of time to include $\boldsymbol{\sigma}$ in optimization. Therefore, most of these methods leave $\boldsymbol{\sigma}$ as an option for the user to define heuristically. Some researchers also claim that finding $\boldsymbol{\sigma}$ by optimization is not helpful for the training process because: 1) the statistical meaning of $\boldsymbol{\sigma}$ is known and set in more robust techniques like root-mean square distance (RMSD) (Saha & Keeler, 1990; Wasserman, 1993), and 2) the issue of over-fitting is more likely to exist if both $\boldsymbol{\sigma}$ and $\boldsymbol{w}$ are optimized. Consequently, we can incorporate mixed approaches in setting the values of $\boldsymbol{\sigma}$ by setting them to an initial design that depends on heuristics techniques like RMSD. Then, their optimal values are found by optimization. More details for the proposed mixed approach are provided in Chapter 3.

2.3. Discussion

The RBN that is used in this work and its various training techniques are illustrated in this chapter. The significance of various RBN elements and techniques in setting their values are also highlighted. Although RBN is selected in this work for the reasons itemized in the introduction of this chapter, it has deficiencies that open opportunities for new and more robust and generalized design, especially for use in DHM applications. These deficiencies are summarized as follows:

1. The fast-training method experiences poor performance due to the heuristic and random selection of the network parameters. The use of the full-training method to predict large-scale problems like the DHM motion is also limited due to the large number of parameters and generalization issues in that method. Therefore, the

opportunity is to use a modified optimization approach within a new RBN training process to overcome the limitations of both training approaches. The randomness in assigning the network's parameters is especially targeted in the new design, which is presented in Chapter 3.

2. Except for the regularization method, none of the methods proposed to improve the network generalization can be applied efficiently in applications with a reduced number of training cases like those in the DHM. On the other hand, the regularization method capability is not guaranteed in many applications. Thus, it is essential to determine alternate methods for designing a new RBN with optimal performance and minimal over-fitting issue, especially for applications with minimal training cases. Such a design will be illustrated in Chapter 3.
3. The typical training approaches define the basis functions' centers ($\boldsymbol{U}$) heuristically. Among the recent methods that have been proposed to find the values of $\boldsymbol{U}$, the OLS method has the most advantages. It is a full-training method that finds not just $\boldsymbol{U}$, but also the number of basis functions and output weights ($\boldsymbol{w}$). The method was originally introduced for applications with a large number of training cases, on the scale of thousands or hundreds of thousands, such as signal processing problems. Therefore, the new RBN design can include the OLS method with some modifications to its termination criterion and its training setups to work properly for the intended applications. Hence, the new design should work for all applications, even those with a limited number of training cases.
4. The heuristic setup of the basis functions' widths ($\boldsymbol{\sigma}$) in the currently available training methods and software packages limits the network's flexibility to produce

highly accurate results in some applications. That requirement especially applies to the applications we address in the course of this work. Based on the presented method for setting $\boldsymbol{\sigma}$, we can incorporate mixed approaches to assign their values within a multiple-stage-based training process. The approaches include the heuristic-based RMSD approach to initiate the values of $\boldsymbol{\sigma}$. Then, the optimal values are found in later stages by optimization.

5. The RBN cannot predict points that are out of the training space. The network cannot provide accurate outputs when the input is outside the range of training data (i.e., no extrapolation). This limitation, however, does not affect the targeted DHM applications of this work because the extrapolation capability is not needed. Applications like motion prediction involve known input limits, beyond which the application either does not work or is invalid.

All proposed opportunities for the aforementioned deficiencies are considered in a new methodology discussed in Chapter 3. The new methodology uses minimal heuristics to allow for automation of the whole training process. Certainly, the new design efficiency is achieved by offering methodologies for setting all necessary network parameters that are task independent (i.e., the network parameters that are not task related) to their optimal values. Heuristic approaches (or rule of thumb) are then used to set other task-relative network parameters. With the new training algorithms, which will be detailed in Chapter 3, the new RBN design can outperform other designs when applied on applications with a reduced number of necessary training cases. Moreover, the high performance of the new design is critical because it is applied to DHM motion prediction, which requires prediction of hundreds of outputs from the same model.

CHAPTER III

NEW DESIGN FOR RADIAL-BASIS NETWORK WITH REDUCED TRAINING SETS

3.1. Introduction

This chapter presents new methodologies for designing a radial-basis network (RBN) with an optimal training process to improve the network performance (i.e., the most possible accuracy in the predicted network outputs) for different applications. The improved performance is especially applicable for those applications with reduced available training data. Ultimately, this work points towards the use of artificial neural networks (ANNs) for modeling dynamic human motion (Xiang et al., 2010), which constitutes a regression problem with respect to neural networks. However, predicting dynamic motion is a complex problem and can be computationally demanding. Furthermore, simulations can be sensitive to changes in task parameters that extend beyond anticipated bounds. This in turn presents a case where accumulating a large number of data (training cases from which to learn) can be difficult. Thus, there is a need for a neural network that can accurately model complex problems with minimal data. To be sure, predicting human motion provides the motivation for this work. Nonetheless, in keeping with the analogy of mimicking cognitive performance, there is a distinct need for an ANN with improved performance, and responding to this need yields a tool with potential application to a broad range of problems.

Fundamentally, ANNs provide a means of modeling large sets of data. Conceptually, they provide a computational representation of how one takes in and

processes data—of how one learns. Consequently, potential applications are extensive. However, as with learning in general, there are many facets to neural networks. There are many ingredients, so to speak. Often, the challenge is not simply applying a network to a particular problem, but determining the most appropriate components for a particular problem in order to maximize computational speed and accuracy. In general, depending on the type of ANNs used to predict a problem, one or multiple network parameters are set up heuristically. Thus, the performance of the resulting trained network is vulnerable.

The work in this chapter steps through the process of designing a new ANN specifically for problems with limited training data and with potential application to digital human modeling. We contend that if selected properly, existing methods for setting the network parameters can be integrated to provide a novel network that outperforms current approaches with respect to computational speed and accuracy (for a set number of training cases). Based on the RBN training process, the new design involves multi-stage training techniques for automatically determining all network parameters that are not problem specific.

This chapter presents the following specific contributions: 1) improved RBN performance when predicting a task with any number of training cases, 2) a modified orthogonal least squares (OLS) algorithm for determining basis function centers and initial weighting factors, 3) integration of the OLS approach and an optimization-based approach, and 4) automatic objective calculation of all network parameters. Note that the OLS method is usually presented as an independent training process to replace the typical RBN training process. Alternatively, we propose using a modified OLS method as one

component within the context of an overarching methodology for problems with minimal training sets.

The rest of this chapter is arranged so the methods used in the new training process are presented first. Then, experimental examples are illustrated to test the validity of the new methodologies. Finally, discussions and conclusions for this chapter are presented.

3.2. Method

The new RBN design involves a multi-stage training process for determining necessary network parameters. This process involves four main components, as presented in Figure 3.1, after training sets of inputs ($\boldsymbol{x}$) and their corresponding outputs ($\boldsymbol{t}$) are provided. First, the network training inputs $\boldsymbol{x}$ are normalized using the standardization approach. Secondly, preliminary values for the Gaussian spreads $\boldsymbol{\sigma}^o$ are set for all basis functions using root mean square distance (RMSD). Then, the basis functions and their centers $\boldsymbol{U}$ are automatically set using a new OLS method with a modified convergence criterion. The preliminary values for connection weights $\boldsymbol{W}^o$ are also calculated from the OLS method. Finally, optimal values for $\boldsymbol{W}$ and $\boldsymbol{\sigma}$ are calculated by minimizing the sum square error (SSE) over all training cases.

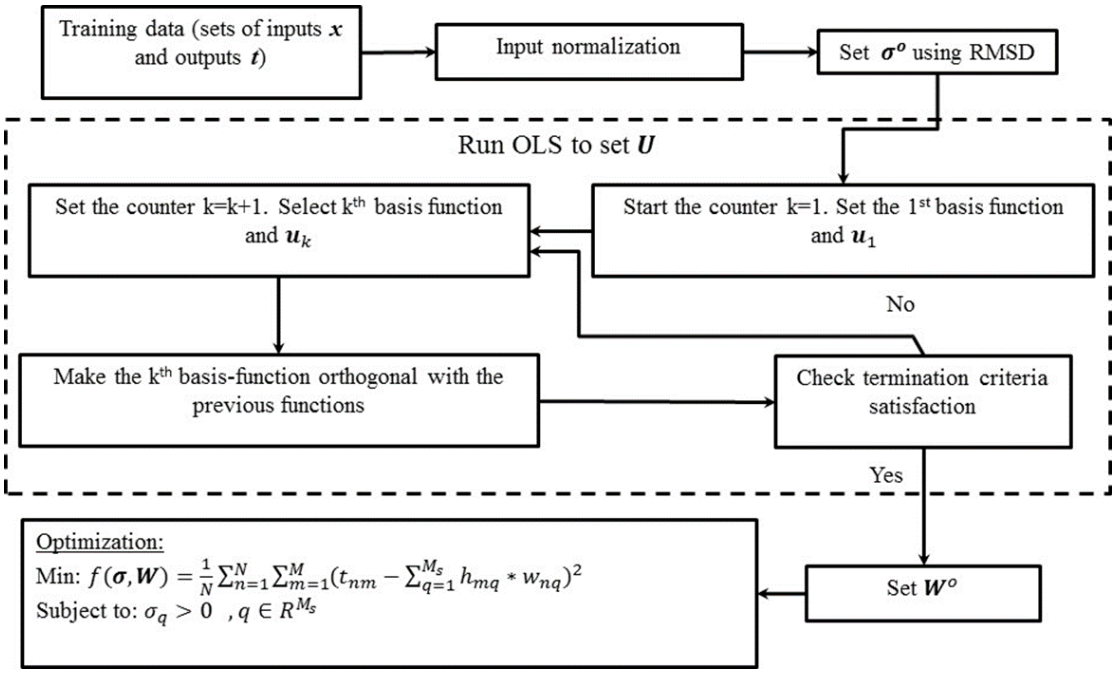

Figure 3.1: Flow chart for the steps of the new training process of the RBN design.

The transformation of inputs, such that they all have similar ranges, helps to set proper preliminary values for $\boldsymbol{\sigma}^o$ based on the RMSD method. Then, unlike the typical OLS methods, all available training cases are traversed in order to select the ones that have the most significant contributions in the outputs as the centers $\boldsymbol{U}$ of the network basis functions. This in turn implicitly sets the number of basis functions as well as a preliminary output weight matrix $\boldsymbol{W}^o$. Furthermore, a novel *double-termination* criterion for the OLS method is introduced to facilitate more robust use with varied applications. The criterion guarantees the proper *complexity* for optimum computational speed and accuracy, where a network's complexity is proportional to the number of basis functions. Unlike the randomly assigned initial guess in traditional training methods, the values for $\boldsymbol{\sigma}^o$ and $\boldsymbol{W}^o$ are determined from RMSD and OLS, respectively, and are used as the initial guess with the optimization process in order to improve convergence. $\boldsymbol{U}$ is excluded as a

design variable from the optimization, contrary to the traditional approach (Looney, 1997), because the OLS method is designed primarily to provide an adequate final $\boldsymbol{U}$. This integration of OLS and RMSD, with an optimization-based approach, is the primary novelty of the proposed approach and has proven to be effective, as will be demonstrated with a series of empirical and real-world problems.

3.2.1. Normalizing the input

As a preprocessing step, normalization is performed to decrease the relative variance of each element in the input $\boldsymbol{x}$. In general, the normalization should be performed in the ANN in both the training and testing phases for the following reasons:

1. Compressing inputs by normalization changes them all to be in a consistent range. This step minimizes the bias within the ANN for one input feature over another (Priddy & Keller, 2005) when calculating the RMSD (in Section 3.2.2). In the context of the RBN, the basis functions are locally activated (i.e., each basis function is activated to provide output only when $\boldsymbol{x}$ is located in specific region of the training space where the function is located). Outputs from the basis functions are calculated based on the distances between all inputs and their corresponding components in the basis function center (Equation 3.1). Therefore, various types of inputs should have similar dimensions, so that they all have the same importance in distance calculations.

$$h_q = \exp\left[-\frac{\sum_{i=1}^{I}(x_i - u_{qi})^2}{2{\sigma_q}^2}\right] \tag{3.1}$$

where: h_q: the output of the q^{th} basis function.

$\boldsymbol{x}$: vector that represents the network inputs ($\boldsymbol{x} = [x_1, x_2, \dots, x_I]$).

$\boldsymbol{u}_q$: vector that represents the center of the q^{th} basis function ($\boldsymbol{u}_q = [u_{q1}\ u_{q2}\ \ u_{q3}\ \ldots.\ u_{qI}]$).

σ_q: spread or Gaussian width of the q^{th} basis function.

2. Input normalization is helpful for setting the centers of the basis functions in the RBN for better network performance. In general, basis function centers are selected as the inputs of some training cases. Hence, the training process is more efficient and successful when the dimensions of these centers are the same, because overlapping between basis functions will be consistent in all dimensions and for all centers (see Figure 2.3). In addition, the locations and distances between adjacent centers are organized and equally spaced, which produces a training grid with relatively evenly distributed basis functions.
3. Normalization speeds up the training process (Priddy & Keller, 2005). Initiating the training process with inputs that have the same scales improves its performance. It finishes faster, because extracting the problem patterns, which are the inherent relationships between various problem inputs and outputs, is easier with inputs that have small variances.

There are, of course, many approaches to normalization: statistical normalization, standardization, min-max normalization, sigmoidal normalization, energy normalization, etc.; and all methods improve the network training process. For this work, a standardization method is used, because it provides small feature variances and the same normalized ranges for all input features. Standardization compresses all the values to be between -1 and 1. Unlike other approaches, standardization retains the sign of the original input, thus allowing the trained network to be sensitive to changes in sign for the input

features. This is critical for general applications, given that different features can represent vastly different physical entities.

Equation 3.2 shows the standardization equation applied to the i^{th} component of **x**, where $\boldsymbol{x}^m$ is the m^{th} training case. The standardized value of the i^{th} component in the m^{th} training case $\bar{x}_i^m$ is determined by dividing the original value x_i^m by the maximum absolute value of the i^{th} component, $x_{i_{max}}$, as shown in Equation 3.3, where M is the number of training cases. The consequent transformed value is always between -1 and 1.

$$\bar{x}_i^m = \frac{x_i^m}{x_{i_{max}}} \in (-1,1) \tag{3.2}$$

$$x_{i_{max}} = \max[\left|\mathrm{x}_i^1\right|, \left|\mathrm{x}_i^2\right|, \left|\mathrm{x}_i^3\right|, \ldots, \left|\mathrm{x}_i^M\right|] \tag{3.3}$$

The process of input normalization using standardization is fully automatic in this work, regardless of the application. The normalization is automated for both training mode, when the network receives the training inputs (i.e., training data) in the training process, and testing mode, when the trained network receives new test inputs (i.e., test data). The normalization in the testing mode is performed using the values, $x_{i_{max}}$ is specific, that are found in the training mode.

3.2.2. Setting the basis function spreads

The radial basis functions spreads (i.e., widths) $\boldsymbol{\sigma}$ have a significant impact on network accuracy, as they define the area covered by individual basis functions. In typical training approaches, the spread values are treated as design variables to be optimized in the training process. Other approaches set all the basis functions spreads to a fixed small value (e.g., 0.1) in order to reduce the computational time in the training

process. The proposed process in this work, however, uses a new mixed approach in setting $\boldsymbol{\sigma}$ to the optimal values. The RMSD method is first used to assign preliminary spread values $\boldsymbol{\sigma}^o$. Then, the final $\boldsymbol{\sigma}$ values are found by optimization. This section summarizes the RMSD component of the proposed methodology, and the use of optimization is discussed in Section 3.2.4. Note that well-selected values for $\boldsymbol{\sigma}^o$ are especially necessary not only for defining effective basis functions but also for finding the proper basis functions centers $\boldsymbol{U}$, as discussed in Section 3.2.3.

Each element in $\boldsymbol{\sigma}^o$ is determined based on the RMSD between each center and the closest neighboring basis function center (Saha & Keeler, 1990; Wasserman, 1993), which is shown in Figure 3.2. Equation 3.4 represents the calculation of the RMSD (i.e., Euclidean distance) for the Gaussian spread σ_j° of the j^{th} basis function, which is the difference between a center $\boldsymbol{u}_j$ and its closest neighbor $\boldsymbol{u}_{j,k}$. I is the dimension of the input vector $\boldsymbol{x}$. In order to calculate the RMSD for each basis function, in which its center ($\boldsymbol{u}_j$) could be set using any training case as will be discussed in the next step of the training process (Section 3.2.3), it is assumed in this step that all training cases are centers for basis functions. In other words, the calculations of this step assume that there are M basis functions, and their spreads $\boldsymbol{\sigma}^o$ need to be calculated. Thus, the step of setting $\boldsymbol{\sigma}^o$ assigns every training case to have its own preliminary spread σ_j°. Consequently, there are M preliminary spreads as candidates for their corresponding basis functions, but typically fewer basis functions ($Q < M$) are selected in the next step.

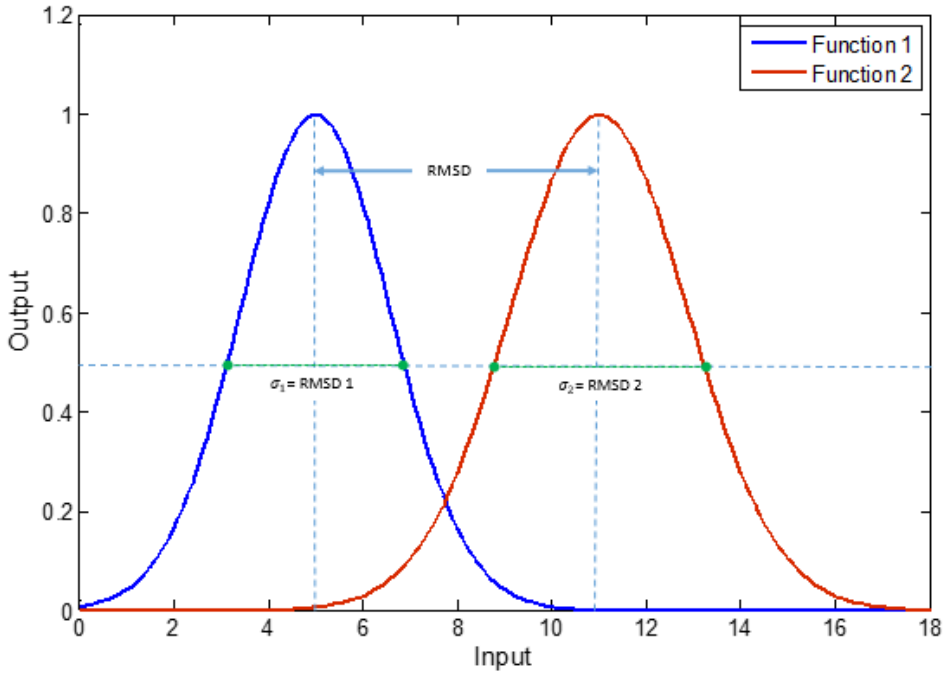

Figure 3.2: Two-dimensional plot for the root mean square distance (RMSD) between two neighboring basis functions.

$$\sigma_j^{\circ} = \text{RMSD} = \sqrt{\sum_{i=1}^{I} \left(u_{ji} - u_{j,ki}\right)^2} \tag{3.4}$$

By determining $\boldsymbol{\sigma}^{o}$ with Equation 3.4, overlapping between adjacent centers is minimized. Each basis function contributes more to the neural network for inputs closer to its center than to any other centers. The closest center $\boldsymbol{u}_{j,k}$ is the normalized input vector for the k^{th} training case ($\boldsymbol{u}_k = \overline{\boldsymbol{x}}_k$) with the smallest Euclidean distance among all M-1 centers from the center $\boldsymbol{u}_j$, which is the normalized input in the j^{th} training case ($\boldsymbol{u}_j = \overline{\boldsymbol{x}}_j$). Once all the Euclidean distances are calculated and saved in an array, the smallest distance is identified through direct search.

The distance-based approach of setting the preliminary spread values $\boldsymbol{\sigma}^{o}$ in this work gives larger σ_j° for the basis functions with further centers and vice versa. The next step is to set the basis function centers ($\boldsymbol{U}$) by selecting the important training cases based on their contributions (importance) in the network predicted output(s).

3.2.3. Selection of basis function centers

The matrix of basis function centers $\boldsymbol{U} = \left[\boldsymbol{u}_1\ \boldsymbol{u}_2\ \ldots \boldsymbol{u}_Q\right]^T$ is selected based on the orthogonal least square (OLS) method proposed in the literature (Chen, Cowan, & Grant, 1991), but with modifications to the method for improved performance. This section summarizes the traditional OLS method and details new enhancements.

As one of the important network parameters to be selected, the basis function centers (locations of the basis functions) in the RBN are generally selected as a subset of random size from the training cases (detailed in Chapter 2). The reason for selecting the centers from the training cases is that the RBN is designed to produce its output(s) based on local activation of the closest basis functions to the network input (Equation 3.5).

$$\mathrm{y} = \sum_{q=1}^{Q} \exp\left[-\frac{\sum_{i=1}^{I}(x_i - u_{qi})^2}{2{\sigma_q}^2}\right] * w_q \tag{3.5}$$

In Equation 3.5, y is the network output, Q is the number of basis functions, x_i is the r^{th} input in the input vector, u_{qi} is the i^{th} element in the vector of the q^{th} center (the q^{th} row in the matrix $\boldsymbol{U}$), σ_q is the Gaussian spread of the q^{th} basis function, and w_q is the q^{th} output weight.

With random selection of a subset from the available training cases to be set as basis function centers, important training cases might be left out of the training space.

This, in turn, would leave a portion of the training space uncovered by any basis function or poorly explained (predicted) by the network. Moreover, noisy training data might be included in the selected centers, which would reduce the network output accuracy, even if the network were well-trained. Therefore, methods like the orthogonal least square method (OLS) (Chen, Billings, & Luo, 1989) and the k-means clustering method (Haykin, Haykin, Haykin, & Haykin, 2009) are used to consider the importance of various training cases when selecting the basis function centers. This work uses the OLS method for the following reasons:

1. The OLS method has a better approach for the setting of its heuristically defined parameter than that in the k-means method.
2. The k-means method is performed with no consideration for the network performance (i.e., without testing the network's errors). The method formulates the basis function centers based on clustering various training cases depending on the distances between them. Clustering the data might result in losing some of it and might increase the error in actual testing because of using clustered centers that might not be distributed to cover the whole training space to produce appropriate accuracy. In contrast, the OLS method has a termination criterion that is directly related to the network performance by calculating the contribution of each training case in the predicted network output for all training cases.
3. The k-means method does not help draw any conclusions about the importance of a specific training case or the task inputs because the method produces clusters that are used as basis function centers. On the other hand, the OLS method maintains the

selection of centers from the original training data based on a direct relationship with their contributions to the network output.

Orthogonal least square (OLS) method overview

In general, the OLS method determines which training cases are most significant. This is done by formulating an error function that is used to evaluate (score) each normalized training case. Then, training cases with the highest scores are selected and provide the locations of the basis function centers $\boldsymbol{U}$. The number of selected training cases is $M_s \leq S$, so the dimension of $\boldsymbol{U}$ is $M_s \times I$. S is a randomly selected subset number of cases of the complete training set (M). The OLS algorithm first designs an initial network using just a single basis function that has a center equal to the normalized input of a training case. This initial training case is selected as the training case that results in the minimum error when predicting the output of all training cases (i.e., the case with the highest score). Then, additional basis functions are included one at a time, the error is re-evaluated, and the process is repeated iteratively until a criterion based on the error is satisfied. With each iteration, the additional basis function is transferred to a space where it becomes orthogonal to all previously added basis functions, in order to calculate the contribution of each basis function in the network predicted output independently. The OLS also implicitly determines the preliminary values for the output weight vector $\boldsymbol{w}^o$.

As suggested earlier, the original OLS method is used typically as an independent training process for RBN. This work, however, uses OLS within a new multi-stage training process. In the new process, the OLS method is mainly used for setting $\boldsymbol{U}$. Unlike typical training approaches, setting $\boldsymbol{U}$ using the OLS is not performed randomly. Rather, it considers the relative importance of the training cases when selecting $\boldsymbol{U}$ and inherently

the most appropriate number of basis functions. The proposed advancements include 1) a novel double termination criteria and 2) use of all *M* training cases within the OLS algorithm, rather than just a subset *S* of the complete training set. Unlike the original OLS, the new design is mainly introduced for applications with a limited number of training cases. Thus, inclusion of all training cases in searching for the most contributed cases can help in better selection of the most necessary basis functions and their centers $\boldsymbol{U}$, and eventually fewer heuristics in the training process. The new termination criteria, as will be shown later, guarantees the termination of the OLS procedures with a number of basis functions closest to the optimal for any application.

Algorithm procedures

The first step with the OLS method is to extract the columns of an orthogonal matrix for use when developing the basis functions. This is done in order to separate the significance (contribution) of each individual training case (i.e., different basis functions are correlated), given that inputs of the training cases are used as basis function centers. For simplicity, the network is assumed to have scalar output, denoted by y, for the calculations in the rest of this section. When multiple outputs exist, the OLS calculations are found for each output separately, and then the error scores are averaged over all outputs. For a problem with single output, Equation 3.6 segregates the vector of exact (true) output from all *M* training cases $\boldsymbol{t} = [t_1, t_2, \dots, t_M]^T$ into the sum of the predicted network $\boldsymbol{y}$ and the error $\boldsymbol{e}$. Note that with the proposed modification, all available *M* training cases are eventually used to search through during the iterative process, but all *M* training cases are not necessarily used as basis functions ($M_s \leq M$). $\boldsymbol{y}$ is further segmented in Equation 3.7, where $\boldsymbol{H} = [\boldsymbol{h}_1 \boldsymbol{h}_2 \dots \boldsymbol{h}_M]$ (the dimension of $\boldsymbol{H}$ is $M \times M$) is a

matrix representing the outputs from the *M* basis functions for the *M* training cases. $\boldsymbol{h}_i = [h_1\ h_2\ \ldots h_M]^T$ is a vector that represents the i^{th} basis function outputs, where each output is produced when the corresponding training case is received by the network. $\boldsymbol{w} = [w_1\ \ldots\ \ldots w_M]^T$ is a vector of output weights from all *M* basis functions.

$$\boldsymbol{t} = \boldsymbol{y} + \boldsymbol{e} \tag{3.6}$$

$$\boldsymbol{y} = \boldsymbol{H} \cdot \boldsymbol{w} \tag{3.7}$$

The next step in the OLS algorithm involves using QR-decomposition (Golub & Van Loan, 2012) to transform the rows of the $\boldsymbol{H}$ matrix in Equation 3.7 into a set of orthogonal basis vectors. This process is shown as follows:

$$\boldsymbol{H} = \boldsymbol{ZA} \Longrightarrow\ \boldsymbol{y} = \boldsymbol{ZA} \cdot \boldsymbol{w} \tag{3.8}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & \alpha_{12} & \cdots & \alpha_{1\,M_s} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \cdots & \ddots & \alpha_{M_s-1\,M_s} \\ 0 & & 0 & 1 \end{bmatrix} \tag{3.9}$$

The decomposition of $\boldsymbol{H}$ produces an orthogonal matrix $\boldsymbol{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_M]$, where $\boldsymbol{z}_i = [z_{i1}, z_{i2}, \ldots, z_{iM}]^T$, and an upper triangular matrix $\boldsymbol{A}$, which is shown in Equation 3.9. $\boldsymbol{Z}$ is then used to calculate $\boldsymbol{g} \in R^M$, which is a vector of the OLS solutions for the *M* training cases. $\boldsymbol{g}$ is defined using the Gram-Schmidt method (Björck, 1967), as shown in Equation 3.10, with its scalar form shown in Equation 3.11. The significance of $\boldsymbol{g}$ is that it is used to calculate the contribution of each training case in reducing the network prediction error. In the new produced orthogonal space, $\boldsymbol{Z}$ and $\boldsymbol{g}$ represent the orthogonal basis functions and their weights (scales), respectively, to calculate the network output $\boldsymbol{y}$, as will be shown next.

$$\boldsymbol{g} = (\boldsymbol{Z}^T\boldsymbol{t})/(\boldsymbol{Z}^T\boldsymbol{Z}) \tag{3.10}$$

$$g_i = \frac{\boldsymbol{z}_i{}^T \boldsymbol{t}}{\boldsymbol{z}_i{}^T \boldsymbol{z}_i}, \qquad 1 \le i \le M \tag{3.11}$$

The space spanned by the original basis functions, which are not orthogonal to each other and are represented by $\boldsymbol{H}$ and $\boldsymbol{w}$ (Equation 3.7), is the same space spanned by the new orthogonal basis functions, represented by $\boldsymbol{Z}$ and $\boldsymbol{g}$ in Equation 3.12. Unlike the original basis functions, the contribution of each orthogonal basis function when producing $\boldsymbol{y}$ can be calculated independently. Since the error $\boldsymbol{e}$ is usually not known a priori (Equation 3.6), it can be minimized by maximizing the predicted (i.e., explained) part of the true output $\boldsymbol{t}$ variance introduced by the orthogonal basis functions $\boldsymbol{Z}$. The explained variance of $\boldsymbol{t}$ is found and maximized by the Gram-Schmidt method. $\boldsymbol{e}$ represents the unexplained variance of $\boldsymbol{t}$ that cannot be found, which is responsible for the error that occurs in all prediction models. The explained $\boldsymbol{t}$ variance is maximized by selecting the training cases, based on $\boldsymbol{Z}$ and $\boldsymbol{g}$, that have the maximum contributions to create the new orthogonal basis functions in the network model.

$$\boldsymbol{y} = \mathbf{Z} \cdot \mathbf{g} \tag{3.12}$$

Again, using the Gram-Schmidt method, each g_i and $\boldsymbol{z}_i$ are substituted into Equation 3.13 to calculate what is called the *error reduction ratio* $[err]_i$, which is used to determine which training case should be used to set the next basis function and its center, $\boldsymbol{u}_i$. The value $[err]_i$ represents the contribution of the i^{th} training case in the $\boldsymbol{t}$ explained variance. In other words, $[err]_i$, which is proportional to g_i and $\boldsymbol{z}_i$, represents the increment to the network accuracy introduced by the i^{th} training case.

$$[err]_i = \frac{g_i^2 \boldsymbol{z}_i^T \boldsymbol{z}_i}{\boldsymbol{t}^T \boldsymbol{t}} \tag{3.13}$$

The training case with the highest $[err]_i$ is selected as the next most important basis function. The normalized input associated with that case is then used as the center for its corresponding basis function ($\boldsymbol{u}_q = \bar{\boldsymbol{x}}_i$).

The OLS method iterates to calculate $\boldsymbol{z}_i$, g_i, and $[err]_i$, which are used to determine a new center with each iteration until the termination criterion is satisfied. The criterion is typically satisfied when the sum of the error reduction rates is close to 1, as shown in Equation 3.14. Then, the final number of selected significant training cases $M_s \leq M$ is the final number of basis functions.

$$1 - \sum_{j=1}^{M_s} [err]_j < \varepsilon \tag{3.14}$$

The tolerance value (ε) is set heuristically to a small value (e.g., 0.01).

Although it is efficient in many applications, the original termination criterion shown in Equation 3.14 can lead to poor results in others. The method was originally designed for signal processing applications that usually include thousands of training cases, many of which are highly dependent on each other. Poor results are especially prevalent when the output has large variance (i.e., is distributed over a large scale). In such case, the OLS might terminate before it achieves enough reduction in the variance. The termination criterion terminates the method earlier than it should. The underlying concept of the OLS method is to reduce the training error by maximizing the explained variance in the true output (i.e., decreasing the error caused by the variance in the network model). Moreover, poor OLS performance can be obtained when the problem output has a relatively large mean value, because the error reduction ratio, Equation 3.14, directly depends on the output values. Thus, the method may terminate based on the error

reduction ratio condition, but still has large actual training error. In that case, the network ends up with fewer neurons than needed. Furthermore, if the tolerance ε is set to be too small, it leads to a network with an excessive number of basis functions.

This work proposes new termination criteria in order to improve the OLS method performance, as well as to prevent unstable performance at different types of outputs. The proposed criteria terminate the algorithm based on the calculation of the mean square errors (MSEs) and the sum of error reduction ratios ($\sum_{j=1}^{k}[err]_j$), as shown in Equation 3.15.

$$\{\,[1-\textstyle\sum_{j=1}^{k}[err]_j \le \varepsilon_1]\ \ \&\ \ [MSE_k \ge MSE_{k-1}] \quad \text{or}\ \ MSE_k \le\ \varepsilon_2\,\} \tag{3.15}$$

$$\text{MSE} = \frac{1}{MN}\sum_{n=1}^{N}\sum_{\text{m}=1}^{M}(t_{nm}-y_{nm})^2 \tag{3.16}$$

In Equation 3.15, and assuming the k^{th} iteration is the current iteration, MSE_k and MSE_{k-1} are the calculated MSEs for the current and previous iterations, respectively. ε_1 is the tolerance for error reduction ratio (set heuristically to small value, e.g., 0.0001). ε_2 is the tolerance for the MSE_k in the current iteration (set heuristically to small value, e.g., 0.1). In Equation 3.16, y_{nm} is the n network-predicted output for the m^{th} training case, and $t_{nm}$is the n^{th} exact output in the m^{th} training case.

The new criteria guarantee the termination of the OLS method only after it selects the proper number of neurons for the final network design, and for any type of problem. Along with the calculated $\sum_{j=1}^{k}[err]_j$, the criteria account for the training error by calculating the MSE at each iteration. Thus, the first condition is not satisfied unless MSE_k stops decreasing or starts increasing. Requiring that $MSE_k \ge MSE_{k-1}$ ensures that

the OLS method reaches its optimum design at the k^{th} iteration (i.e., the network's best possible results are reached at the k^{th} iteration and start to move beyond the optimal model if more basis functions are added). The second termination criterion is introduced to address applications with relatively small mean outputs and outputs with small variance. It depends on the MSE_k, but considers the first condition implicitly, where MSE_k cannot be small unless $\sum_{j=1}^{k}[err]_j$ is almost 1. When the MSE_k is small, but not zero, the issue of over-fitting is avoided, because its existence is directly indicated by obtaining a too-small training error value. Thus, that condition avoids the possibility of designing a more complex network than needed.

After the OLS is completed, the normalized inputs of the training cases that are selected as the most significant cases are used as the final centers of the network basis functions, where $\boldsymbol{u}_i = \bar{\boldsymbol{x}}_i$. Consequently, the basis function centers $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_{M_s}]$ with the total number of basis functions equal to M_s. The network's preliminary output weight vector $\boldsymbol{w}^o$ is calculated directly by back-substitution (Equation 3.17). If the network has multi-outputs, the network output weight becomes matrix $\boldsymbol{W}^o$, in which each row corresponds to one output.

$$[\boldsymbol{g} = \boldsymbol{A}\boldsymbol{w}^o] \equiv \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{M_s} \end{bmatrix} = \begin{bmatrix} 1 & \alpha_{12} & \cdots & \alpha_{1\,M_s} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \cdots & \ddots & \alpha_{M_s-1\,M_s} \\ 0 & & 0 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{M_s} \end{bmatrix} \tag{3.17}$$

As stated, the OLS procedures work iteratively to calculate $\boldsymbol{z}_i$, g_i, and $[err]_i$ to assign and add the new center after each iteration until the termination criteria are satisfied.

The full procedures for the OLS algorithms are summarized in the following steps:

Step I: Start the counter for the number of selected basis function centers k=1. Using all *M* training cases, calculate $\boldsymbol{z}_i$, g_i, and $[err]_i$ for $1 \leq i \leq M$, which are shown in Equations 3.18 , 3.11, and 3.13, respectively. Note that $\boldsymbol{z}_i$ in the first step is set to $\boldsymbol{h}_i$ (Equation 3.1) since there are no previously chosen basis function centers.

$$\boldsymbol{z}_i = \boldsymbol{h}_i \tag{3.18}$$

If the problem has multiple outputs (assume *N* number of outputs), then g_{ij} is the i^{th} solution calculated for predicting the j^{th} output ($\boldsymbol{t}_j$) (Equation 3.19). $[err]_i$ is calculated as the average of g_{ij} calculated from *N* outputs (Equation 3.20).

$$g_{ij} = (\boldsymbol{z}_i^T \boldsymbol{t}_j)/(\boldsymbol{z}_i^T \boldsymbol{z}_i) \tag{3.19}$$

$$[err]_i = \sum_{j=1}^{L} (\frac{g_{ij}^2 \boldsymbol{z}_i^T \boldsymbol{z}_i}{\boldsymbol{t}_j^T \boldsymbol{t}_j})/N \tag{3.20}$$

Step II: Find the training case with the maximum error reduction rate among the calculated errors ($[err]_i$) in *Step I* (Equation 3.21). Then, set the first orthogonal column $\boldsymbol{z}_1$ to be the basis function output $\boldsymbol{h}_1^{i_1}$ that corresponds to that training case (Equation 3.22).

$$[err]_1^{i_1} = \max\{[err]_i \ , \ 1 \leq i \leq M\} \tag{3.21}$$

$$\boldsymbol{z}_1 = \boldsymbol{z}_1^{i_1} = \boldsymbol{h}_1^{i_1} \tag{3.22}$$

Step III: Set the counter to k=k+1. For the training cases ($1 \leq i \leq M$, and $i \neq i_1, \ldots\ldots, i \neq i_{k-1}$), calculate the following:

$$\alpha_{jk}^i = \frac{\boldsymbol{z}_j^T \boldsymbol{h}_i}{\boldsymbol{z}_j^T \boldsymbol{z}_j} \ , \quad (1 \leq j < k) \tag{3.23}$$

$$\boldsymbol{z}_k^i = \boldsymbol{h}_i - \sum_{j=1}^{k-1} \alpha_{jk}^i \, \boldsymbol{z}_j \tag{3.24}$$

$$g_k^i = (\boldsymbol{z}_k^i)^T \boldsymbol{t} / ((\boldsymbol{z}_k^i)^T \boldsymbol{z}_k^i) \tag{3.25}$$

$$[err]_k^i = \frac{(g_k^i)^2 (\boldsymbol{z}_k^i)^T \boldsymbol{z}_k^i}{\boldsymbol{t}^T \boldsymbol{t}} \tag{3.26}$$

In Equation 3.23, $\alpha_{jk}^i$is the k^{th} element in the j^{th} row in the matrix$\boldsymbol{A}$. In Equation 3.24, $\boldsymbol{z}_k^i$ is the produced k^{th} orthogonal vector in the matrix $\boldsymbol{Z}$ using the i^{th} training case. In Equation 3.25, g_k^i is the k^{th} OLS solution produced from the i^{th} training case. In Equation 3.26, $[err]_k^i$ is the k^{th} error reduction ratio due to the selection of the i^{th} training case. As in *Step I*, this step can simply include multiple outputs, where the values in Equations 3.25 and 3.26 are changed to be averaged like those in Equations 3.19 and 3.20.

Step IV: Find the case with maximum error among the calculated errors ($[err]_k^i$) in *Step III* (Equation 3.27). Then, set up the new current $\mathbf{z}_k$ to correspond to the case with maximum error ($[err]_k^{i_k}$) (Equation 3.28).

$$[err]_k^{i_k} = \max(\, [err]_k^i, \;\; 1 \le i \le M, i \ne i_1, .., i \ne i_{k-1} \,) \tag{3.27}$$

$$\mathbf{z}_k = \boldsymbol{z}_k^{i_k} = \boldsymbol{h}_{ik} - \sum_{j=1}^{k-1} \alpha_{jk}{}^{i_k} \, \boldsymbol{z}_j \tag{3.28}$$

Step V: Check the termination criteria (Equation 3.15). If the condition satisfies, exit the algorithm. Otherwise, return to *Step III*.

3.2.4. Optimizing network parameters

This section illustrates the optimization procedure, which is the last component of the new training process. The optimization procedure minimizes the training error in

order to decrease the network bias produced by the OLS method. Bias is defined as the difference between an estimator's expectations (i.e., the network's predicted outputs) and the true (actual) values of the outputs being estimated. The OLS method decreases model variance, which refers to the error produced when predicting the same output (i.e., the sensitivity of the model for the changes in the training set), but increases its bias. Figure 3.3 clearly shows the inverse relationship between the bias and variance, where the bias increases when variance decreases and vice versa. The optimal network design should minimize both variance and bias. As in the figure, the actual (true) error, "Test Error," is optimum (minimum) when both values are minimized. Consequently, this step decreases the bias and provides the optimal network design. More information about model complexity and bias-variance trade-off topics can be found in the literature (Friedman, 1997; Geman, Bienenstock, & Doursat, 1992; Kohavi & Wolpert, 1996).

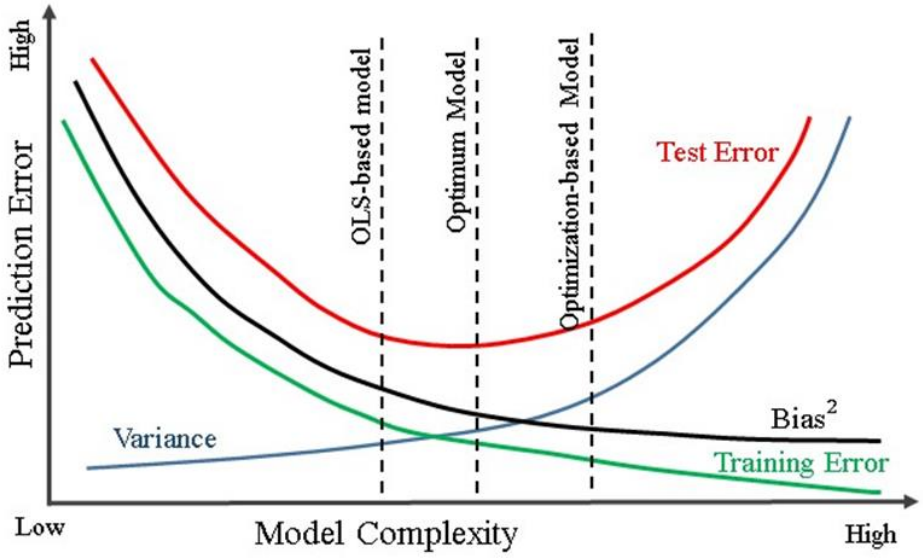

Figure 3.3: Model complexity vs. error in terms of its variance and bias.

As shown in Figure 3.3, the bias is reduced by minimizing the training error. The error can be represented by the sum-squared error (SSE) and minimized in Equation 3.29. The SSE is the difference between the predicted network output y_m and the actual output t_m for all training cases. The optimization has the basis function spreads $\boldsymbol{\sigma}$ and output weights $\boldsymbol{w}$ as its design variables. It also uses the values $\boldsymbol{\sigma}^o$, $\boldsymbol{w}^o$ found in the previous steps as initial guesses to help increase computational speed. The optimization is solved using the sequential quadratic programming method that is implemented with the MATLAB® environment (Fletcher, 2013; Hock & Schittkowski, 1983; Powell, 1983).

$$\text{Given: } \boldsymbol{\sigma}^o, \boldsymbol{w}^o \tag{3.29}$$

$$\text{Minimize: } f(\boldsymbol{\sigma}, \boldsymbol{w}) = \sum_{m=1}^{M}(t_m - y_m)^2$$

$$= \sum_{m=1}^{M}(t_m - (\sum_{q=1}^{M_s} h_{mq} * w_q))^2$$

$$\text{Subject to: } 0 < \sigma_q \quad ; q \in R^{M_s}$$

Using both $\boldsymbol{\sigma}$ and $\boldsymbol{w}$ as design variables provides additional degrees of freedom when creating the final optimal designs. Thus, the final optimal regression surface can be detailed enough to represent the complexity of the predicted problem sufficiently. Along with the number of basis functions, which is set in the OLS step, both $\boldsymbol{\sigma}$ and $\boldsymbol{w}$, which are optimized in this step, determine the regression surface complexity. The actual importance of $\boldsymbol{\sigma}$ is to change the degree of overlapping between various basis functions (see Equation 2.1 and Figure 2.3), while $\boldsymbol{w}$ is to scale the basis function outputs (i.e., provide the weighted importance for the basis function outputs) (see Equation 2.1 and Figure 2.5). Eventually, $\boldsymbol{w}$ also contributes in determining the final models' complexity order. The values of $\boldsymbol{\sigma}$ are constrained in order to keep positive values.

The optimization to involve multiple outputs (assume N number of outputs) is formulated as follows:

$$\text{Given: } \boldsymbol{\sigma}^o, \boldsymbol{W}^o \tag{3.30}$$

$$\text{Minimize: } f(\boldsymbol{\sigma}, \boldsymbol{W}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}(t_{nm} - y_{nm})^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}(t_{nm} - (\sum_{q=1}^{M_s} h_{mq} * w_{nq}))^2$$

$$\text{Subject to: } 0 < \sigma_q \quad ; q \in R^{M_s}$$

In Equation 3.30, $\boldsymbol{W}^o = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \ldots\ \boldsymbol{w}_N]$ is a matrix of preliminary output weight vectors for all N outputs, t_{nm} is the true m^{th} training value of the n^{th} network output, y_{nm} is the predicted m^{th} training value of the n^{th} network output, h_{mq} is the q^{th} basis function output for the m^{th} training case, and w_{nq} is the output connection weight between the q^{th} basis function and n^{th} output.

Unlike optimization in a traditional RBN, which finds $\boldsymbol{U}$, $\boldsymbol{\sigma}$, and $\boldsymbol{w}$ as design variables, this work does not optimize $\boldsymbol{U}$ with $\boldsymbol{\sigma}$ and $\boldsymbol{w}$, for the following reasons:

1. Optimizing $\boldsymbol{U}$ does not generally provide an acceptable generalization for the predicted problem, because the resulting locations of the centers might change completely from that in the training space (see Sections 2.2.3 and 2.2.4.1). It is more stable for the network to use the original training cases as their basis function centers, because their locations represent real values within the training grid. Optimizing $\boldsymbol{U}$ values could also change the training space by finding new locations outside the training space, which makes the network performance unstable. On the other hand, $\boldsymbol{\sigma}$ and $\boldsymbol{w}$ values are expected to change slightly from those found in the previous steps, to reduce the training error without changing the original training space.

2. Selecting $\boldsymbol{U}$ from the original training cases (the OLS method sets $\boldsymbol{u}_i = \bar{\boldsymbol{x}}_i$) helps post-analysis for the studied problem. These analyses are concerned with studying the most significant basis functions, training cases, and input parameters by connecting the network parameters to the task parameters. If $\boldsymbol{U}$ values are optimized to values different from the exact inputs of the training cases, it will be difficult to extract useful information about the original training cases and their input values.

After the optimization step, which sets the $\boldsymbol{\sigma}$ and $\boldsymbol{w}$ values, the network training process is completed. When the network is provided with new input (i.e., test case), the network provides instant prediction for the output. Using the parameters $\boldsymbol{u}_q$ and σ_q found in the steps of the training process, the basis function outputs (h_q) are calculated (Equation 3.31). Then, the network output (y), Equation 3.32, is calculated using $\boldsymbol{h}$ and $\mathbf{w}$.

$$h_q = \exp\left[-\left\|\boldsymbol{x} - \boldsymbol{u}_q\right\|/2\sigma_q^2\right] \tag{3.31}$$

$$y = \boldsymbol{h} \cdot \boldsymbol{w} \tag{3.32}$$

In Equation 3.31, $\boldsymbol{x} = [x_1 \dots \dots x_I]^T$ is the network input, $\boldsymbol{u}_q = [u_1 \dots \dots u_I]^T$, which is the q^{th} row in $\boldsymbol{U}$, is the q^{th} basis function center found in the OLS method, and σ_q is the spread of the q^{th} basis function. In Equation 3.32, $\boldsymbol{h} = [h_1 \dots \dots h_{M_s}]$ is a vector output from the M_s significant basis functions and $\boldsymbol{w} = [w_1 \dots \dots w_{M_s}]^T$ is the output weight vector found by the optimization.

3.3. Results

The new network, henceforth referred to as Opt_RBN, is evaluated in terms of its *testing error* (accuracy). Testing error is the error produced from the model when predicting output that corresponds to new inputs that have never been used as training cases. The Opt_RBN performance is first tested on three experimental regression problems and compared with other common networks. Then, its performance is evaluated and compared on practical regression problems.

3.3.1. Experimental regression problems

In the four experimental problems, Opt_RBN is compared with the following models: 1) linear regression, 2) feed-forward back-propagation network (FFN), and 3) the OLS-based RBN. The linear model is included as a well-accepted baseline. The FFN is included because it is the most commonly used ANN. The OLS-based RBN is included because it represents the closest ANN to the proposed Opt_RBN.

Regarding the used ANN models, the FFN model is proposed in the literature (Hagan, Demuth, & Beale, 1996), where the user heuristically sets the number of basis functions in the hidden layer before the network connection weights ($\boldsymbol{w}$) are optimized based on the back-propagation approach and using early stopping criterion. With respect to the OLS-based RBN, the design in MATLAB® (Beale, Hagan, & Demuth, 2001; Chen et al., 1991) is used, where the user sets the desired training error and Gaussian spread value, and the network then creates the proper number of basis functions to reach the training error.

Testing error is calculated based on two common methods: 1) root mean square error (RMSE) and 2) mean absolute error (MAE) (Hyndman & Koehler, 2006). The RMSE measures the average magnitude of the error (Equation 3.33) over the test sample *N*. (i.e., the squared averaged difference between the forecast value (predicted value) (y_i) and corresponding observed value (true or exact value) (t_i)). Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors compared to small ones. This means the RMSE is most useful when large errors are particularly undesirable, which is the case in this work.

$$root\ mean\ square\ error\ (RMSE) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2} \tag{3.33}$$

The MAE measures the average magnitude of the errors in a set of forecasts (predictions), without considering their direction (Equation 3.34). The MAE measures the average of the absolute values of the differences between forecast value (y_i) and the corresponding observation (t_i) over the testing sample *N*. The MAE is a linear score, which means that all the individual differences are weighted equally in the average.

$$mean\ absolute\ error\ (MAE) = \frac{1}{N}\sum_{i=1}^{N}|t_i - y_i| \tag{3.34}$$

3.3.1.1 Example 1

The first simulation example is a mathematical equation with two inputs (x_1 and x_2) and a single output $f(\boldsymbol{x})$ (Equation 3.35).

$$f(\boldsymbol{x}) = \sqrt{2x_1x_2} - x_2 + 4\ , \quad x_i \in [0,20] \tag{3.35}$$

Five training cases are created randomly from the equation, with an input range between 0 and 20 for both variables. Only five training cases are used in this simulation in order to evaluate the performance of the new network with a reduced number of training cases. Three testing cases, which differ from those used to train the models, are used to calculate each model's errors.

The training and testing cases are resampled three times, meaning the four models are built three times using three different sets of training and testing cases. Based on the rule of thumb, there are three basis functions used in the hidden layer of the FFN, while the OLS-based RBN is set to have training error equal 0.01 with spread of 0.4. After trying various values, these settings produce the best possible performance for the used models. A summary of the testing errors is shown in Table 3.1. The Opt_RBN model outperforms all models with relatively high accuracy. The second most accurate model is the RBN. The linear model, as expected, cannot provide accurate results because the equation being simulated is nonlinear. The result of FFN is also relatively inaccurate. With respect to computational speed, all models produce testing results in a fraction of a second.

Table 3.1: Test error produced from the four regression models, Linear, FFN, RBN, and Opt_RBN, for simulation example 1.

Model Type	RMSE	MAE
Linear	42.6	35.3
FFN	34.2	28
RBN	19.6	18
Opt_RBN	12.8	10.9

To ensure that the Opt_RBN always provides improved performance (i.e., minimal test error), its performance is evaluated at various numbers of training cases. The Opt_RBN test error is calculated and compared with the RBN model only, because RBN is the most accurate comparable model. Both models are built and tested using 3, 5, 11, and 21 training cases, and the RMSE for the test set is reported in Figure 3.4. These results are, again, the averages of the errors resulting from using three different training and testing sample sets. The results show superior performance for Opt_RBN, which is approximately half the error obtained by the RBN. Even with a larger number of training cases, the Opt_RBN still outperforms the RBN. Even though an RMSE difference of 5 between the Opt_RBN and RBN seems relatively small in some applications, such a difference can be considered significantly large in applications like human motion prediction.

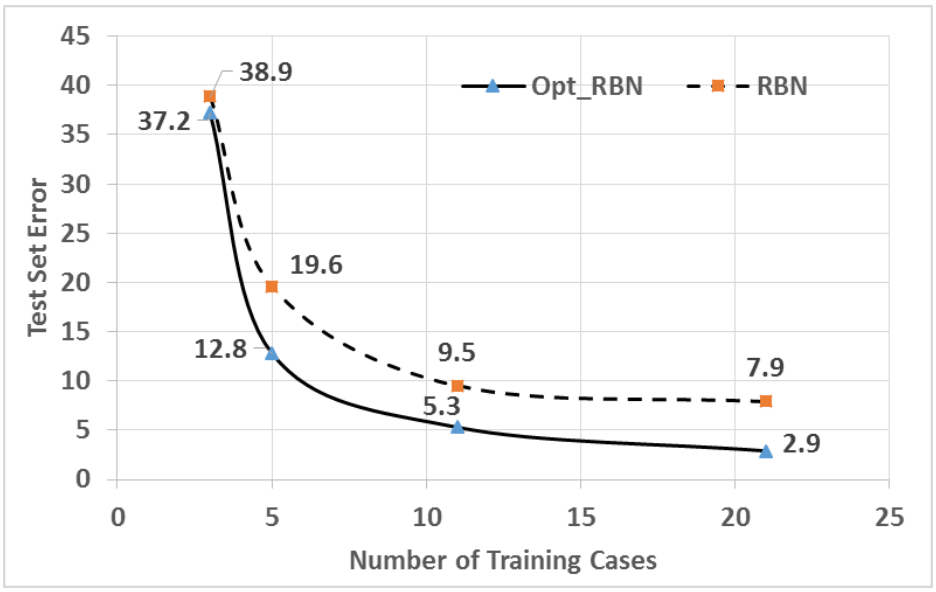

Figure 3.4: Test set RMSE plots for RBN and Opt_RBN at various numbers of training cases for simulation example 1.

The Opt_RBN performance shows faster convergence to its minimum test error using fewer training cases than those in RBN. Both networks start with slightly better Opt_RBN test error at three training cases, which is too small a training size. Then, the Opt_RBN has significantly fewer test errors for the remaining numbers of training cases.

3.3.1.2 Example 2

The second example involves prediction of two mathematical equations, ($f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x})$, both of which have two inputs (x_1 and x_2) (Equation 3.36). Twenty training cases are created randomly from each equation for an input range between 0 and 20 for both variables. The number of training cases used is expected to be limited for a model to be trained to predict two outputs.

$$f_1(\boldsymbol{x}) = \sqrt{2x_1x_2} - x_2 + 4\,, \qquad x_i \in [0,20] \tag{3.36}$$

$$f_2(\boldsymbol{x}) = x_2^2 + 3x_1\,, \qquad x_i \in [0,20]$$

As in Example 1, three test cases are used to evaluate and compare the performance of the four regression models. Again, using the rule of thumb in setting the proper number of basis functions, there are five basis functions used in the hidden layer of the FFN, while the OLS-based RBN is set to have training error equal 0.01 with spread of 0.3. The resulting RMSE and MAE are shown in Table 3.2, where the errors in the table represent the average errors for both predicted outputs. The best performance is achieved by the Opt_RBN for both error measurements (RMSE and MAE).

Table 3.2: Test error produced from the four regression models, Linear, FFN, RBN, and Opt_RBN, for simulation example 2.

Model Type	Average RMSE	Average MAE
Linear	25.1	23
FFN	18.9	16.4
RBN	16.7	11.6
Opt_RBN	5.3	3.6

Figure 3.5 illustrates plots for the resulting RMSEs produced from the RBN and Opt_RBN models trained with 5, 10, 20, and 50 training cases. A relatively small RMSE was obtained in the Opt_RBN with as few as 20 training cases. Figure 3.5 also suggests better ability for the Opt_RBN for reaching small error with fewer training cases. This in turn enhances the promising conclusions regarding the improved performance of Opt_RBN with a reduced number of training cases.

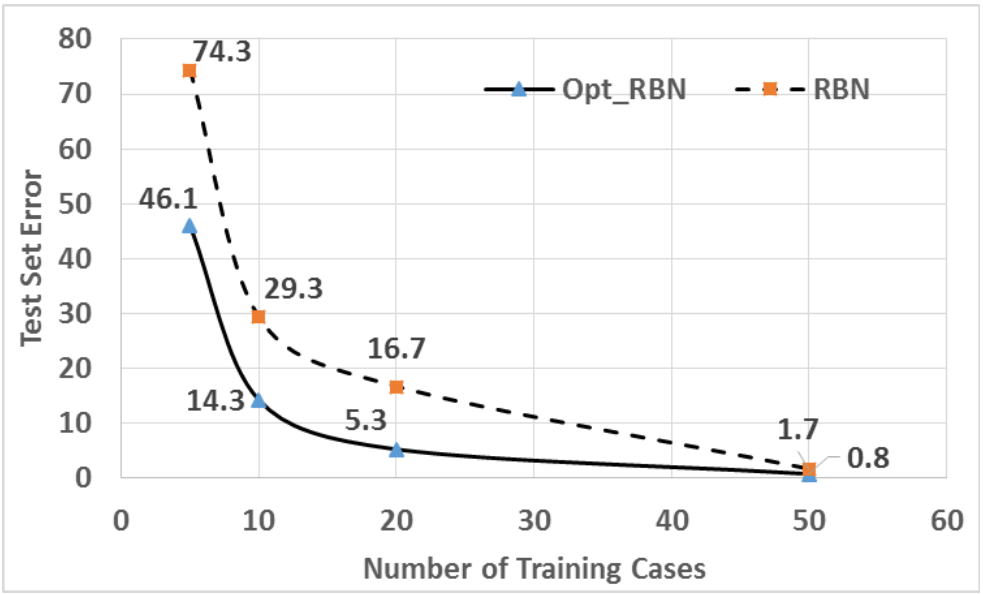

Figure 3.5: Test set RMSE plots for RBN and Opt_RBN at various numbers of training cases for simulation example 2.

On the other hand, the results in Figure 3.5 show that when more training cases are available, the Opt_RBN outperformance starts to slightly decline, where its error results become closer to those produced from the RBN. The different performance between both models decreases with more available training data, because the Opt_RBN can have over-fitting issue due to the optimization step that reduces the training error in the training process. With more training data, the Opt_RBN parameters are optimized to learn the training data more than the proper generalization, which reduces the test error.

3.3.1.3 Example 3

Example 3 is more complex than the first two examples, as it has three outputs, $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$, and $f_3(\boldsymbol{x})$, representing three mathematical equations, each with two inputs (x_1 and x_2) (Equation 3.37).

$$f_1(\boldsymbol{x}) = \sqrt{2x_1x_2} - x_2 + 4\,, \qquad x_i \in [0,100] \tag{3.37}$$

$$f_2(\boldsymbol{x}) = x_2^2 + 3x_1\,, \qquad x_i \in [0,100]$$

$$f_2(\boldsymbol{x}) = x_2 + 3x_1 - 2\,, \qquad x_i \in [0,100]$$

Fifty training cases are created from each equation with an input range of 0-100 for both variables. Note that the range of inputs is more than that in the first two examples, in order to produce a more challenging problem. The wider input ranges increase the problem's complexity; the model is expected to predict all possible inputs over that range of inputs.

The four models' performances are evaluated using five test cases and by calculating the produced RMSE and MAE, which are averaged over all outputs (Table 3.3). There are five basis functions used in the hidden layer of the FFN, while the OLS-based RBN is set to have training error equal to 0.01 with spread of 0.3. The Opt_RBN

model again demonstrates the best performance. The RBN and FFN experience difficulties in predicting this example, as evidenced by the large errors shown in Table 3.3. With respect to computational speed, all models produce testing results in a fraction of a second.

Table 3.3: Test error produced from the four regression models, Linear, FFN, RBN, and Opt_RBN, for simulation example 3.

Model Type	Average RMSE	Average MAE
Linear	621.2	530.8
FFN	110.7	94.1
RBN	92.2	58.6
Opt_RBN	15.7	11

It is necessary for the network models in this example to be trained with more training cases because, given the three outputs (equations) to be predicted using one network with a wide range of inputs, the problem is more complex than the first two examples. Thus, the RBN and Opt_RBN are trained with 10, 20, 50, and 100 training cases. The results show superior performance for the Opt_RBN (see Figure 3.6). When using 20 training cases, the Opt_RBN produces RMSE around two times better than that produced from the RBN and FFN when trained with 50 training cases. In Figure 3.6, even though 10 training cases are not enough in either case, the results show significant differences in the performance of the two models. Furthermore, the Opt_RBN error with 50 cases is still better than that in RBN with 1000 cases. Considering the complexity of Example 3, the use of 100 training cases could be sufficient for the Opt_RBN, since the error is approximately 4.8. On the other hand, given the test error reached by the

Opt_RBN, the RBN results indicate its need for more training cases to approach results comparable to those obtained by Opt_RBN.

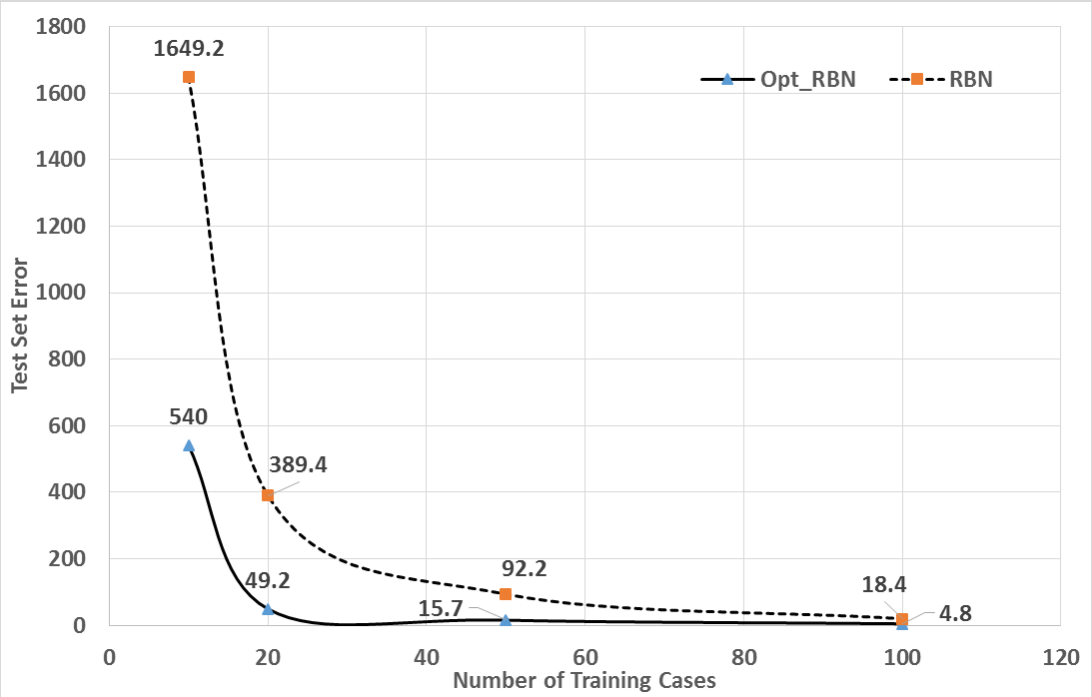

Figure 3.6: Test set RMSE plots for RBN and Opt_RBN at various numbers of training cases for simulation example 3.

3.3.1.4 Example 4

The fourth simulation example is the most challenging one with two equations and five inputs ($x \in \mathbb{R}^5$) (Equation 3.38).

$$f_1(\boldsymbol{x}) = \sum_{i=1}^{5} {x_i}^2 \ , \quad x_i \in [-10,10] \tag{3.38}$$

$$f_2(\boldsymbol{x}) = \sum_{i=1}^{5} \left[({x_i}^2 - x_i)^3 - (2 + x_i)^2\right] \ , \qquad x_i \in [-2,2]$$

Seventy training cases are created for each equation, as well as 10 test cases to evaluate the network models' performances. Again, the training and testing cases are resampled three times. There are 15 basis functions used in the hidden layer of the FFN, while the OLS-based RBN is set to have training error equal to 0.01 with spread of 0.3. The produced averaged RMSE and MAE for the test cases are shown in Table 3.4, which shows that the Opt_RBN outperforms the other models. The Opt_RBN shows around 20% better performance than the FFN and RBN.

Table 3.4: Test error produced from the four regression models, Linear, FFN, RBN, and Opt_RBN, for simulation example 4.

Model Type	Average RMSE	Average MAE
Linear	313.7	158.3
FFN	182.4	12
RBN	192.3	12
Opt_RBN	151.8	10.4

The RBN and Opt_RBN are trained with 10, 30, 50, 70, and 100 training cases, and the error results are shown in Figure 3.7. Opt_RBN produces smaller errors at all presented training cases, and its errors decline faster than the RBN. In Figure 3.7, Opt_RBN, when trained with only 30 cases, provides error less than the RBN when trained with 100 cases.

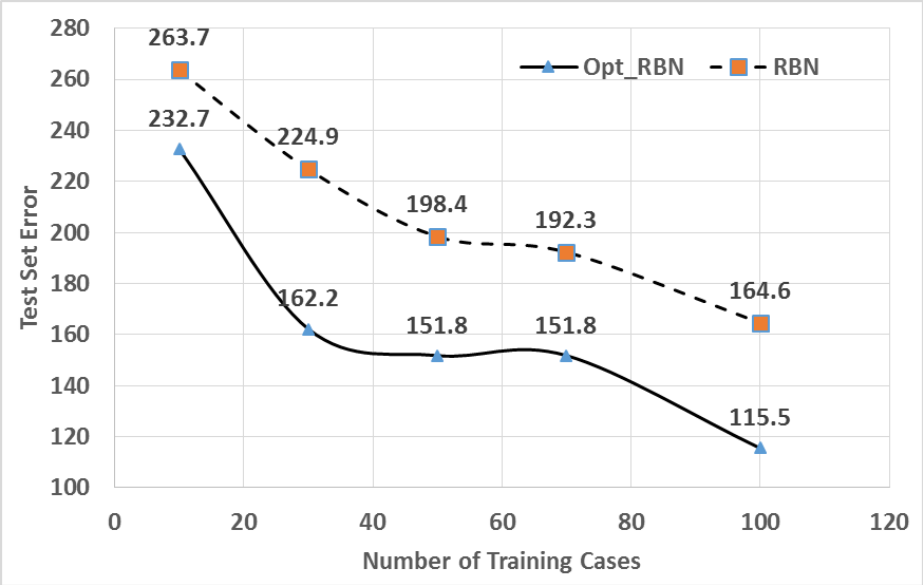

Figure 3.7: Test set RMSE plots for RBN and Opt_RBN at various numbers of training cases for simulation example 4.

It is typical for the ANN in some cases to provide results with no improvement, or sometimes worse results, when trained with more training cases. Such a case occurs for the Opt-RBN, as shown in Figure 3.7, when trained with 50 and 70 cases, respectively. The main reason for that to occur is that sometimes the locations of the additional training cases within the training space are very close to other previously existing cases. Thus, the additional training cases do not provide the network with new information regarding the training space in order to improve the hyper-surface that is produced when the network is being trained. Furthermore, sometimes some of the additional training cases might improve the accuracy in the resulting regression hyper-surface at some portions (less test errors for some test cases), but other training cases cause issues like over-fitting (see Chapter 2) that lead to poor performance in other surface portions (more test errors for some other test cases).

In general, the results of the four presented examples indicate that RBN and FFN models need more training cases in order to provide accuracy comparable to that provided by Opt_RBN. In other words, Opt_RBN is capable of improving the prediction results for a problem for the available given training cases. For example, if example 4 has 100 cases available for training, Opt_RBN can produce results that are around 30% better than those from RBN.

3.3.2. Practical (real-world) regression problems

For real-world regression problems, the Opt_RBN performance is evaluated and compared with the OLS-based RBN on two problems that are related to prediction of forces on the knees for injury prevention application using multi-scale human modeling software. The network comparison is also performed at different numbers of training cases, but only one set of training and test cases are used (no resampling or training with multiple sets of data) when evaluating the network performance. There is only one set of training cases available in these problems, because collecting new cases for resampling in the problems is too costly. Beside the practical implication of evaluating the successful prediction capability for the new RBN design, the new network provides a new, faster tool for injury prevention. In addition, the network facilitates the integration of multiple software within the same environment without running them separately. Furthermore, predicting the knee forces problem illustrates an initial investigation for the network performance when is being applied on digital human modeling problems.

Overview: Multi-scale predictive human model for injury prevention

To improve human performance and help prevent orthopedic joint injuries, which account for more lost days for military personnel than any other medical issue, a tool for coupling dynamic motion prediction, muscle activation, and high-fidelity finite-element models (FEAs) of various joints within a multi-scale human model is being developed. Although multi-scale modeling is an active area of research, there have been few efforts to seamlessly link high-fidelity biomechanical models with a complete system-level digital human model (DHM) for injury prevention (Sultan & Marler, 2012). Thus, the developed system (Figure 3.8) links the following: 1) Santos dynamic motion prediction "Santos® Human," 2) OpenSim model to calculate muscle activation force, 3) FEA model (ABAQUS®) to calculate the maximum stress, strain, and contact pressure in the joint components, and 4) a system for predicting the propensity for injury based on potential mechanical failure in soft tissue.

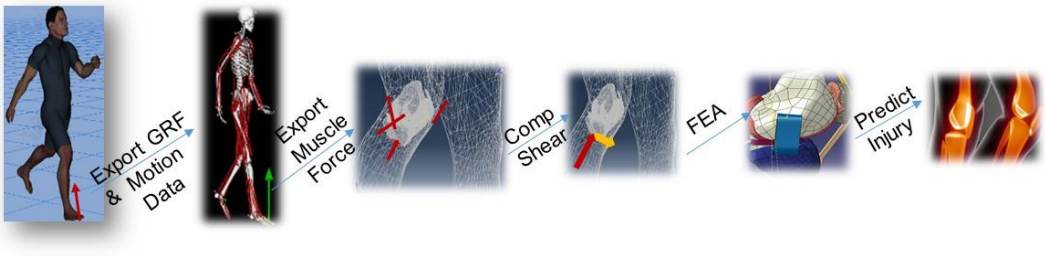

Figure 3.8: Illustrative diagram for the linked software and models that form the multi-scale predictive human modeling for injury prevention.

In order to evaluate FEA results in real time, ANN is used in this work to approximate the high-fidelity models. This integrated multi-scale DHM is a promising system and allows one to track joint angles and torques, muscle activation, and joint

stress during a simulated task. The presented problem is to calculate the maximum forces and stresses on the knee joint while simulating two motion tasks: walking and stairs ascent. Extracting these values for the knee joint over various segments of the motion profile is a time-consuming process, because multiple software programs for human modeling and FEA (as seen in Figure 3.8) need to be run successively. We present this problem as an example of a problem with limited available training cases to evaluate the Opt_RBN design0F[1]. For this problem, there are four inputs: knee joint angle (degrees), compression force on the knee (N), shear force on the knee (N), and ground reaction force (N). The problem has three outputs: maximum stress on the knee (MPa), maximum percent of strain on the knee (%), and maximum contact pressure on the knee (MPa). The network-predicted results are evaluated separately for the walking and stairs-ascent tasks. The same numbers of inputs and outputs and training cases exist for both tasks.

Walking task

For the walking task, there are 25 training cases and three test cases (Table A.1 in Appendix A). The Opt_RBN and RBN networks are trained with 6, 12, 17, and 25 cases, and the error results of predicting the test cases are shown in Figure 3.9. Opt_RBN outperforms RBN at all used training cases. The exceptional performance of the Opt_RBN is especially clear in the results of training with fewer cases. Even with the full 25 training cases, RBN produces error more than that produced by Opt_RBN. Except when RBN is trained with 25 cases, the Opt_RBN resulting error with six training cases is better than any of those resulting from the RBN.

[1]The work of collecting and preparing the training cases for the knee force problems was performed with valuable input and direct contribution from Dr. Sultan Sultan.

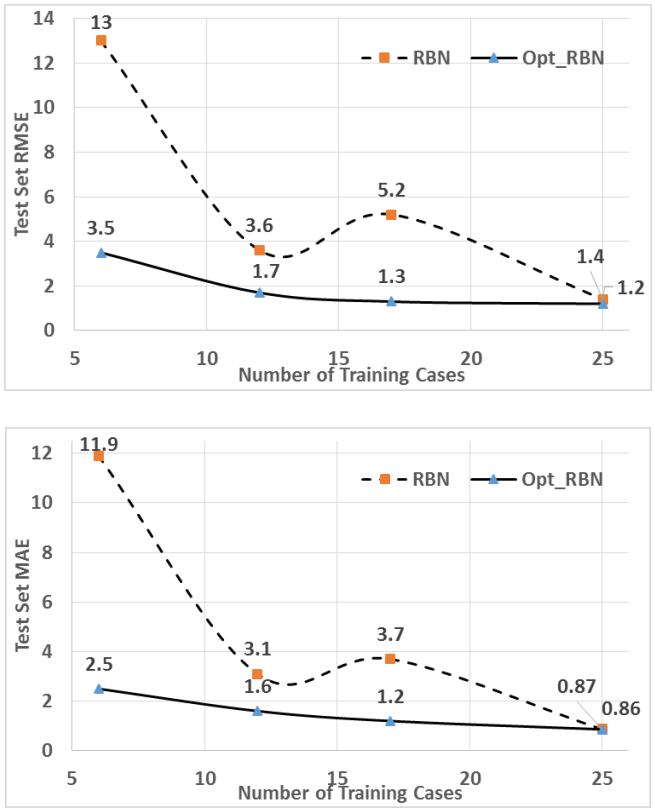

Figure 3.9: Test set RMSE and MAE for RBN and Opt_RBN at various numbers of training cases for predicting the knee stresses and forces for the walking task.

The same results trend is shown for both RMSE and MAE plots in Figure 3.9. The results show that the RBN model has larger test error when trained with 17 cases than when trained with 12 cases. Beside the aforementioned reasons in Example 4 (Section 3.3.1.4), such a case occurs because the models in this problem are trained with one set of training cases. Actually, having one set of training cases increases the frequency of such cases occurring in the practical problems, especially those with limited numbers of training cases or those that are costly to create.

Stairs-ascent task

The second task is stairs ascent; its training and test cases are shown in Table A.2 (Appendix A). Again, the Opt_RBN and RBN networks are trained with 6, 12, 17, and 25 cases, and the test results are shown in Figure 3.10. As in the walking task, the Opt_RBN shows better performance in the stairs-ascent task when trained with fewer training cases. However, RBN in this task relatively outperforms the Opt_RBN when trained with the 17 and full 25 training cases. Although the Opt_RBN result becomes worse when trained with 17 and 25 cases for the reasons mentioned earlier, its error in both cases is not far from that produced by the RBN. On the other hand, the Opt_RBN results with six training cases are more than twice as good as those of the RBN.

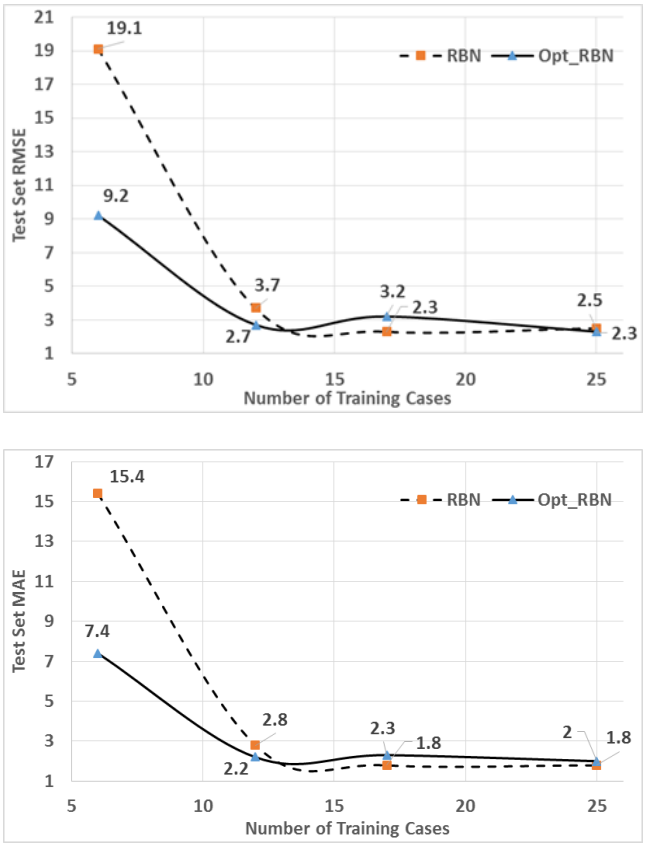

Figure 3.10: Test set RMSE and MAE for RBN and Opt_RBN at various numbers of training cases for predicting the knee stresses and forces in the stairs-ascent task.

The results of the presented examples prove the improved performance introduced by the new RBN design (Opt_RBN). The general practical implication shows that Opt_RBN can be trained with fewer training cases, compared to other ANN and regression models, to achieve a proper accuracy level. When only a limited number of training cases exists, the network can also provide better results over competing ANNs.

3.4. Discussion

This work proposes a new RBN design to overcome the poor performance of ANNs when used in applications that have limited numbers of training cases available. The new RBN design consists of multi-stage training techniques to set up necessary network parameters in a rigorous and integrated process. In addition, the algorithms are modified, so the training process can be performed with minimal heuristics. This integration of OLS and RMSD, with an optimization-based approach, is the primary novelty of the proposed approach and proves to be effective.

The new design "Opt_RBN" is tested on four experimental problems, and the results are compared with those from three models: linear regression, FFN and RBN. The results show that Opt_RBN outperforms the other models in all examples. In addition, for any given number of training cases, the results prove the better response for the Opt_RBN to produce smaller error compared with the competing ANNs. Then, Opt-RBN is evaluated and compared with RBN on two practical regression problems. In general, Opt_RBN evaluation shows substantial outperformance when trained with fewer training cases. The Opt_RBN shows stable performance, especially when trained with fewer training cases for all presented experimental and real-world problems.

The new double termination criteria in the OLS method and the quadratic cost function used in the optimization step guarantee that the Opt_RBN design demonstrates high robustness and stability to provide improved performance. The robust and stable behaviors of the new design are illustrated by its results in all presented examples, as well as when the design is evaluated with multiple training and test samples in the experimental problems.

The new design proposes a smarter ANN that is capable of improved learning rather than needing more training data. The new design opens new fields for the use of ANNs in applications with limited numbers of training data, such as digital human modeling. The design is introduced with a focus on improving the prediction ability for a unique problem, which is the regression problem with reduced available training sets. The use of training algorithms with minimal heuristics allows the new RBN design to produce results with quality that none of the competing methods have achieved. That prediction quality is facilitated by the use of OLS to set the inputs of the significant training case, which are selected from all available training data, as $\boldsymbol{U}$, and the optimization to find the optimal $\boldsymbol{\sigma}$ and $\boldsymbol{w}$ values.

The improved performance of the new design is achieved at the expense of training time, because the network includes a multi-stage training process. Although the new network runs in a fraction of a second for the test cases like the other networks, it requires more time to be trained. However, its training time is still less than one minute in all presented examples, and training time is not as important as the run time for test cases for most practical applications. The training time for the Opt_RBN design might be an issue for future work when the network is used to predict large-scale problems, in terms

of outputs. That is because the optimization needs to find a large number of variables, possibly in the thousands. In the next chapter, the model performance will be evaluated when predicting large-scale practical problems that have limited numbers of training cases, such as motion prediction of a digital human model with full degrees of freedom. Moreover, incorporation of constraints for such applications within the network design will be investigated in Chapter 5.

CHAPTER IV

NEW DESIGN FOR PREDICTION OF LARGE-SCALE HUMAN DYNAMIC MOTION APPLICATIONS

4.1. Introduction

There are many algorithms designed to simulate various digital human model (DHM) tasks and scenarios. As a unique DHM algorithm in terms of its predictive capabilities and accuracy, predictive dynamic (PD) is used for simulation of various DHM motions and scenarios. Predictive dynamics is a physics-based algorithm that consists of an optimization problem with hundreds of design variables and thousands of constraints (Xiang, Chung, et al., 2010). Although PD simulations are crucial and useful, they are computationally expensive. For even small changes to the task conditions, the simulation needs to run for a relatively long time (minutes to tens of minutes). Thus, there can be a limited number of training cases due to the computational time and cost associated with collecting training data. In addition, the PD problem is relatively large with respect to the number of outputs, where there are hundreds of outputs (between 500-700 outputs) to predict for a single problem. Therefore, there is a critical need for a powerful computational model to provide real-time simulations for PD problems, and this necessity leads to the use of tools like the artificial neural network (ANN) in this work.

In response to the special needs of the large-scale DHM problems like PD with limited training data available, this chapter proposes the use of a newly developed radial-basis network (RBN) proposed earlier in Chapter 3 (Opt_RBN) (Bataineh & Marler, 2015). Although the Opt_RBN design is proven to improve the prediction results for

application with a reduced number of training cases, applying the new design on the large-scale PD application is somewhat difficult. The Opt_RBN experiences a memory issue when running the optimization step in its training process (Section 3.2.4) to predict all PD outputs from a single network model. Therefore, this chapter introduces new algorithms to modify some steps of the new Opt_RBN training process to address the memory issue. Eventually, with the new modifications, the Opt_RBN can provide real-time prediction of a complete PD problem. Nonetheless, the new RBN design and its training process proposed in Chapter 3 should still be the typical choice when predicting any regression application with reduced training sets. The modified steps should only be used for large-scale applications similar to the PD.

This work's contributions include: 1) a modified Opt_RBN training process for improved performance in large-scale problems with minimal training data, 2) application of the new modified Opt_RBN design for real-time prediction of PD tasks for full DHM, and 3) construction of an RBN design that can be populated for any general large-scale problem in various applications.

With the successful implementation of the modified algorithms within the new RBN design in this chapter, the network performance for any potential over-fitting issue (see Chapter 2 for details) is investigated on experimental simulations. Then, its capability to provide real-time prediction of two common PD tasks, walking and going prone, is evaluated. The results are promising, with relatively small errors obtained when predicting approximately 500-700 outputs from a single network model. Although this chapter presents modification to the Opt_RBN driven by its potential use with PD, the consequent ANN design can be used with a broad range of large-scale problems; PD is

simply a well-studied example problem for the proposed developments. The new proposed ANN design can be used for general applications in various large-scale engineering and industrial fields that experience delay issues when running computational tools that require a massive number of procedures and a great deal of CPU memory.

4.2. Background: Predictive dynamic (PD)

Predictive dynamics is a physics-based motion simulation algorithm for DHM (Xiang, Chung, et al., 2010). The PD method is distinguished from other motion simulation tools because it produces simulations that reflect the effects of any changes in the DHM conditions. Since its development, PD has been used to simulate different motion tasks and scenarios (Kim et al., 2008; Kim, Xiang, Yang, Arora, & Abdel-Malek, 2010; Kwon et al., 2014; Xiang, Arora, & Abdel-Malek, 2010, 2012; Xiang, Arora, Rahmatalla, et al., 2010). Successful validation of the PD results has been provided using motion capture systems (Rahmatalla, Xiang, Smith, Meusch, & Bhatt, 2011).

The PD algorithm involves solving a nonlinear optimization problem to find control points (i.e., motion profiles) for all body degrees of freedom (DOFs), while adhering to the equation of motion and considering various physical limits and other motion-related constraints. Equation 4.1 shows a simplified formulation of the PD optimization problem. The problem is to find the design variables $\boldsymbol{q}$, which represent the control points (i.e., joint angle profiles) of all body DOFs, to minimize a group of human performance measures, $f(\boldsymbol{q})$, subject to the physical equality and inequality constraints. The constraints include: body contact points, joint ranges of motion (ROMs), torque

limits, zero moment point (responsible for balance), equation of motion, ground reaction forces, etc.

$$\begin{aligned} &\text{Find: } \boldsymbol{q} \text{ (control points for 55-DOFs)} \\ &\text{Minimize: } f(\boldsymbol{q}) = f(\boldsymbol{q} - \boldsymbol{q}_{MoCap}) + f(\textstyle\sum_1^{DOFs} joint\ torque) \\ &\text{Subject to: } h_i(\boldsymbol{q}) = 0,\ \ i = 1, \dots, m \\ &\qquad\qquad g_j(\boldsymbol{q}) \leq 0,\ \ j = 1, \dots, k \end{aligned} \tag{4.1}$$

In Equation 4.1, $\boldsymbol{q}$ is a vector that represents the design variables (control points for various body DOFs), $\boldsymbol{q}_{MoCap}$ is a vector that represents the reference motion provided by motion capture (i.e., seed motion), $h_i(\boldsymbol{q})$ is the i^{th} equality constraint ($\in \mathbb{R}^m$), and $g_j(\boldsymbol{q})$ is the j^{th} inequality constraint ($\in \mathbb{R}^k$).

The control points ($\boldsymbol{q}$) that are found by optimization in PD eventually form B-splines for all DOFs that simulate the motion in a DHM model. Each DOF has its B-spline, Figure 4.1, to be calculated by optimization with satisfied constraints, under the provided task condition. Having more control points in a B-spline leads to more accurate motion simulation, but it is computationally more expensive due to the existence of more design variables. When a PD task is being developed, its conditions eventually become inputs for the task. Although many inputs are common in all tasks, like loading and clothing, other inputs are task specific, like step size in the walking task, box height in the jumping-on-the-box task, etc.

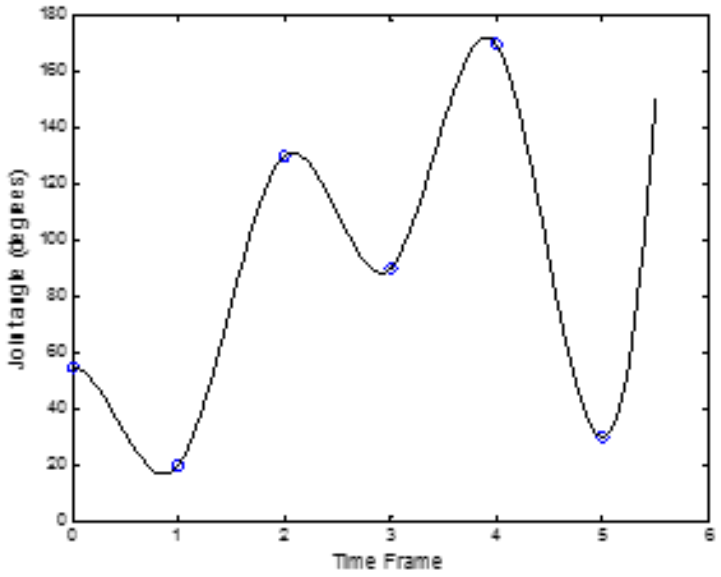

Figure 4.1: B-spline for six control points (i.e., joint angle profiles) at six time frames of a total task time.

The PD algorithm in this work is applied on a DHM called Santos (Abdel-Malek et al., 2006). Santos is a full-body DHM with a high-fidelity 55 DOFs. Even though there are many task conditions to be considered in motion simulation, this work uses Santos as a soldier under the conditions of various loading configurations and some reduced ROMs.

In PD simulation with Santos, creating a relatively large number of different task conditions (thousands) is difficult because it is time costly. The running time for each case, even with minor condition changes, takes minutes to hours to complete and produce the simulation. This process cannot be automated completely because the simulations require some post-processing procedures. Hence, the time-cost issue leads to a PD problem with a limited number of simulations available to be used by pattern recognition

tools like ANN to be trained to provide real-time PD simulations. In general, the number of created cases (conditions) for PD applications is expected to be limited to 10s to 100s.

The large size of the PD problem with respect to the number of outputs is another challenge for creating a simulation model to provide real-time prediction of the problem. The number of outputs in a PD task is approximately 500-700 outputs, depending on the number of control points in each task. These outputs, as mentioned earlier, represent $\boldsymbol{q}$ for all 55 DOFs. Typical optimization tools cannot be used in solving such problems. Therefore, the PD algorithm uses special optimization software that is designed to solve large-scale optimization problems like PD motion simulation. The following section illustrates new methodologies proposed for successful real-time simulation of a large-scale PD problem, which has a limited number of training cases, using the new RBN design.

4.3. Method

The fundamental ANN model in this work is the new RBN (Opt_RBN) for reduced training sets, which is detailed in Chapter 3. That RBN design is chosen because it outperforms other typical ANN models, especially for applications with a limited amount of training data. As one of these applications, the PD problem is also considered large-scale in terms of the number of its outputs (approximately 500-700 outputs). As stated earlier, the new Opt_RBN design experiences a memory issue when being trained to predict this relatively large number of outputs. In general, the large-size problem that causes the memory issue in the new design might be produced when there is a large

number of training cases (tens of thousands), inputs (thousands), outputs (hundreds), or a combination of them.

In the case of the PD problem, the original Opt_RBN design cannot be trained when the problem has more than 200 outputs. Hence, for successful simulation of the PD problem, this work illustrates some modifications for the steps of the network training process that experience the memory issue. The modifications essentially would allow the training process to be performed by typical algorithms and software, especially for the optimization step of the process, without the need for a special large-scale optimization program or a special large storage computer. The new modified Opt_RBN design performance is also investigated for potential limitations. Eventually, the proposed new design can be populated for any large-scale application with reduced training sets.

Before presenting the new design methodologies, the network inputs and outputs need to be defined from the PD problem. The PD background provided earlier demonstrates the specific PD configurations in this work. The general PD problem has 41 inputs, 16 for the loading configurations (like dress, armor, backpacks, etc.), 24 for joint range of motion (ROM) (there are 12 DOFs in the different joints in the spine: spine low, mid-low, mid-high, and high, where each joint includes three DOFs: spine-bending, spine-flexion, spine-rotation), and 1 that is related to the weapon location between the hands. The loading inputs represent the mass values for the body links that are affected by the loadings and the three-dimensional locations of their centers of mass. The ROM inputs represent the upper and lower limits of the included 12 ROMs. Other task-specific inputs can be added, like walking speed in the walking task or box height in the jumping-on-the-box task. The network outputs consist of control points that represent the joint

motion profiles, where the number of points is task dependent, for the full 55 DOFs. The total number is expected to be between 500-700 outputs. Figure 4.2 provides a general diagram of the ANN model for the PD problem. The ANN model can be populated to simulate other PD outputs like joint torque for the full 55 DOFs, ground reaction forces on feet, etc.

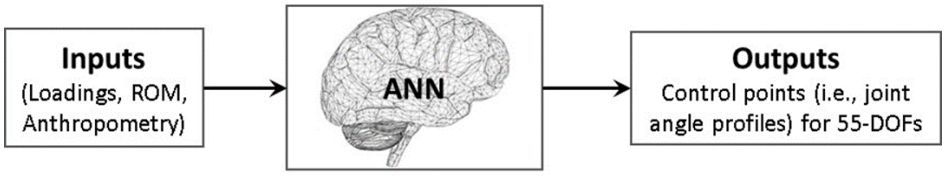

Figure 4.2: ANN diagram for general predictive dynamic (PD) applications.

Before it can be used, the ANN needs to be trained on the task being modeled. A detailed description of the used fundamental RBN design is provided in Chapter 3. In summary, the training process in that design consists of four main steps. First, inputs of training cases are all normalized by the standardization method to rescale all inputs to be within the same scale, between -1 to 1. Next, preliminary values for the network basis function (i.e., Gaussian) spreads $\boldsymbol{\sigma}^o$ are set for all potential basis functions using the root mean square distance (RMSD) by the Euclidean distance-based calculations. Then, the basis function centers $\boldsymbol{U}$ and preliminary values for connection weights $\boldsymbol{W}^o$ are set using a new orthogonal least square (OLS) method. Finally, optimal values for $\boldsymbol{W}$ and $\boldsymbol{\sigma}$ are calculated by optimization. The steps of setting $\boldsymbol{\sigma}^o$ and optimization are modified in this work to overcome the memory issue when applied in PD applications.

4.3.1. Training process for large-scale problem with reduced training set

In PD problems, there is a limited number of training cases, as well as a relatively large number of outputs to be predicted. This work modifies the training process in the new RBN design (Opt_RBN) for use with maximum accuracy in large-scale PD motion application without the aforementioned memory issue. Although the modification to the Opt_RBN is driven by potential use with PD, the consequent ANN design can be used with a broad range of large-scale problems. However, the training process proposed in Chapter 3 should still be the typical choice when predicting any regression application with reduced training sets, and the modified steps should only be applied for large-scale applications similar to PD.

With respect to the Opt_RBN training process, the optimization step (Equation 3.29), when applied to the PD problem, has tens of thousands of design variables (approximately 40000). Thus, typical CPU memory and optimization algorithms cannot solve this large optimization problem. Therefore, the modifications proposed for the new RBN design specifically tackle the step of setting the preliminary basis function spreads $\boldsymbol{\sigma}^o$ and the optimization step of setting the basis function spreads and connection weights $\boldsymbol{W}$. These modifications are necessary to allow division of the original optimization step to perform in multiple runs for groups of outputs. Each optimization run has a smaller problem size (approximately 300-1000 design variables) than that in the original optimization problem. With the proposed modifications, the original single optimization problem reaches the exact final solutions reached by the multiple optimization runs. The proposed design modifications are illustrated next, and the

Opt_RBN design with the new modifications is then evaluated for potential performance issues.

4.3.1.1 New approach for setting of basis function spreads

In the original proposed design in Section 3.2.4, along with the connection weights ($\boldsymbol{W} = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \ldots\ \boldsymbol{w}_N]$) corresponding to N outputs, the vector of the basis function widths, $\boldsymbol{\sigma}$, is included as design variables in the optimization step, Equation 3.30. Unlike the connection weights, $\boldsymbol{W}$, the widths $\boldsymbol{\sigma}$ are coupled and cannot be separated into different optimization runs. In other words, all $\boldsymbol{\sigma}$ elements contribute in producing all network outputs, while each vector of $\boldsymbol{W}$ contributes in producing one of the network outputs independently from the other vectors (i.e., all $\boldsymbol{W}$ elements are completely independent from each other). That said, in order to be able to divide the optimization (Equation 3.30) to be solved in multiple smaller optimization problems, the $\boldsymbol{\sigma}$ needs to be removed as a design variable from the optimization.

Removing $\boldsymbol{\sigma}$ from the optimization necessitates introducing more rigorous heuristic setting of its preliminary values $\boldsymbol{\sigma}^o$ because these values will be used as the final values and will no longer be optimized. The new heuristic setup is especially needed for an application with either too limited or too many available training cases. As stated, in the original design, $\boldsymbol{\sigma}^o$ elements are set to the RMSD (Section 3.2.2). However, the RMSD value might be almost zero when there is a relatively large number of training cases, since the distance between the normalized inputs in many training cases is too small. In a case when few training cases exist, the calculated $\boldsymbol{\sigma}^o$ based on the RMSD could be too large to produce a network with accurate predictions. If the widths are removed from the optimization step, their values need to be found using a more

intelligent method in order to use these values in the final network design. Along with considering the availability of various numbers of training cases, the new method should also consider performing the training steps with minimal heuristics.

The new modified design of the width $\boldsymbol{\sigma}$ calculation is proposed in Equation 4.3. Setting $\boldsymbol{\sigma}$ for large-scale problems like PD mainly depends on the use of the Manhattan distance (Köthe & Garling, 1969; Schaefer & Wolff, 1999) between the input vectors of the two adjacent training cases, which is shown in Equation 4.2. The Manhattan distance is the sum of absolute distances between two points (vectors) in all dimensional space. In multi-dimensional space, this distance has different representation of the actual distance between two adjacent points than that provided by the Euclidean distance. Then, the ratio of the number of training cases to the number of inputs is used together with the Manhattan distance to set the width σ_i (Equation 4.3). The step of width calculation assumes all available training cases as potential basis functions; thus, a width value is set for each available training case. The widths that correspond to the selected training cases in the OLS step are used in the final network design.

$$\left\|u_{jk}\right\|_1 = \sum_{\mathrm{i}=1}^{I} \left|u_{ji} - u_{j,ki}\right| \tag{4.2}$$

$$\sigma_j = \left(\frac{M}{I}\right) \left\|u_{jk}\right\|_1 , \qquad j \in \mathbb{R}^M \tag{4.3}$$

In Equation 4.2, $\left\|u_{jk}\right\|_1$ represents the Manhattan distance between the inputs of the j^{th} training case and its closest k^{th} training cases. u_{ji} is the i^{th} input element of the j^{th} training case, and $u_{j,ki}$ is the i^{th} input element of the k^{th} training case that is closest to the j^{th} training case. In Equation 4.3, σ_j is the spread (Gaussian width) of the j^{th} training case

when its input is selected as the basis function. I is the size of input vector (i.e., the number of elements in the input vector). M is the number of training cases.

The new formula, Equation 4.3, is formulated so that it counts for the training size of the application in order to provide appropriate $\boldsymbol{\sigma}$ values for any problem. The $\boldsymbol{\sigma}$ values are proportional to the ratio between the size of the training cases and the input dimension $\left(\frac{M}{I}\right)$. A preliminary work was performed by investigating multiple approaches that lead to the ratio $\frac{M}{I}$ as acceptable heuristic setting. In general, if the input size (i.e., the number of input parameters) I increases, more training cases M are required to provide more combinations of these parameters. The increase in I is implicitly considered by increasing M. Even when I increases with fixed M, the resulting $\|u_{jk}\|_1$ is larger because typically the training points are further from each other in higher dimensional space. The larger $\|u_{jk}\|_1$ compensates for the fixed M, producing larger σ_j. Hence, the ratio $\frac{M}{I}$ provides an appropriate factor for any problem to produce proper $\boldsymbol{\sigma}$ values to handle any problem's size with the most generalization. In addition, this ratio reduces the existence of too-small $\boldsymbol{\sigma}$ values.

The heuristic-based $\boldsymbol{\sigma}$ values in the new method are used as the final values. Even when the PD application has a relatively large number of training cases, in the hundreds or thousands, the $\boldsymbol{\sigma}$ values are calculated with a balance between the distance $\|u_{jk}\|_1$ and the training size. In summary, the new method is appropriate for heuristically setting a value $\boldsymbol{\sigma}$ that depends not just on the $\|u_{jk}\|_1$ value, but is also sensitive to various numbers of training cases and inputs.

4.3.1.2 New grouped optimization of network output weights

The weights $\boldsymbol{W} = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \ldots\ \boldsymbol{w}_N]$ for different outputs can be decoupled, because each vector of output weights $\mathbf{w}_n = \left[\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_Q\right]^T$ is independent from the others used to produce the network outputs. Hence, this work proposes a new grouped optimization step for each group of outputs that are related to each other (Equation 4.4). Multiple grouped optimization runs are performed, and each includes the weights of outputs that are related to 1 DOF. There are 55 separate optimization runs corresponding to the 55 DOFs of the DHM. In contrast to the original optimization problem (Equation 3.30), which involves finding all vectors of $\boldsymbol{W}$ in a single run, the new optimization involves multiple runs, each for a smaller group of vectors $\boldsymbol{W}_g$.

For $g = 1, 2, \ldots, G$: $(G = 55)$ (4.4)

Find: $\boldsymbol{W}_g = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \ldots\ \boldsymbol{w}_D]$

Minimize: $f(\boldsymbol{W}_g) = \frac{1}{D}\sum_{d=1}^{D}\sum_{m=1}^{M}(t_{g,d,m} - y_{g,d,m})^2$

$$= \frac{1}{D}\sum_{d=1}^{D}\sum_{m=1}^{M}\left(t_{g,d,m} - \sum_{q=1}^{Q}\left[h_{m,q} * w_{g,d,q}\right]\right)^2$$

In Equation 4.4, G is the number of optimization problems to be solved (55). $\boldsymbol{W}_g = \left[\mathbf{w}_{g,1}, \mathbf{w}_{g,2}, \ldots, \mathbf{w}_{g,D}\right]$ is the connection weight matrix that corresponds to the g^{th} group of outputs ($\boldsymbol{W}_g \in R^D$). $\boldsymbol{w}_{g,d} = \left[w_{g,d,1}, w_{g,d,2}, \ldots, w_{g,d,Q}\right]^T$ is the output connection weight vector that corresponds to the d^{th} output in the g^{th} group of optimization. D is the number of outputs in each g^{th} group. $t_{g,d,m}$ is the true value for the d^{th} output of the m^{th} training case in the g^{th} group. $y_{g,d,m}$ is the predicted (network) output for the d^{th} output of the m^{th} training case in the g^{th} group. $h_{m,q}$ is the q^{th} basis function output when receiving

the input of the m^{th} training case. $w_{g,d,q}$ is the connection weight value between the q^{th} basis function and the d^{th} output in the g^{th} group of optimization.

The memory and running time issues are solved in the new grouped optimization. It is also found that for practical PD applications the total running time in all grouped optimizations is less than that for the single optimization run. As an example to demonstrate that, a comparison of the running time is performed between the single optimization and the grouped optimization for the PD problem when the network is being trained with part of the outputs, because the single optimization cannot be performed for full PD problem. For a PD problem with 200 outputs, which is the maximum number of outputs the optimization can solve in a single run, the running time in the new grouped optimization is approximately 20 minutes, while it is approximately 24 minutes for the single optimization. The solutions from both problems are exactly the same.

4.3.2. Performance analysis for over-fitting issues

The performance of the modified Opt_RBN with the proposed new steps is evaluated for a potential issue that could limit the network accuracy. Specifically, the new network performance is evaluated for an over-fitting issue; it is of special concern in this work, because $\boldsymbol{W}$ is the only design variable in the optimization problem. A typical training algorithm (Equation 2.4 in Section 2.1.1), which involves a similar optimization problem, generally experiences poor network performance that is mainly due to over-fitting. Although that training algorithm differs from the new multi-stage training process in the Opt_RBN, it is worth assuring the resistance of the Opt_RBN design with the new modified optimization step for over-fitting by evaluating some experimental examples.

The examples are employed only to evaluate the network performance for minimal over-fitting because the new design is already validated on various problems in Chapter 3. The new Opt_RBN design is then thoroughly evaluated in the next section when applied on full PD problems.

To check for over-fitting in the new design, its existence is assumed. Then, the opposite is shown to be true. When an over-fitting problem exists, a simple method called regularization, or weight-decay, can be used to reduce its effect. The method is performed by adding a function of the design variables as a second objective function to the original one (see Section 2.2.3 for more details). Thus, the optimization problem becomes a multi-objective optimization (MOO), as shown in Equation 4.5. For simplicity, the optimization in Equation 4.5 assumes all outputs to be calculated in a single problem (G=1). The first cost function is the network prediction error, and the second one represents the regularization, which is basically the norm function of the design variables (Bishop & Nasrabadi, 2006).

$$\text{Minimize: } f(\boldsymbol{W}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}(t_{nm} - \sum_{q=1}^{Q} h_{mq}w_{nq})^2 + \|\boldsymbol{W}\| \qquad (4.5)$$

In Equation 4.5, t_{nm} is the true n^{th} output value in the m^{th} training case. h_{mq} is the q^{th} basis function output when the m^{th} training case is fed to the network. w_{nq} is the output weight value connecting the q^{th} basis function and the n^{th} output.

For network models with too many basis functions and DOFs, involving only the error function in the optimization problem might lead to relatively large values for $\boldsymbol{W}$ (on the order of thousands to tens of thousands) as the optimal design values. Such large values are a sign of a potential over-fitting issue because the produced network regression surface (i.e., the output (Equation 2.3) becomes too oscillated and unsmooth, see Chapter

2 for more details). When the regularization cost function is added to the optimization, the optimal $\boldsymbol{W}$ values are reduced which leads to a smoother resulting regression curve (Girosi, Jones, & Poggio, 1995).

Even with the regularization part, the over-fitting problem might exist because more weight might be given to minimize the error function in the MOO (i.e., the error function might be biased over the regularization function). Therefore, both functions in the MOO should be normalized to have comparable weights, as shown in Equation 4.6. The functions f_1^N and f_2^N represent the normalized values of the error function and the regularization function, respectively (Equation 4.7). The weights λ_1 and λ_2 are used to provide weighting factors for both functions, respectively. The values λ_i are positive and have a total of 1 ($\sum_{i=1}^{C} \lambda_i = 1$ and $\lambda_i \geq 0$). Normalization of the cost functions allows evaluation of a finite number of combinations for λ_i to analyze the effect of the regularization function on the optimization. Equation 4.7 shows the simplest and most common method to produce the normalized function $f_i^N(\boldsymbol{x})$.

$$\text{Minimize: } f(\boldsymbol{w}) = \lambda_1 f_1^N(\boldsymbol{w}) + \lambda_2 f_2^N(\boldsymbol{w}) \quad ; \sum_{i=1}^{C} \lambda_i = 1 \tag{4.6}$$

$$f_i^N(\boldsymbol{x}) = \frac{f_i(\boldsymbol{x}) - f_i^{min}(\boldsymbol{x})}{f_i^{max}(\boldsymbol{x}) - f_i^{min}(\boldsymbol{x})} \quad , \qquad f_i(\boldsymbol{x}) \in [0, 1] \tag{4.7}$$

The maximum value of the i^{th} cost function ($f_i^{max}(\boldsymbol{x})$) is found when the optimization is performed without that objective ($\lambda_i = 0$). The minimum value of the i^{th} cost function ($f_i^{min}(\boldsymbol{x})$) is found when the optimization is performed with that cost function only ($\lambda_i = 1$). When one function is minimized independently, the other function value is maximized. Thus, the new value for each function is between 0 and 1. Since the multi-objective function in Equation 4.6 is discontinuous due to the norm in the

second function, a special MATLAB® optimization package, "matlab_CVX," is used in this work to solve the optimization with a cost function that includes the norm (Grant, Boyd, & Ye, 2008).

To investigate over-fitting, the normalized MOO is tested on two of the experimental examples presented in Section 3.3.1. If any over-fitting exists in the original optimization (Equation 4.4), the average root-mean square error (RMSE) for the test cases should be lower when the regularization function is involved in the optimization ($\lambda_2 \neq 0$). Otherwise, the RMSE with no regularization ($\lambda_2 = 0$) should be lower than any other combination of λ_i.

Example 1: The first experimental example is shown in Equation 3.35. In this example, 21 cases are used to train the network using the proposed methods. The maximum value is 56421.07 (at $\lambda_1 = 0$) for error function and 194402277.7 (at $\lambda_2 = 0$) for regularization function. The minimum value for both functions is zero. The network optimization step is performed using the MOO and at various λ_i between 0 and 1. The average test RMSE is found for six test cases at each combination of weights. The RMSE values are drawn versus the used λ_2, as shown in Figure 4.3. In the figure, when $\lambda_2 = 1$ the error becomes significantly too large compared to that produced using any other λ_2 values. Actually, the MOO (Equation 4.6) always produces significantly large error when it involves the regularization function only ($\lambda_2 = 1$), because the error function is completely neglected and the weights are all minimized to zero. In Figure 4.3, it is obvious that the test error with no regularization ($\lambda_2 = 0$) is lowest. The solution with the regularization function shows more error that gradually increases when the contribution of the regularization function increases. Consequently, Example 1 shows

that the new modified optimization step still resists the occurrence of over-fitting because the minimum test error is produced when the regularization function is not included ($\lambda_2 = 0$) (i.e., when $\lambda_2 = 0$, the optimization (Equation 4.6) becomes the original formula (Equation 4.4)).

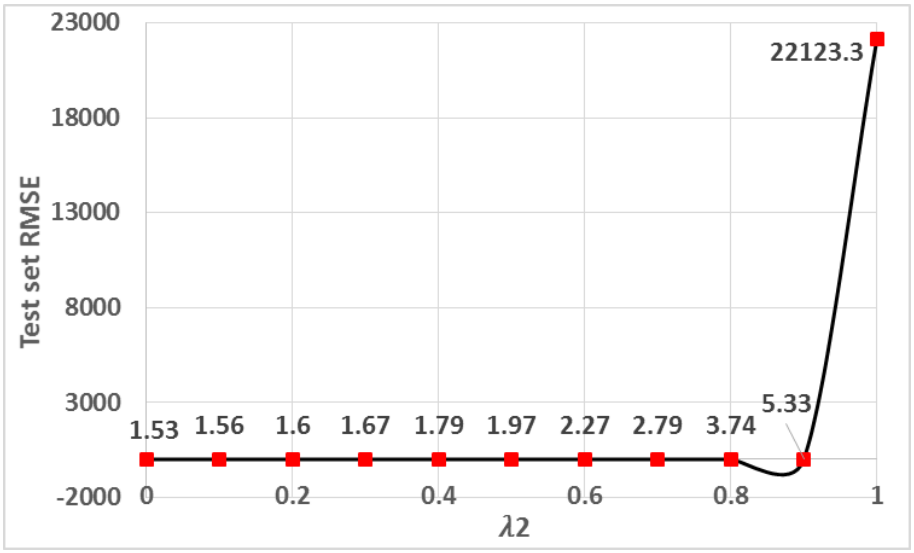

Figure 4.3: Test RMSE versus the weight value of the regularization function in simulation Example 1.

Example 2: In Example 2, which is shown in Equation 3.36, 50 cases are used to train the network. The functions' maximum values are 89462.99 for the error function and 49089046.88 for the regularization function. The minimum value for both functions is zero. As in Example 1, the values of average test RMSE versus λ_2 are drawn in Figure 4.4. The average RMSE is calculated for nine test cases in this example. Again, the results show that the best test error is that obtained with no regularization (when $\lambda_2 = 0$).

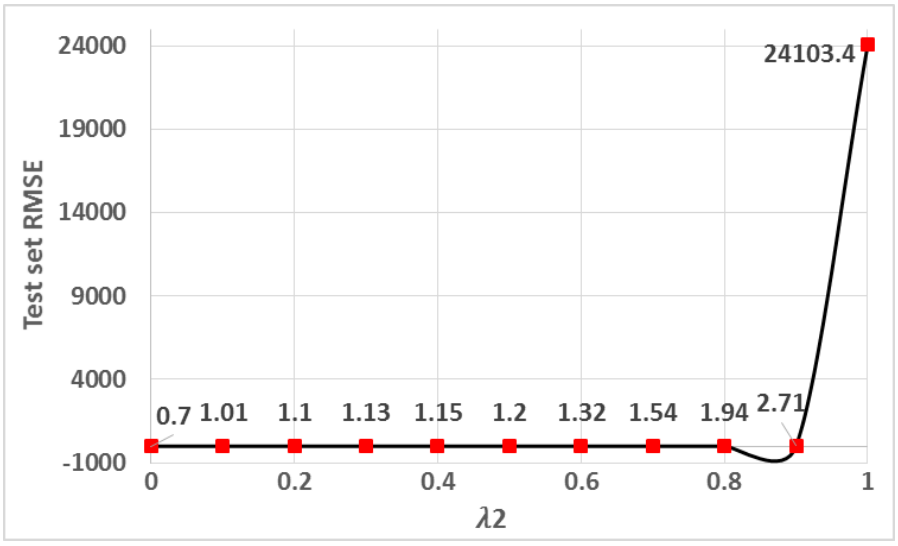

Figure 4.4: Test RMSE versus the weight value of the regularization function in simulation Example 2.

The results obtained in Examples 1 and 2 clearly indicate that, when over-fitting is assumed in the new design and the regularization function is added to the original cost function, the added function increases the test RMSEs produced from the network design. That in turn proves that with the new modification (Equation 4.4) the new Opt_RBN design has not only no over-fitting issue, but also the most generalized design.

4.4. Results

The new modified Opt_RBN is evaluated on PD applications. The new design not only enhances the computational speed of motion prediction, but also provides an insightful test case for the new network. The network is tested on two PD tasks that are created for the Santos software. The tasks are walking forward and going prone. These tasks are selected because they differ in terms of complexity and behavior. The modified

Opt_RBN performance is evaluated objectively by calculating test error and comparing it with the results of a typical RBN design (Beale, Hagan, & Demuth, 2001; Chen, Cowan, & Grant, 1991), which is the same network used to compare results provided in Section 3.3. In addition, subjective evaluation of the visual results is provided.

4.4.1. Walking forward task

The walking task is a common and basic task often discussed in the field of DHM. In this work, the task includes 42 inputs, which represent the loading conditions, joint ROMs, weapon point locations on the hands, and walking speed. The loading conditions (*16 inputs*) include the total weight of the added equipment on the back and other body segments and the segment centers of mass in three dimensions. The ROMs (*24 inputs*) include normal and reduced ranges of the upper and lower limits of the bending, extension-flexion, and rotation for four spinal joints (low, mid-low, mid-high, and high). Although other ROMs can be included as extra inputs in the task, this work includes these spinal ROMs specifically to evaluate the effect of various loadings, which are mainly added on the back, on the spinal joints under various ROMs ranges. Note that this task is performed with weapon in hand, which is also considered in the loading conditions. The task has 9 control points for each DOF, which means 495 outputs in total, and 399 training cases are collected representing various combinations of inputs. The network training time is approximately 41 minutes. The final network includes 74 basis functions.

To evaluate the network performance, five test cases are used. The test cases are the cases that have never been used to train the network. With regard to objective

evaluation, RMSE for modified Opt_RBN predicted outputs are compared with those produced from RBN, which takes approximately 3 minutes to be trained for this task. The average RMSE for the five test cases is 0.031 for the Opt_RBN and 0.04 for the RBN. Although the results are small for both models, the Opt_RBN has approximately 25% less error. Since the outputs are calculated in radians, which is the main reason for obtaining small RMSE in both models, the direct conversion to degrees would produce an error of approximately 1.78 for the Opt_RBN and 2.3 for the RBN. Reducing the error when predicting joint angles by 25% on average is an important improvement, because even errors as small as 2.3 degrees in each joint angle profile might lead to odd visual motion simulations and violate more constraints when all DOFs are predicted with the same error level.

Another objective measurement of the results' accuracy is the number of outputs each of the network models (Opt_RBN and RBN) can predict with less error than the other model (i.e., count the number of outputs with smaller RMSE for each network). Since the PD problem involves hundreds of outputs, it is insightful to measure the performance of the Opt_RBN design when predicting the problem's outputs individually. Such a measurement can be helpful in studying the general trend of the network when predicting each output. The average number of the more accurately predicted outputs among the five test cases is calculated. The calculated results show that the Opt_RBN is more accurate than RBN in this task with an average of 269 outputs compared to 217 for the RBN. Both networks have the same exact error in 9 outputs. Among the 486 compared outputs, the Opt_RBN provides less error in 52 more outputs than the RBN. The performed objective evaluations are based on the reported RMSEs for the whole

problem, and at the level of each output, a necessary conclusion is drawn regarding the outperformance shown for the modified Opt_RBN design over the typical RBN.

With respect to the subjective evaluation, the simulation results from the Opt_RBN design are evaluated visually. The RBN visual results are not included to compare against the Opt_RBD design, because the visual results from both models do not show obvious difference over the whole motion. However, the objective evaluations already provide the necessary insights on the performance of the modified Opt_RBN design compared to the RBN. Figure 4.5 shows the visual results for the Opt_RBN, where the motions resulting from the five test cases are simulated for the Santos model. In the figure, Santos moves from right to left, so the first simulation frame in each case is on the right side. In Figure 4.5, all simulated motions produced from the Opt_RBN look acceptable. The network is generally able to predict proper simulations for the corresponding provided input conditions. Specifically, the effect of heavier loads is represented in all simulations.

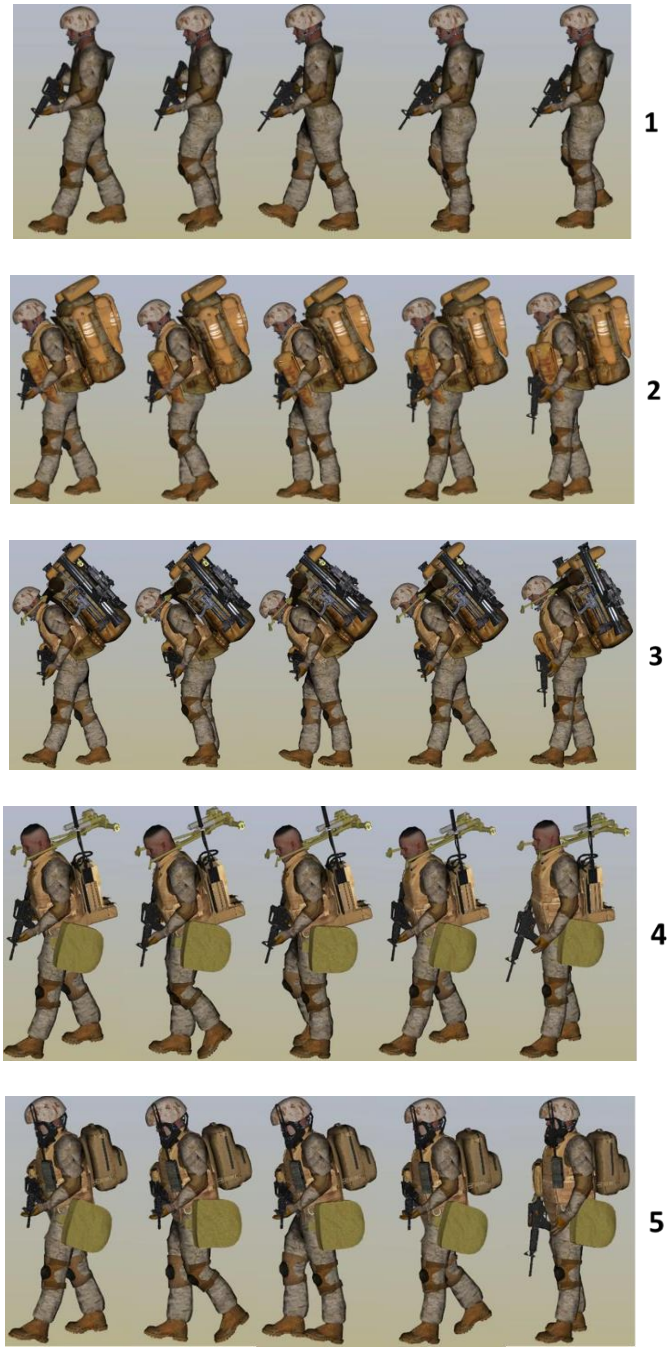

Figure 4.5: Selected key frames for walking task simulation results of test cases 1-5 using the modified Opt_RBN.

In the results of the walking task, the Opt_RBN performance is successful in providing high-fidelity outputs objectively and subjectively. The small RMSE values for

the presented test cases proves the accurate results obtained from the Opt_RBN design. The visual evaluation also shows acceptable results.

4.4.2. Going-prone task

Going prone is another task that is commonly performed by a warfighter. The task has 41 inputs, which represent the loading conditions, joint ROMs, and weapon point locations on the hands. The task has the same inputs used for the walking task, except with 1 less input (the one that represents walking speed). The going-prone task has 550 outputs, because there are 10 control points for each DOF. The task involves 306 training cases. The training process takes approximately 18 minutes, and the final network includes 38 basis functions. This task is also performed with a weapon in hand. Five test cases, which represent five different loading and ROM conditions, are evaluated for the task. Furthermore, the same subjective and objective evaluations are performed on the modified Opt_RBN results.

For the results of predicting the five test cases, the average RMSE for Opt_RBN predicted outputs is 0.018, while it is 0.026 for RBN, which takes approximately 3 minutes to be trained for this task. Similar to what is performed in the walking task, when the prediction errors are converted to degrees, the RMSE becomes 1.03 for the Opt_RBN and 1.5 for RBN. This comparison shows around 30% improvement for the results produced by Opt_RBN over those from the RBN. As mentioned, the improvement of seemingly small errors in the PD task is critical, because of the necessity for the highest possible accuracy in the nature of the simulated PD problems. Reducing the average error for each DOF from 1.5 to 1 degrees can produce a significant difference when these

errors add up for the full 55-DOF DHM in a single simulation. In general, although the simulated motions in many cases produced from the Opt_RBN and RBN might not be visually notable or differ significantly, the simulation differences could occur in many other cases, especially in cases where prediction errors are present in most of the DOFs. In addition, in complicated problems like the PD tasks, where there are hundreds to thousands of constraints to be met, any improvements in the simulated motions, even with seemingly fewer errors, means fewer violated constraints. That is especially true because none of the constraints are considered yet in the network design when predicting these simulations.

In terms of model comparison based on the number of outputs with smaller errors, the Opt_RBN design shows superiority over the RBN with an average count of 181 for the Opt_RBN versus 172 for the RBN. Even though the numbers are close, this result provides more proof that the RBN does not outperform Opt_RBN by any means. The better RMSE value presented for Opt_RBN than for RBN supports that conclusion. The reported too-close counts together with the different RMSE values indicate that both models have comparable prediction errors when RBN predicts an output with smaller errors. On the other hand, the models produce different errors, with less error for the Opt_RBN and outputs with larger errors for the RBN.

It is important to keep in mind that the new Opt_RBN introduced in this work is a design that improves the general performance, but that the improvement is significantly more obvious when fewer training cases are available. That fact is thoroughly evaluated and validated on various examples in Chapter 3. At any rate, the number of training cases used to train the networks in the going-prone task (306 cases) is not too limited for a PD

task. Other tasks might have as few as 30-40 training cases, and the Opt_RBN should also show even more significant performance than the RBN design.

Another significant result to be discussed for the network models' performance on the going-prone take is the fact that both networks produce the same error values in 197 outputs. This relatively large number of outputs with the same error values demonstrates that the going-prone task is a too constrained task, and the task shows less effects on the resulting motion over various changes in loading conditions compared to the walking task. This in turn means that the task has some joint DOFs, which are represented as a group of control points, with minor changes at various task conditions. Consequently, both compared network models are able to predict these outputs with the same accuracy level.

With respect to the subjective evaluation, the simulation results from the Opt_RBN design are evaluated visually for the five test cases of the going-prone task. Figure 4.6 provides visual representation for the simulation results produced from the Opt_RBN. It is clear that Santos performs the task by moving from the left to the right of the screen. Thus, the first simulation frame in each case is on the left side. Again, all simulated motions are acceptable and no odd results are notable. With fewer effects of various loads on the resulting motion than in the walking task, this task shows a unique strategy for handling different load configurations in each case. In general, the Opt_RBN is successful in providing high-fidelity results for the going-prone task, which involves more outputs to predict and fewer available training cases than the walking task.

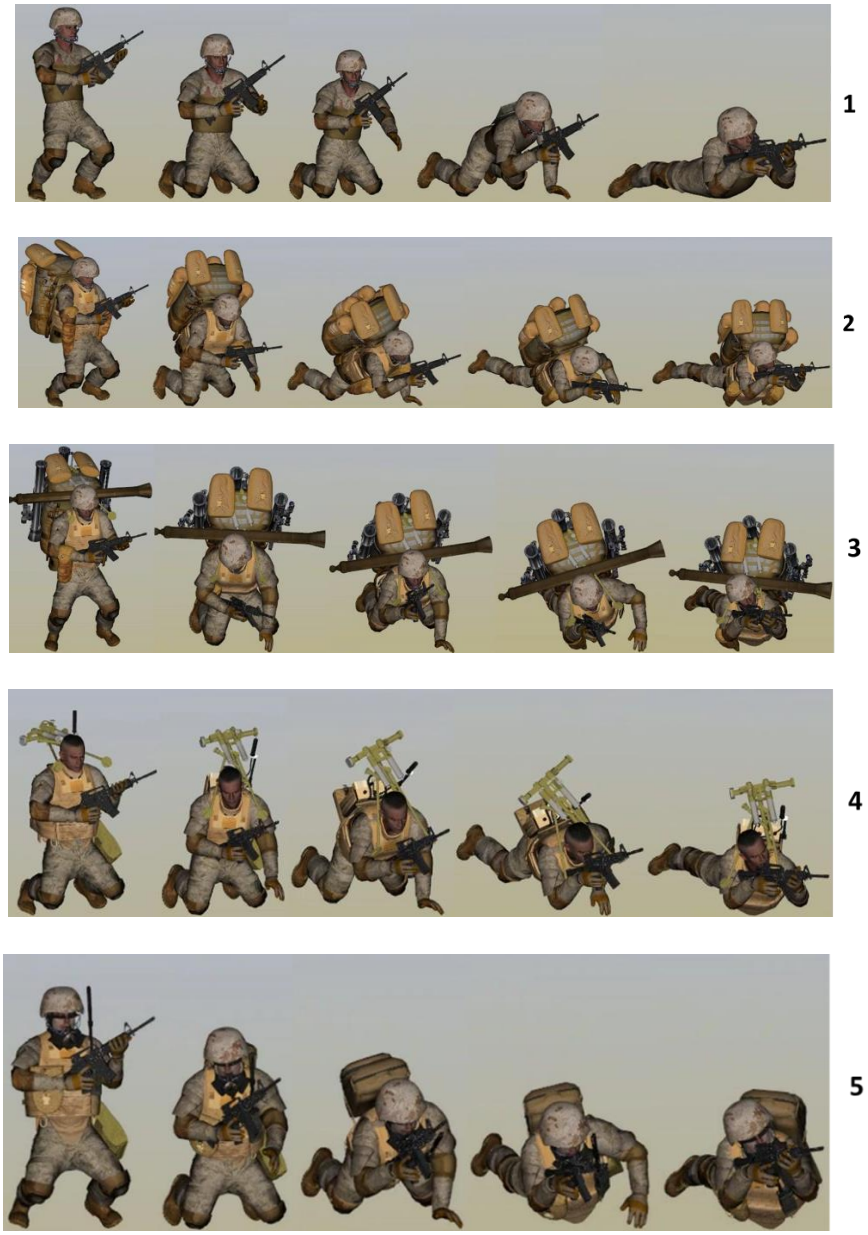

Figure 4.6: Selected key frames for going-prone task simulation results of test cases 1-5 using the modified Opt_RBN.

With both objective and subjective measurements, the performance of the new Opt_RBN in predicting the going-prone task is acceptable. The new design outperforms RBN when both are compared in terms of the RMSE and counting the number of outputs

with lower error values. Like the first task, the visual evaluation of the produced motion (i.e., outputs) from Opt-RBN is acceptable in all cases.

4.4.3. Sensitivity analysis

To check the prediction sensitivity of the modified Opt_RBN at various numbers of training cases, it is trained and evaluated with various numbers for a walking task, and its results are compared with those obtained from the RBN. Although it can be done, this sensitivity analysis does not involve the going-prone task, because currently the task does not have enough available sets of training cases to perform such an analysis. On the other hand, a results trend similar to that obtained in the walking task is expected to occur in the going-prone and other PD tasks since all PD tasks have the same fundamental problem complexity.

The numbers of training cases used in this study are 44, 132, 198, 399, 918, 1224, and 1529. The error results of predicting five test cases (the same cases evaluated in the walking task) for both networks are presented in Figure 4.7. Except for the case with 44 training cases, in which both networks show the same behavior, the modified Opt_RBN outperforms the RBN in all presented combinations of training cases. For example, when both networks are trained with 132 cases, the error produced from Opt_RBN is 20% less than that in RBN and exactly the same produced from RBN when it is trained with 198 cases. Too-close results are produced from both networks when Opt_RBN is trained with 198 cases and RBN is trained with 399 cases. These results show that the Opt_RBN can use as few as 50% of the training cases required by the RBN to produce the same prediction errors. Similar results are also produced when Opt_RBN is trained with 399

cases compared to RBN trained with 918 cases. These results are necessary indeed when the networks are trained with fewer training cases, which is the main reason for introducing the new Opt_RBN design in this work. Consequently, when applied on the PD problem, as shown for the walking task, the Opt_RBN provides better performance than the RBN when both are trained with fewer training cases. In any PD task in general, few training cases (as low as 50-100) might be the only available cases for the network to be trained to simulate that task. In that case, the use of the Opt_RBN will be most beneficial.

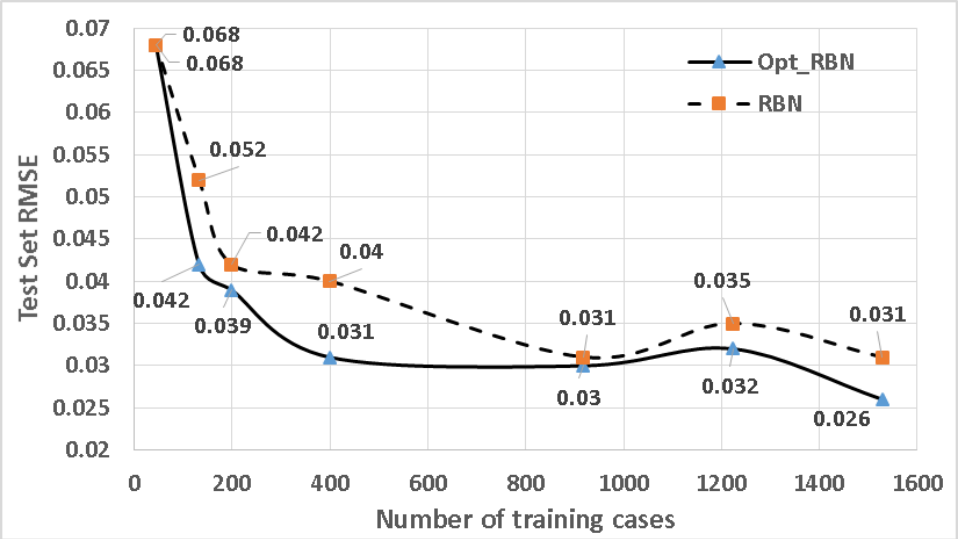

Figure 4.7: Test set RMSE evaluation for the modified Opt_RBN and a typical RBN design at various numbers of training cases.

In Figure 4.7, although the minimum RMSE that is reached by RBN and Opt_RBN when they trained with 1224 cases could be not significantly different (the RBN error is 0.031 and Opt_RBN is 0.026), the results indicate the superiority of the

Opt_RBN over the RBN even when trained with a larger number of training cases. Such results might be lower or higher for other tasks, but will indeed be significant.

In Figure 4.7, for both network models, some portion of the error curve goes up, although the network is trained with a larger number of training cases. This case may occur when the additional training cases cannot help the training process find better network design. Such a case occurred when the new Opt_RBN design was evaluated on practical problems in Chapter 3. However, this case can typically be avoided by performing a resampling on the available data, where the network is retrained multiple times with different sets of training and test cases. Consequently, smoothly decreasing test error curves should be achieved for both RBN and Opt_RBN models.

When Opt_RBN is used to predict a PD task, the presented error analysis can provide an approximation of the number of training cases needed for the network training process to produce a model with acceptable accuracy. The results in Figure 4.7 show that, after the Opt_RBN is trained with 399 cases, no significant improvement is obtained. In addition, given the large number of cases added to train the network, and keeping in mind the necessity for training with a minimal set of cases, the error reduction from 0.031 to 0.026 is negligible. In general, the performed comparison can be insightful for the PD application by providing a balance between the desired prediction error and the necessary number of training cases to be collected. The reduction in the number of training cases in the PD application is critical for saving effort, since collecting each PD case is computationally costly. Based on the results shown in Figure 4.7 for the walking task, training the network with 399 cases instead of 918, where both produce similar prediction

errors, could save approximately 55% of the time consumed in collecting the training cases.

4.5. Discussion

This work leverages the previous work in Chapter 3 to develop a modified Opt_RBN design to be used in simulating PD tasks. The original design, which is intended for applications with a reduced number of training cases, is modified to work for large-scale PD problems in terms of the number of outputs. Although the Opt_RBN design is proven to improve the prediction results for application with a reduced number of training cases, applying the Opt_RBN design on the large-scale PD application experiences some difficulty. Specifically, the Opt_RBN experiences a CPU memory issue when running the optimization step in the network's training process (Section 3.2.4) to predict all PD outputs from a single network model. Thus, the special need for the computationally expensive PD problem, and other large-scale problems in general, to produce real-time simulations provides the basic motivation of this work. Eventually, with the new modifications, the Opt_RBN can successfully predict the PD problem. Nonetheless, the new RBN design and its training process proposed in Chapter 3 should still be the typical choice when predicting any regression application with reduced training sets. The modified steps should only be used for large-scale applications similar to PD.

This chapter's contributions include: 1) a modified Opt_RBN training process for improved performance in large-scale problems with minimal training data, 2) application of the new modified Opt_RBN design for real-time prediction of PD tasks for full DHM,

and 3) construction of an RBN design that can be populated for any general large-scale problem in various applications. Although this chapter presents a modification to the Opt_RBN driven by its potential use with PD, the consequent ANN design can be used with a broad range of large-scale problems; PD is simply a well-studied example problem for the proposed developments. The new proposed ANN design can be used for general applications in various large-scale engineering and industrial fields that experience delay issues when running computational tools that require a massive number of procedures and a great deal of memory.

The modified algorithms in the Opt_RBN training steps are, first, successfully implemented. Next, the modified Opt_RBN, due to the performed modifications, is investigated on two simulation examples for a potential over-fitting issue. Then, the capability of the modified Opt_RBN is evaluated for providing real-time motion prediction of two common PD tasks, walking and going prone. In general, the results of the new network are acceptable objectively and subjectively. The objective comparisons for the Opt_RBN results with those from the RBN show superior performance by the Opt_RBN design. The reported RMSEs for both networks indicate more than 25% improvement of the new network for both presented tasks.

As another objective comparison, the new network's results are compared with RBN by counting the number of outputs for which each network has more accurate results. Although the new network outperforms the RBN in both tasks, the calculated numbers in the going-prone task are close (181 and 172 for Opt_RBN and RBN, respectively). Such results together with the reported RMSEs for that task lead to the conclusion that RBN has high errors for only a few outputs relative to the errors produced

in all outputs. That is because even though the counts are close for both networks, the RMSE refers to larger error than the counter does. Assuming the 172 more accurately predicted outputs by the RBN are comparable to the first 172 more accurately predicted outputs by Opt_RBN. Thus, the errors produced for these 172 outputs from each network cancel each other out. Then, the remaining nine more accurately predicted outputs by the Opt_RBN are responsible for the difference in the average RMSEs between both networks. Consequently, the new network is able to provide relatively too-small errors for all outputs compared to the RBN, which clearly provides poor results for some outputs.

In terms of training times, RBN took approximately 2 to 3 minutes to train for each task; the new network took approximately 16-40 minutes. Both networks run in a fraction of a second for the test cases. Given the improvements in the results and the problem sizes, the training times for the new network are acceptable. Furthermore, the training time is not as important as the run time for test cases for most practical applications.

Sensitivity analysis is performed for the modified Opt_RBN and compared with the RBN to check the performance at various numbers of training cases. Although it can be done for any PD task, this analysis is performed for the walking task only because currently it is the only task that has enough available sets of training cases to perform such analysis. Besides providing additional proof for the superiority of Opt_RBN's performance over RBN's, the analysis introduces a tool that can be used as guidance to balance collecting the proper number of training cases for the PD tasks with the design of a network with acceptable results. It will be useful in future work to thoroughly

investigate the Opt_RBN prediction capability for other PD tasks over various numbers of training cases. A selection criterion can, then, be set up for the needed training cases to be collected for different PD tasks. Moreover, it is beneficial in future work to investigate finding a proper ratio of outputs to inputs and/or number of training cases for improved ANN performance in PD applications.

Furthermore, based on the presented results for the evaluated PD tasks, which show that some task outputs have either too-minor changes or no changes over various task conditions, the ANN might help in developing new tools in the future to improve the PD task development process. For example, when a PD task is being developed, the task could have a reduced number of design variables (i.e., eliminate the unchanged outputs) or fix the unchanged DOF. That in turn would save development effort and task running time. The ANN can help in that effort by reducing the number of design variables (i.e., control points) for a task after the network prediction capability is evaluated to check for the control points with minor changes. Future work on the use of the new modified Opt_RBN design might also involve the prediction of other PD outputs like ground reaction forces on the feet, joint torque, etc.

In Chapter 5, the use of modified Opt_RBN in this chapter will be expanded to implement the constraints within the network design. Some of the PD constraints can be easily violated, even with highly predicted results from ANN because it is a regression model with approximated results. A task like ladder climbing is highly constrained because the feet and hands need to stay at specific locations over most parts of the task. Hence, multiple methods for constraint implementations within the network design will be presented and evaluated.

CHAPTER V

NEW APPROACHES FOR CONSTRAINT IMPLEMENTATION

This chapter investigates methodologies for implementing predictive dynamic (PD) motion-task contact constraints within the new radial-basis network (RBN) model presented in Chapters 3 and 4. Implementing the contact constraints is especially important in PD tasks because these constraints can easily be violated, even with highly accurate network results. Two main approaches for constraint implementation within the new RBN design are introduced and evaluated. The approaches are investigated for PD constraints, but these approaches can be applied for any problem in general. Before the approaches are illustrated, a brief background on PD constraints is provided.

5.1. Introduction

The PD motion results produced by the new modified RBN, detailed in Chapters 3 and 4, have been tested and validated. However, such motion results can violate some constraints within the simulated PD task. Therefore, this chapter works toward improving the performance of the new network design by introducing new approaches to satisfy the potentially violated constraints. Specifically, since most PD tasks include a relatively large number of constraints (on the order of hundreds or thousands), the new approaches focus on implementing the constraints that are difficult to satisfy even with the highly accurate predictions from artificial neural network (ANN) designs. Even with network-predicted motion that is very close to the final one, the PD optimization, considering the

network predicted motion as initial guess, takes more than 5 minutes, on average, to satisfy all the constraints.

With PD tasks like walking and going prone (presented in Chapter 4), there are required constraints, such as those related to the hands' locations on the carried weapon and the penetration of the feet through the ground. These constraints are referred to as contact constraints. Figure 5.1 presents an example of ANN-predicted motion for the task of jumping on a box with violated contact constraints. This PD task involves a contact constraint that is specific for its kind, in which the feet location by the end of the task should be exactly on top of the box. Therefore, constraint satisfaction checks are necessary when ANN predicts the PD tasks. That in turn allows reaching the level of using ANN outputs as the real-time, final, and reliable outputs for any predicted task.

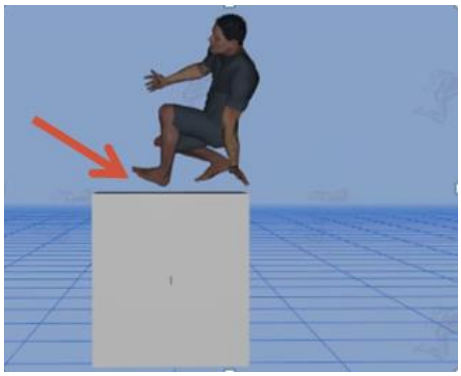

Figure 5.1: Example of ANN-predicted motion for the jump-on-box task with violated contact constraints.

Among the few works that use an ANN for digital human modeling (DHM) problems, there is one that applies the ANN to posture prediction, but with no precision for contact constraints (Bataineh, Marler, & Abdel-Malek, 2013). An RBN is used in that

work to predict the posture for touching a specific point. Even though the used network is capable of providing instant posture prediction for a full human model, the network fails to produce posture with hands exactly on the assigned target point. Consequently, even in a well-trained ANN, the contact points need to be implemented within the design to provide more accurate results with satisfied constraints.

Since typical training processes for an ANN involve unconstrained optimization (Looney, 1997; Wasserman, 1993), constraints are generally incorporated outside the network design. Scholars have introduced network models with methods for constraint satisfaction that appear under different names, such as adaptive constrained neural network, dynamic ANNs, and the hybrid ANN approach. The common limitations in these constraint implementation models are that they create slower network training and running designs, worse network performance, or a combination of both. In addition, most of these models are task-specific (see Chapter 1 for details). Furthermore, some approaches are proposed for the time-series types of ANNs. Hence, the methods cannot be directly applied for RBN. Therefore, applying the approaches to the PD problem is not feasible, because of the large-scale problem size and required highly accurate results that cannot be sacrificed.

This chapter introduces new approaches for constraint implementation within the ANN without sacrificing its prediction capability or training speed. In addition, some of the new approaches are added to the new RBN design and applied on the investigated PD problems. As an integrated part of the new RBN design (presented in Chapters 3 and 4), this work investigates two different methods for constraint satisfaction checks on the network-predicted results. One is performed only once within the network training

process, and the other is performed every time in the testing phase after the network provides its outputs. The primary goal is to modify the network-predicted outputs in order to satisfy critical constraints in the simulated PD task while considering various conditions (i.e., various combinations of inputs).

This chapter presents the following specific contributions:

1. Illustrating two new approaches for constraint implementation incorporated within an RBN. The approaches are applied to satisfy critical constraints in PD problems, but can also be used for any type of constraints in general.
2. Improving the accuracy of the new RBN-predicted outputs and motion for PD tasks.

In the remaining sections of this chapter, a general background on PD constraints is provided with an emphasis on illustration of the type of constraints that are incorporated in the proposed approaches. Next, the methods section illustrates new approaches to incorporate constraints. Their expected pros and cons are summarized. Finally, the results of applying each approach are evaluated and compared on different PD tasks.

5.2. Background: predictive dynamic (PD) constraints

In the formulation of a PD task (presented in Chapter 4), there are different kinds of constraints that include, but are not limited to, body contact points, hand locations on the weapon, joint ranges of motion (ROMs), and torque limits. The number of constraints for each type is fixed, except for the contact constraints. The number of contact constraints in a task like walking is less than that in ladder climbing, where the hands need to be on the ladder. Along with the constraints related to hand locations on the

weapon, the contact constraints between the body's hands and feet and other objects and the ground are the main equality constraints in a PD task ($h_i(\boldsymbol{q})$ ($\mathbb{R}^m$) constraints in Equation 4.1 in Chapter 4). Satisfying the equality constraints is the main reason the optimization takes a long time to finish. Satisfying these constraints is critical for accurate and visually accepted motion results. Therefore, the equality constraints, in particular, need to be implemented within the new RBN design. Although the new network does not provide a procedure to satisfy the constraints, violating the equality constraints cannot be tolerated. The inequality constraints are typically satisfied quickly, and their slight violation is tolerable since it does not generally affect the resulting motion significantly.

After the inequality constraints are removed from the PD optimization, for a simple illustration, the optimization problem presented in Chapter 4 is rewritten in Equation 5.1.

$$\begin{aligned} &\text{Find: } \boldsymbol{q} \text{ (control points for 55-DOFs)} \\ &\text{Minimize: } f(\boldsymbol{q}) = f(\boldsymbol{q} - \boldsymbol{q}_{MoCap}) + f(\textstyle\sum_1^{DOFs} joint\ torque) \\ &\text{Subject to: } h_i(\boldsymbol{q}) = 0,\ \ i = 1, \dots, m \end{aligned} \tag{5.1}$$

In Equation 5.1, $\boldsymbol{q}$ is a vector that represents the design variables (which are the control points for various body DOFs), $\boldsymbol{q}_{MoCap}$ is a vector that represents the reference motion provided by motion capture (i.e., seed motion), and $h_i(\boldsymbol{q})$ is the ith contact constraint ($\in \mathbb{R}^m$). The problem is to find the design variables $\boldsymbol{q}$ to minimize a group of human performance measures, $f(\boldsymbol{q})$, subject to the contact constraints.

Along with providing accurate network-predicted motion with satisfied constraints, the approaches should eliminate most of the aforementioned deficiencies in

the current ANN constraint implementation methods. The approaches can also be applicable to broader regression problems. The next section details the new approaches.

5.3. Method

There are two new methods proposed in this work for considering contact constraint implementation. The first method, called "constrained network design (CND)," involves imposing the constraints within the training process. The second method, called "locally adaptive network outputs (LANO)," satisfies the constraints by running an optimization that modifies the network output(s). Each method is illustrated, and its performance is evaluated.

5.3.1. Constrained network design (CND)

The first proposed approach is the constrained network design (CND) method. In the CND method, constraints are considered within the optimization problem of the new RBN training process (presented in Chapter 3). The main advantage of the CND method is that, after the network training process is complete, the network-predicted output in the test mode is instant. There are no time-consuming steps within this method to satisfy the constraints. In addition, calculating the network's design variables ($\boldsymbol{W}$) (i.e., the output weights) that satisfy the contact constraints might improve the general network capability for a simulated task.

With the CND method, the constraints are satisfied when $\boldsymbol{W}$ are found in the optimization problem of the network training process (Section 4.3). The optimization problem in Equation 4.4 becomes constrained, and the new network training process is

modified as shown in Figure 5.2. The concept is that the PD contact constraints ($h_i(\boldsymbol{W})$) need to be satisfied for the training cases in the resulting RBN design. $h_i(\boldsymbol{q})$ in the original PD problem (Equation 5.1), where $\boldsymbol{q}$ is the design variables, is presented in Figure 5.2 as $h_i(\boldsymbol{W})$, where $\boldsymbol{W}$ is the design variable in the network optimization problem (Equation 4.4).

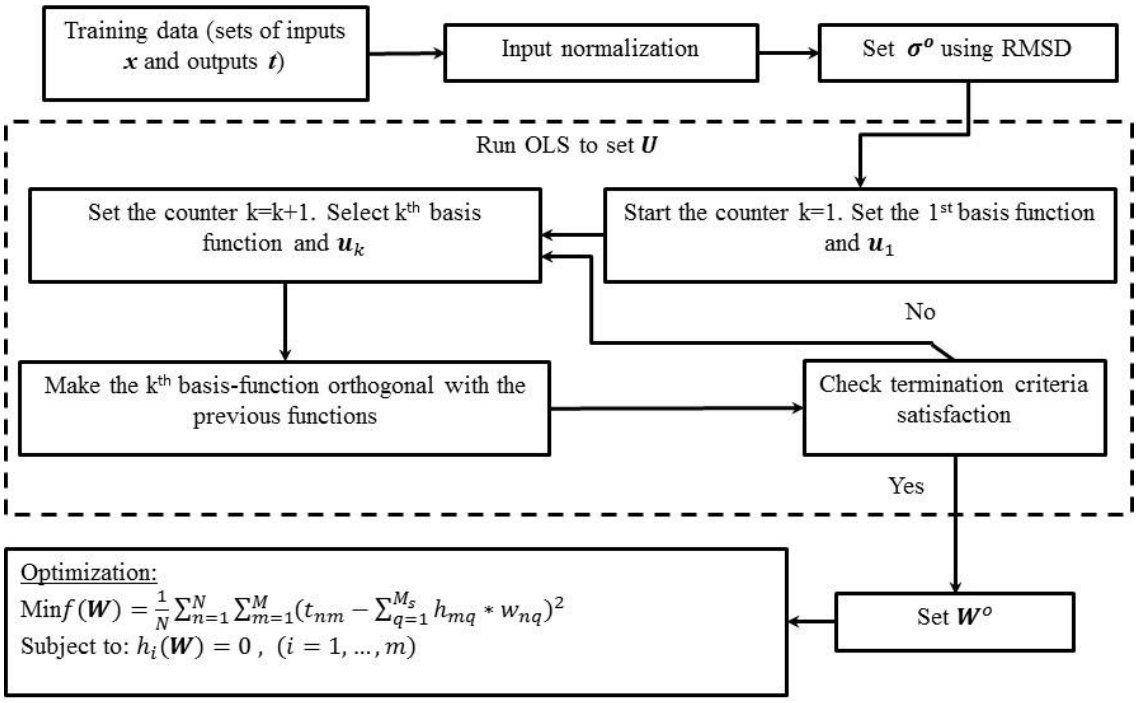

Figure 5.2: The new RBN design training process with a modified optimization step to produce a constrained network design (CND).

Conceptually, in Figure 5.2, the design variables ($\boldsymbol{W}$) in $f(\boldsymbol{W})$ and $h_i(\boldsymbol{W})$ need to be coupled. That is, there should be just one set of $\boldsymbol{W}$. The vectors of $\boldsymbol{W}$ are found to minimize the error $f(\boldsymbol{W})$ while satisfying $h_i(\boldsymbol{W})$ for all used training cases. However, there is no direct method to formulate $h_i(\boldsymbol{W})$ as a function of $\boldsymbol{W}$ while $f(\boldsymbol{W})$ depends on

the same $\boldsymbol{W}$. In other words, $\boldsymbol{W}$ forms the network's hyper-surface to predict $f(\boldsymbol{W})$, and $h_i(\boldsymbol{W})$ might be in a different space from that in $f(\boldsymbol{W})$. The network predicts $f(\boldsymbol{W})$ regardless of the violation in $h_i(\boldsymbol{W})$. Therefore, in the current RBN design, the CND method cannot be implemented.

In addition to the challenge explained above, when simulating the PD problem with constraints, assuming the CND method is implemented successfully, formulating the optimization problem as multiple problems, in order to find groups of $\boldsymbol{W}$ vectors separately (as proposed in Chapter 4), cannot be done. That is because all $\boldsymbol{W}$ vectors are dependent when the constraint is involved. Given that the PD design variables cannot be optimized in a single optimization run, due to the CPU memory issue explained in Chapter 4, the CND method cannot be applied to the large-scale PD tasks.

Since the CND method cannot be applied to a PD task like walking, other methods need to be investigated. Any new method needs to avoid any potential for memory issues when applied on large-scale PD problems. It also needs to preserve the network accuracy.

5.3.2. Locally adaptive network outputs (LANO)

With the method of locally adaptive network outputs (LANO), the network outputs are modified every time the network predicts outputs for a received input (i.e., test case). The method is called so because for every test case, the network predicted outputs are locally changed to satisfy the constraints for that specific case. Figure 5.3 presents a descriptive diagram for the method. When the trained network is provided with new inputs (i.e., new conditions) in the testing mode, it predicts the most appropriate

outputs corresponding to those inputs. Then, the contact constraints are checked for any violation. If they are violated, the outputs are modified by running an optimization that satisfies the violated constraints. If the constraints are not violated, the network's outputs are set as the final outputs. The uniqueness of the LANO method compared to many constrained ANN methods is that it can be applied to many existing network designs. That is simply because the LANO method is performed after the network provides its outputs.

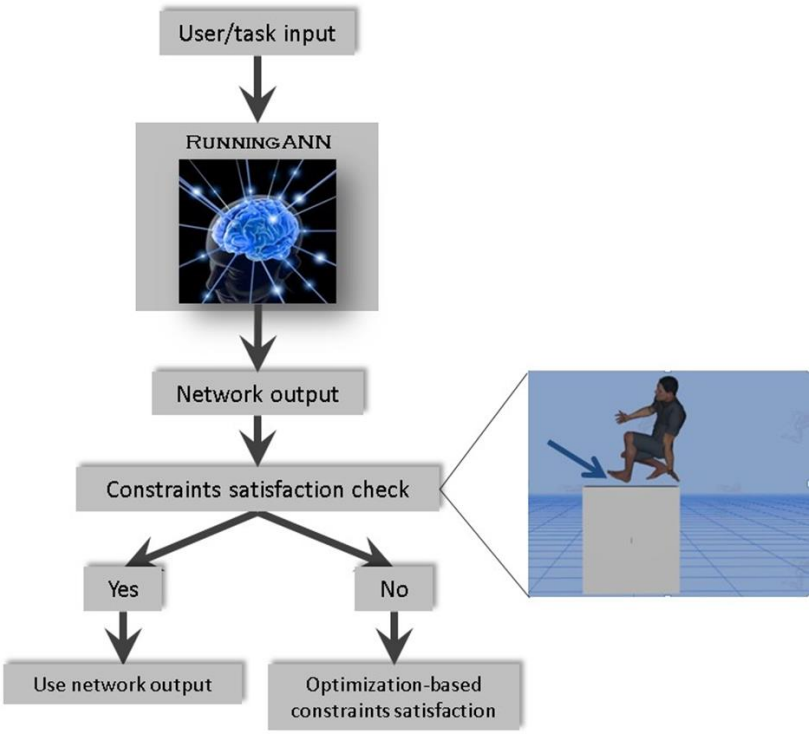

Figure 5.3: Flow chart for the steps of satisfying the violated constraints using the method of locally adaptive network outputs (LANO).

In the LANO method, the optimization-based constraint satisfaction is performed by solving the constrained optimization shown in Equation 5.2. The cost function is the

difference between the new control points ($\boldsymbol{q}$) and those produced by the network ($\boldsymbol{q}_{NN}$). The cost function modifies the control points but keeps the new motion as close as possible to that predicted by the network. All the contact constraints ($\boldsymbol{h}_i(\boldsymbol{q}(\boldsymbol{W}))$, $\boldsymbol{h}_i \in \mathbb{R}^m$) are imposed, as shown in Equation 5.2.

$$\begin{aligned} &\text{Find: } \boldsymbol{q} \\ &\text{Minimize: } f(\boldsymbol{q}) = \|\boldsymbol{q} - \boldsymbol{q}_{NN}\| \\ &\text{Subject to: } h_1(\boldsymbol{q}): \left\|\widetilde{\boldsymbol{X}}_1(\boldsymbol{q}) - \widetilde{\boldsymbol{T}}_1\right\| \le \varepsilon_1 \\ &\qquad\qquad h_2(\boldsymbol{q}): \left\|\widetilde{\boldsymbol{X}}_2(\boldsymbol{q}) - \widetilde{\boldsymbol{T}}_2\right\| \le \varepsilon_2 \\ &\qquad\qquad\qquad\qquad \vdots \\ &\qquad\qquad h_m(\boldsymbol{q}): \left\|\widetilde{\boldsymbol{X}}_m(\boldsymbol{q}) - \widetilde{\boldsymbol{T}}_m\right\| \le \varepsilon_m \end{aligned} \tag{5.2}$$

In Equation 5.2, $\boldsymbol{q}$ is a vector representing the design variables (control points for various body DOFs), $\boldsymbol{q}_{NN}$ is a vector representing the network-predicted outputs, $[h_1, h_2, \dots\dots, h_m]$ are *m* contact constraints, $[\widetilde{\boldsymbol{X}}_1(\boldsymbol{q}), \widetilde{\boldsymbol{X}}_2(\boldsymbol{q}), \dots\dots, \widetilde{\boldsymbol{X}}_m(\boldsymbol{q})]$ are vectors representing the three-dimensional positions for *m* end effectors (i.e., the points on the body that need to be in contact with some target points), $[\widetilde{\boldsymbol{T}}_1, \widetilde{\boldsymbol{T}}_2, \dots\dots, \widetilde{\boldsymbol{T}}_m]$ are vectors representing the three-dimensional positions for *m* target points (i.e., the points with which the end effectors need to be in contact), and $[\varepsilon_1, \varepsilon_2, \dots\dots, \varepsilon_m \ll 1]$ are small values that are set by the developer of the PD task (0.001, for example). The algorithm steps of the LANO method are summarized for a PD problem as follows:

Step1: Run the network, based on the fed input, and provide $\boldsymbol{q}_{NN}$ (i.e., the control points).

Step2: Calculate the values of all contact constraints $[h_1, h_2, \dots\dots, h_m]$.

Step3: Check for existence of any constraint violation:

- If yes, go to Step 4.
- If no, set $\boldsymbol{q}_{NN}$ as the final output (final motion) and terminate the algorithm.

Step4: Run the optimization in Equation 5.2 to solve for $\boldsymbol{q}$.

Step5: Set the final output motion $\boldsymbol{q}$ and terminate the algorithm.

The LANO method can be successfully applied on the PD task. As opposed to the tens of thousands of design variables (which represent $\boldsymbol{W}$) in the CND method, the LANO method includes the network outputs ($\boldsymbol{q}$), which total approximately 500-600 outputs, as its design variables. Such a number of design variables in the LANO method is relatively small compared to the CND, and the optimization can be run with no memory issues. In addition, when the cost function is set so the motion gravitates toward $\boldsymbol{q}_{NN}$, the method is expected to run in seconds to provide accurate and acceptable results along with satisfied contact constraints. Moreover, the LANO method avoids the visually unacceptable results that sometimes occur in optimization problems because the cost function forces the problem to provide results close to those of the ANN. The network results are typically known to be acceptable, if it is trained on cases that are all acceptable. Furthermore, the produced final motion based on the method always results in satisfied contact constraints.

In terms of the limitations in the LANO method, implementation of other PD constraints in using the method does not necessarily preserve its running speed. Applying the method on other applications with highly violated constraints results produced from the network affects the fast speed of the LANO method. That is because the method requires more iterations to satisfy the violated constraints. In addition, the resulting final motion with the LANO method can look inaccurate if the network provides poor predicted motions ($\boldsymbol{q}_{NN}$) for some extreme cases or cases outside its training space. In such cases, the LANO method drives the final motion to be close to $\boldsymbol{q}_{NN}$.

5.4. Results

The illustrated LANO method is evaluated in this section on two different PD tasks, jumping on a box and walking. The method's accuracy and running time are evaluated.

5.4.1. Jumping-on-the-box task

Jumping on the box is a very constrained task because the avatar's feet and hands should be at exact locations on top of the box under all task conditions (see Figure 5.1). Since the box height is an input in this task, the ANN produces accurate motions, but with obvious constraint violations in terms of the feet touching the box. The use of the LANO method is evaluated to satisfy the contact constraints for this task.

In the jumping-on-the-box task, the box height is the only input parameter, and thus few training cases are available to simulate the task. After the network is trained with three training cases, which include the box heights of 50 centimeters (cm), 70 cm, and 100 cm, the network is tested on two cases. The box height in the new test cases are 60 cm and 85 cm, respectively. For each case, the motion results of the LANO method are compared with those produced from PD and the network ($\boldsymbol{q}_{NN}$). Figure 5.4 presents the motion produced from the three models (PD, ANN, and ANN-LANO) for case 1 (height equal 60 cm). The figure includes the start frame (Frame 1), before jumping is started, middle motion frame (Frame 2), and the end frames (Frame 3), after the avatar finishes the task on top of the box.

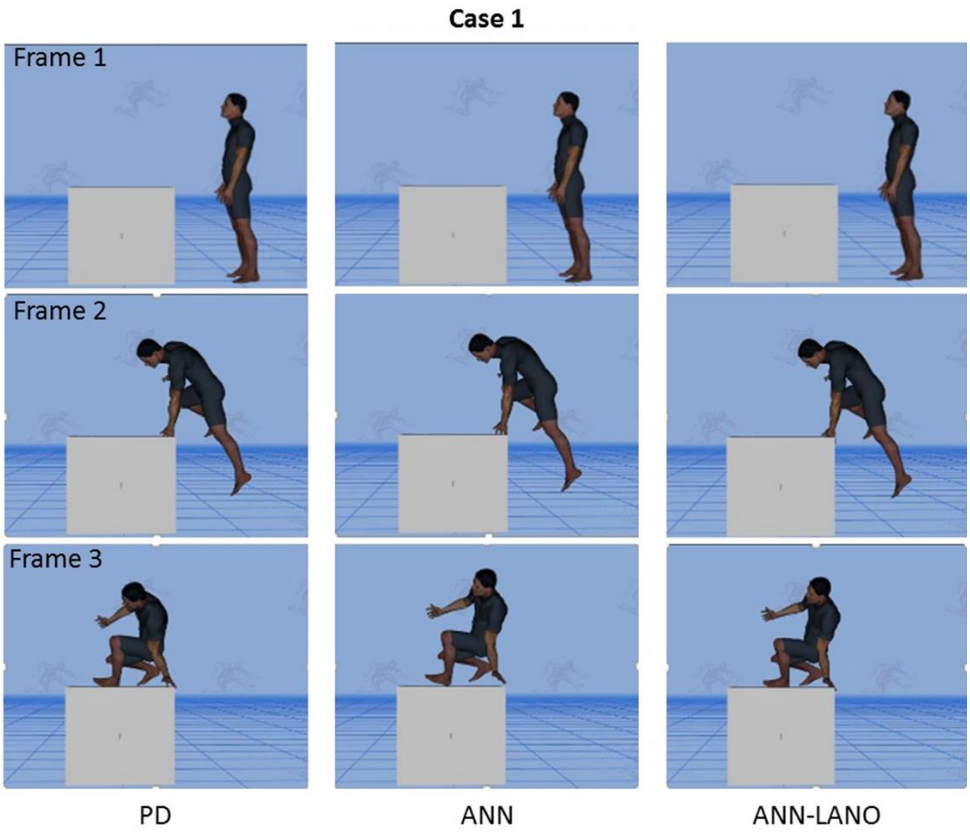

Figure 5.4: Results comparison for the motion produced from predictive dynamics (PD), the new RBN design "ANN," and the new RBN design with the locally adaptive network outputs (LANO) constraint satisfaction method "ANN- LANO" for test case 1 (height equal 60 cm) in the jumping-on-the-box task.

As in the PD result, the network with the LANO method produces motion with no violation, which is evident by the feet in frame 3 reaching the top of the box exactly. The results of the ANN in frame 3, however, show constraint violations with the right foot and left hand slightly off of the box. The LANO method successfully fixes the contact constraints that are violated in the ANN results. The running time for the method in this

case is approximately 2 seconds, which is close to real time. The PD run time is approximately 3.7 minutes, while the ANN runs in approximately real time (0.5 second).

With respect to the general resulting motion in case 1, the results of the ANN and the ANN with LANO method in frame 3 is slightly different from that produced by PD, but all results are accepted and comparable in terms of visual appearance. The prediction errors with the implemented LANO method are also small, based on visual subjective evaluation, as in the other simulated tasks in Chapter 4.

With respect to case 2, Figure 5.5 presents the results comparison for the same three models (PD, ANN, and ANN- LANO). The box height in this case is 85 cm. The general appearance of the motion in frames 2 and 3 indicates similar motions produced from the three models. More constraint violation, however, obviously occurs in the motion produced from the ANN, where the feet and left hand are away from the box. On the other hand, the ANN with LANO method provides results with satisfied constraints where the hand and feet are exactly on the top of the box. Its result also matches that produced from the PD. The running time for the method is approximately 3 seconds. The PD run time is approximately 2.7 minutes, while the ANN runs in 0.5 second. Based on the presented results, the method of LANO is effective in satisfying the violated contact constraints when applied to the jumping-on-the-box task. The optimization running speed is also fast. The LANO method is next evaluated on another common task, the walking task.

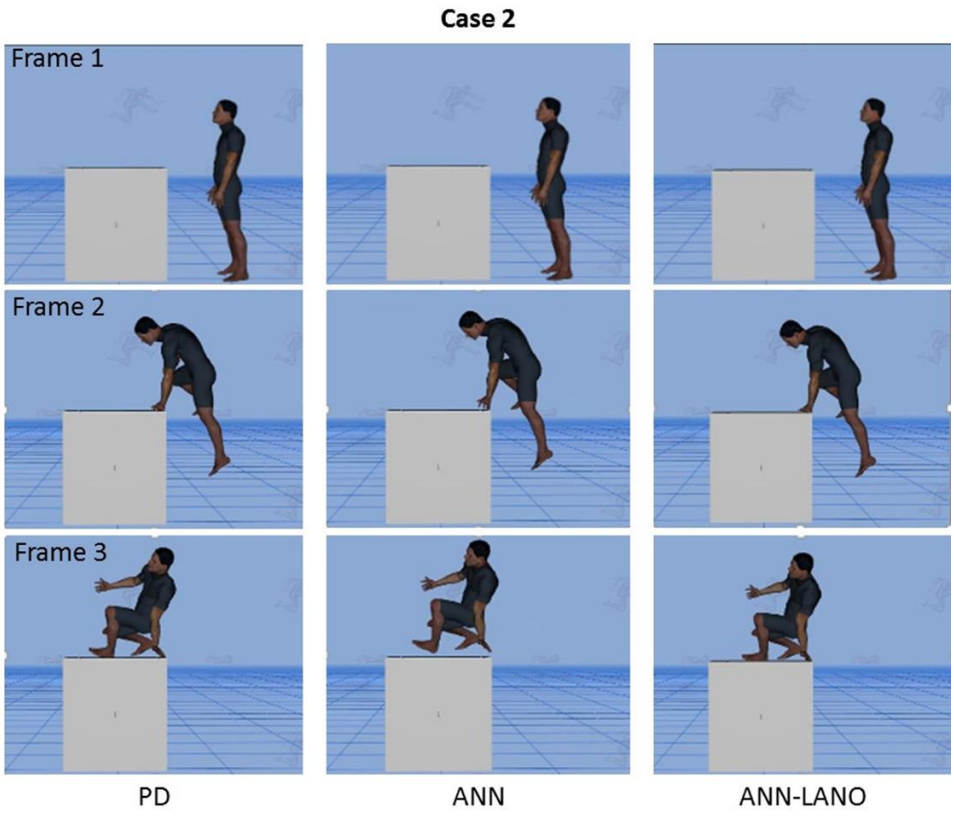

Figure 5.5: Results comparison for the motion produced from predictive dynamics (PD), the new RBN design "ANN," and the new RBN design with the locally adaptive network outputs (LANO) constraints satisfaction method "ANN- LANO" for test case 2 (height equal 85 cm) in the jumping-on-the-box task.

5.4.2. Walking task

In a task like walking with a weapon, the contact constraint violation is less obvious than that in the jumping-on-the-box task. Unlike the changeable box height, the hands on the weapon constraints have slight changes when different weapons are used. Hence, the visual constraint violation is not clear in most cases. However, the contact constraints still need to be satisfied, so the feet do not penetrate the ground and the hands

stay exactly on the weapons during the motion. The same trained network model that is used in Chapter 4 for the walking task, which includes 399 training cases, is used in this section for the evaluation of the LANO method.

When the LANO method is applied on the walking task, the optimization running time is too long. The optimization takes approximately 7.6 minutes on average when the method is applied on five test cases. Although the constraint violation with different box heights is much larger than that in the ground penetration and the hands on the weapon constraints in the walking task, the LANO optimization run time in the walking task is much larger. The reason for the long running optimization is that the gradient-based optimization method (Gill, Murray, & Saunders, 2002) spends a large number of iterations with too-small step sizes. When the initial guess is close to the final solution, the optimization usually starts with these small step sizes. In addition, the fundamental PD source code that is used to develop the walking task is different from that used to develop the jumping-on-the-box task. Depending on the implementation procedures, number of constraints, and task complexity, different versions of the PD source code can have different optimization run time when simulating the motion while satisfying the constraints. Thus, that is another reason for the long optimization run in the walking task. Therefore, other modified LANO algorithms need to be tested for the walking task to investigate faster optimization results. The primary modification involves the cost function.

5.4.2.1 Modified locally adaptive network output(s) (modified-LANO)

The difference between LANO (Equation 5.2) and its modified version (Equation 5.3) is that the modified method adds the joint torque function (from the PD cost

function) to the method's cost function. The torque function is included, because the RBN is trained using cases that are created based on PD that includes this cost function. Thus, the network-predicted motions follow the same prediction behavior that is used in PD with the torque function. Therefore, the new cost function in the modified-LANO should speed up the running speed. In the modified method, the following optimization problem is solved subject to the contact constraints:

$$\begin{aligned} &\text{Find: } \boldsymbol{q} \\ &\text{Minimize: } f(\boldsymbol{q}) = \|\boldsymbol{q} - \boldsymbol{q}_{NN}\| + f(\textstyle\sum_1^{DOFs} joint\ torque) \\ &\text{Subject to: } h_1: \left\|\widetilde{\boldsymbol{X}}_1(\boldsymbol{q}) - \widetilde{\boldsymbol{T}}_1\right\| \le \varepsilon_1 \\ &\qquad\qquad h_2: \left\|\widetilde{\boldsymbol{X}}_2(\boldsymbol{q}) - \widetilde{\boldsymbol{T}}_2\right\| \le \varepsilon_2 \\ &\qquad\qquad\qquad\qquad . \\ &\qquad\qquad\qquad\qquad . \\ &\qquad\qquad h_m: \left\|\widetilde{\boldsymbol{X}}_m(\boldsymbol{q}) - \widetilde{\boldsymbol{T}}_m\right\| \le \varepsilon_m \end{aligned} \tag{5.3}$$

Although the modified-LANO method can save running time while satisfying the contact constraints, it might experience some limitations. The used optimization, Equation 5.3, can produce final optimal motion that looks inaccurate and too different from the network seed motion ($\boldsymbol{q}_{NN}$). The reason for the different final motions is that the PD optimization is non-quadratic since it involves the torque cost function. Hence, the optimization could produce poor locally optimum solutions, which are not all necessarily accurate.

5.4.2.2 Network output(s) as initial guess (NOIG)

The network output as initial guess (NOIG) method uses the network-predicted output ($\boldsymbol{q}_{NN}$) as the initial guess (IG) and the seed motion in the optimization in order to help with faster convergence to the final optimal solution. With the NOIG method, the

optimization starts from an initial solution close to the final one, and the new motion in the cost function is gravitated toward that initial solution. The method uses the same cost function used in the LANO method (Equation 5.2) with the use of the network-predicted output ($\boldsymbol{q}_{NN}$) as the IG. Instead of having one default initial motion for all input cases, the NOIG method uses the $\boldsymbol{q}_{NN}$ as the initial motion (i.e., starting point in the optimization) for each new case.

With respect to the NOIG method limitations, the $\boldsymbol{q}_{NN}$ used as IG does not always assure running the optimization quickly, because of the nonlinear behavior of some of the constraints. As with the LANO method, even with well-predicted $\boldsymbol{q}_{NN}$, the optimization could iterate to large numbers with small step sizes.

5.4.2.3 Methods comparison

For the walking task, the modified-LANO and NOIG methods are compared along with the original LANO method. As stated, since the task-violated constraints are difficult to show visually, the methods are compared in terms of: 1) the changes in the prediction errors relative to the PD results, both root-mean square error (RMSE) and mean-absolute error (MAE), 2) running time for the optimization to provide the final motion, and 3) the amount of violation in other important excluded PD constraints. The results from all methods are compared along with those produced by the default network (without the constraint-satisfaction steps). The visual motion results are excluded from the comparison because the motions produced from all the methods look similar; the violated constraints are difficult to see.

Table 5.1 presents the comparison results for use of the LANO method and its derivatives (modified-LANO and NOIG). The presented results are the average results

for the methods when tested on five test cases. The table also includes the results from the network that is presented in Chapter 4, which has no constraints implemented in its design, as a reference for the running time and errors. To appreciate the results of all methods presented in the table, note that the average running time for the original PD problem (with all constraints considered) among the five test cases is 8.4 minutes. Even though the network-predicted outputs ($\boldsymbol{q}_{NN}$) are used as IGs, which are close to the PD final motions (based on the subjective and objective evaluations of the visual motion results shown in Chapter 4), the original PD optimization still takes a relatively long time to satisfy all constraints.

Table 5.1. Comparison results for the method of locally adaptive network outputs (LANO) and its derivatives when applied on five test cases in a predictive dynamic (PD) walking task.

Comparison method	**ANN**	**LANO**	**Modified-LANO**	**NOIG**
RMSE	0.0193	0.06	0.0233	0.0187
MAE	0.0123	0.033	0.0142	0.0119
Run Time	<1 sec	7.6 min	5.4 min	2.9 min

Although the results of the LANO method show that it slightly decreases the running time, compared to running the full PD problem, and satisfies the contact constraints, it increases the prediction errors. Compared with the errors provided by direct use of ANN-predicted outputs, which have some violated constraints, both error types are increased for the resulting motion when the LANO method is used, as shown in Table 5.1. The reason for that error increase is that using an ANN-predicted motion as the seed motion drives the optimization cost function towards that motion. Hence, the resulting final motion can differ from that obtained when the original default seed motion is used.

Although the seed motion provided by the network has slight errors compared to the final optimal one with satisfied constraints, the optimization runs for an average of 5.4 minutes for the reasons described earlier.

In Table 5.1, both modified LANO methods (modified-LANO and NOIG) show superior results over the original method. When the two modified methods are compared, the results of using NOIG show a superior advantage with significantly less running time and slightly smaller prediction errors. Along with satisfied contact constraints, the NOIG method shows error reduction over that provided directly by the ANN, because the motion results are similar to satisfied constraints produced from the NOIG method. Although the optimization running time in all presented constraint satisfaction methods takes a longer time than expected, the NOIG method is considerably the fastest. Based on the presented results, the NOIG method saves 62% and 46% of the running time of the LANO and modified-LANO, respectively. Therefore, the use of the NOIG method in the walking task is better than the others as far as the running times and prediction errors. On the other hand, since the presented LANO method and its derivatives do not run instantly in some PD tasks like walking, there is still the need to implement the constrained network design, which is illustrated in Section 5.3.1, to produce real time predicted-motions with satisfied constraints.

The results in Table 5.1 indicate that the NOIG method does not significantly improve the prediction accuracy. Given the slight error changes when all contact constraints are met with NOIG, it can be concluded that the $\boldsymbol{q}_{NN}$ with no method for constraint satisfaction has slight constraint violations. Thus, although the network does

not include constraint satisfaction steps in its design, the new ANN design (Chapter 3 and 4) provides acceptable results for the PD motion problems.

The presented LANO method and its derivatives are introduced to satisfy the contact constraints. Thus, the resulting final output (motion) is guaranteed to satisfy these constraints. However, any violations in other PD constraints are also important to monitor and evaluate for some types of these constraints. For the rest of this section, some of the excluded violated constraints are evaluated, and the performances of the constraint satisfaction methods are compared for these constraints.

The PD problem involves many types of constraints to be studied. Studying the violations in the most critical constraints can provide additional insight into the performance of the LANO methods on the excluded constraints. Among these constraints, the zero moment point (ZMP) constraints, which are responsible for the body balance, are most critical. This is because the ZMP violation is important for accurate motion results. In addition, implicit evaluation of the ZMP constraints considers the calculations of the torque limit constraints for various DOFs. More information about the ZMP constraint and its formulations can be found in the literature (Xiang, Arora, & Abdel-Malek, 2010; Xiang, Arora, Rahmatalla, & Abdel-Malek, 2009).

The ZMP constraints are evaluated for the violations produced from the new constraint methods together with that from the default ANN with no constraint checks. In the walking task, there are 64 ZMP constraints in total. For the same five test cases evaluated earlier, the modified LANO methods are compared in terms of the number of violations in the ZMP constraints, as well as the maximum violation in the violated ones (Table 5.2).

Table 5.2. Zero moment point (ZMP) constraint violations (averaged values for five test cases) produced from the modified locally adaptive network outputs (LANO) methods applied in predictive dynamic (PD) walking task.

Comparison method	ANN	Modified-LANO	NOIG
Number of Violations	11.2	11.8	11.6
Max. Violation	-0.78	-0.09	-0.11

Table 5.2 presents a summary of the violations in the ZMP constraints. Note that all ZMP constraints have the same upper and lower limits between 0 and infinity. Hence, the amounts of violation for all of them are comparable. The results show highly comparable performance for the modified LANO methods in terms of number of violations and the maximum violation. The NOIG method produces comparable violations, on average, but slightly higher maximum violation value. The results from the default ANN provide a lower number of violations and larger maximum value. Compared to those obtained from modified LANO methods, the maximum violation in the ANN is too large. Therefore, the new constraint implementation methods (modified LANO methods) help reduce the violation in one of the important inequality constraints. This trend should exist for other excluded constraints, especially the joint angle and torque limits of various DOFs since these constraints are easier to satisfy than the ZMP.

Table 5.3 presents the same evaluation, but for the individual test cases, in order to enhance the conclusion drawn about the performance of the new methods. In Table 5.3, the default ANN with no constraints shows random behavior in the number of violated ZMP constraints and their maximum values. Even with many ZMP violations, the modified-LANO method provides stable behavior in terms of producing relatively small maximum violation in all test cases. The NOIG method also shows a trend similar

to that in the modified-LANO method, which enhances the conclusion regarding the improved performance over the network model with no constraint satisfaction method. When the constraint satisfaction method is used, it satisfies the contact constraints and reduces the violations in the excluded constraints.

Table 5.3. Detailed zero moment point (ZMP) constraint violations produced from the modified locally adaptive network outputs (LANO) methods applied on five test cases in predictive dynamic (PD) walking task. Each test case represents a combination of loading and ROM conditions.

Case number	Comparison method	ANN	Modified-LANO	NOIG
Case 1	Number of Violations	2	10	7
	Max. Violation	-0.01	-0.03	-0.01
Case 2	Number of Violations	21	16	16
	Max. Violation	-0.78	-0.05	-0.05
Case 3	Number of Violations	13	15	15
	Max. Violation	-0.07	-0.03	-0.11
Case 4	Number of Violations	17	16	16
	Max. Violation	-0.54	-0.09	-0.07
Case 5	Number of Violations	3	2	4
	Max. Violation	-0.02	-0.01	-0.03

The results of the walking task show that the new modified LANO methods for constraint implementation are successful in providing satisfied contact constraints for the PD problem. The implicit correction of the produced motion when the contact constraints are satisfied allows the reduction of violations in other excluded critical constraints as well. Such reduced violations are obvious in the ZMP constraints, which involve implicit satisfaction of the joint torque limits.

5.5. Discussion

This chapter proposes two main approaches for the implementation of PD contact constraints within the new RBN design. The first one is called the CND method and incorporates constraints within the network training process. The second one is called the LANO and is applied after the network provides its outputs.

The CND method is not evaluated for any PD task, because the method requires all design variables in the large-scale PD task to be found in a single optimization run. That cannot be done since the software experiences memory issues during such a task. Therefore, the method is not evaluated on the PD task, but is expected to help reduce the constraint violations, if any exist. The method could also enhance the general network-prediction ability and improve the violation in other excluded constraints like the joint angle and torque limits of various DOFs.

The results of implementing the LANO method and applying it for the PD tasks are evaluated. Along with satisfying the constraints, the network performance (in terms of prediction accuracy) with the method is improved over that without the method. The method's success is emphasized in terms of running times and prediction errors, especially for the jumping-on-the-box task, where the method satisfies the constraints within 2-3 seconds. As shown by the slow running time when the LANO method is applied for the walking task, it is important to emphasize that the running time for the method in the presented tasks are relative. The time may differ significantly from one task to another or for various versions of the task. The task could have different versions because the task developer keeps updating the task based on user feedback to improve the

task and fix some odd results. The task's constraint limits might be relaxed so the optimization needs less running time to satisfy all the constraints.

Modified LANO methods (modified-LANO and NOIG) are, then, introduced to improve the accuracy level and running time for the walking task. Although the results of the methods vary, both show improved results in terms of running time and accuracy when applied to the walking task. In addition, the violations in the ZMP constraints, which are excluded from the optimization, are evaluated. The ZMP violations are significantly reduced for both methods, when compared to those produced from the default ANN with no constraint satisfaction procedures.

The use of the new aforementioned constraint implementation methods (CND and LANO) can be applied to all ANN designs, including those that do not allow the implementation of constrained optimization methods. Many existing ANN designs, including the new design that is developed in this work (Opt_RBN), include unconstrained optimization algorithms, and there is no direct method to change these designs to accept constraint implementation. Therefore, the constrained optimization proposed in the new methods can be modified to an unconstrained one in order to be applied in such unconstrained ANN designs. That is especially needed for the CND method, since it is applied within the network training process. Thus, the optimization in the CND method can be modified to be solved using unconstrained optimization algorithms.

The change can be performed using various methods like sequential unconstrained minimization techniques and augmented Lagrangian methods (Arora, 2004). As one of the sequential unconstrained techniques, the penalty function method

(Fiacco & McCormick, 1990). The penalty function method is simple to implement. Moreover, the method is expected to achieve fast and acceptable optimal solutions because of the provided IG for $\boldsymbol{W}$ elements found in the network training process (details in Chapter 3).

The idea of the penalty function method is that it forms a composite function $\Phi(\boldsymbol{W}, r)$, Equation 5.4, that contains the original cost, $f(\boldsymbol{W})$ in Figure 5.2, for example, and constraint functions together. The parameter r is added to penalize the composite function for constraint violation. The penalty is larger when the violation is larger. Then, the composite function is minimized using unconstrained optimization methods that are used for ANNs. The penalty r is adjusted every time after the composite function is minimized. The procedure is repeated until the function reaches its optimal point and no further improvement is possible. In other words, when there is a constraint violation, the original cost function $f(\boldsymbol{W})$ is penalized by adding positive value $P(\boldsymbol{h}(\boldsymbol{W}), r)$. The maximum constraint violation of a constraint $h_i^+(\boldsymbol{W})$ is only positive when that constraint is violated.

$$\text{Minimize: } \Phi(\boldsymbol{W}, r) = f(\boldsymbol{W}) + P(\boldsymbol{h}(\boldsymbol{W}), r) \tag{5.4}$$

$$P(\boldsymbol{h}(\boldsymbol{W}), r) = r \sum_{i=1}^{m} [h_i^+(\boldsymbol{W})]^2 ; \quad h_i^+(\boldsymbol{W}) = \max(0, h_i(\boldsymbol{W})) \tag{5.5}$$

In Equation 5.4, $\Phi(\boldsymbol{W}, r)$ is the composite (i.e., transformation) function, $f(\boldsymbol{W})$ is the cost function, $P(\boldsymbol{h}(\boldsymbol{W}), r)$ is the penalty function, $\boldsymbol{h}(\boldsymbol{W})$ is the set of constraints, $\boldsymbol{W}$ is a matrix representing the design variables (network outputs weights), and r is a scalar penalty parameter ($r > 0$). In Equation 5.5, m is the number of constraints and $h_i^+(\boldsymbol{W})$ is the maximum constraint violation ($h_i^+(\boldsymbol{W}) \geq 0$).

With the implementation of unconstrained techniques like the penalty function method to solve constrained optimization problems, the use of ANN designs for problems with constraints is broader and more applicable to new applications. In order to allow the real-time prediction with satisfied constraint from ANNs, future work need to entail implementing the CND method as an unconstrained method using the penalty method or other approaches like the Lagrangian. Successful implementation of such approaches within the ANN training process could expand the use of the ANN designs to predict applications with critical constraints to satisfy.

In future work, hybrid use of CND and LANO methods within the same network design is worth exploring. In the hybrid design, the CND method (shown in Figure 5.2) is first applied to the network training process to consider constraint satisfaction for the provided training cases. Then, the LANO method (Equation 5.2) is applied in the test mode after the network predicts its outputs to satisfy any violated constraints. In the result, while the constraints are always satisfied, the network running time with both methods implemented might be faster than that with the LANO method only. The reason for the faster run is that the constraint violations, if any, in the predicted network outputs are smaller than those without the CND method. The method already satisfies the constraints within the network design; this reduces the number of violated constraints, their occurrence, and their values when tested for new cases.

All illustrated methods for constraint implementation consider only the contact constraints (equality constraints) in PD tasks. Future work should evaluate the incorporation of other constraints and the effects on the accuracy of the final outputs and the network running time. For example, constraints like joint torque limits might require

more time to be satisfied. Therefore, applying the LANO method on other applications might not result in the same speed because of the existence of highly violated constraints produced from the ANNs. In addition, future work may investigate alternative approaches like considering the cost function to be one of the constraints, which could reduce the optimization running time. Instead of optimizing the difference between the network output and the final motion to the lowest possible value, the difference is set to a small value that can be obtained quickly. On the other hand, the heuristic small value is hard to guess for the PD application because the difference value is calculated for a relatively large number of outputs.

CHAPTER VI

ARTIFICIAL NEURAL NETWORK AS A TOOL FOR SIMULATION ANALYSIS

6.1. Introduction

In addition to providing a method for simulating motion and for enhancing predictive dynamic (PD), artificial neural networks (ANNs) can provide unique insight into the details of human motion. The calculated radial-basis network (RBN) parameters provide useful information about the task-critical training cases, inputs, and outputs. The PD motion problem presented in Chapter 4 is illustrated in this chapter as an example to investigate this hypothesis. The parameters' values, which are found when the RBN is trained on a task, can indicate the most significant training cases for the network performance, as well as the inputs and outputs with the most significant variation (i.e., the critical inputs and outputs for the network design). In this chapter, the following parameters are analyzed, and their relevance to the simulated task is detailed: the basis functions, basis function centers ($\boldsymbol{U}$), basis function spreads ($\boldsymbol{\sigma}$), output weights ($\boldsymbol{W}$), task inputs ($\boldsymbol{x}$), and task outputs ($\boldsymbol{y}$).

The work in this chapter suggests that output weights ($\boldsymbol{W}$) can be used to determine the basis functions that cause the greatest reduction in the network test error. These basis functions then indicate the training cases that have the greatest effect on network performance. That is, $\boldsymbol{W}$ and the corresponding basis functions tend to point towards the most significant training cases, and such insight can be helpful for studying

human performance. The inputs with the most change in value can be extracted from $\boldsymbol{U}$. These inputs are specified as the most dominant inputs, because the inputs with most change in value among the vectors of $\boldsymbol{U}$ have the greatest effect on the task outputs. The outputs with the most change in value, and their corresponding degrees of freedom (DOFs) can also be identified using the training cases that are used to create the network's basis functions. This chapter's primary new contribution is the use of ANN and its associated parameters as a tool for task analysis.

Section 6.2 addresses the fundamental implications of the basis function values, the basis-function centers ($\boldsymbol{U}$), the basic-function spreads ($\boldsymbol{\sigma}$), the output weights ($\boldsymbol{W}$), the task inputs ($\boldsymbol{x}$) with the most change in value, and the task outputs ($\boldsymbol{y}$) with the most change in value. Section 6.3 then uses these findings to analyze the tasks of walking, and going-prone.

6.2. Method

The network parameters are all found in the training process. In this section, we analyze the general meanings of these parameters. Methods for concise evaluation of the most significant training cases and task features (inputs and outputs) are also provided.

6.2.1. Interpretation of neural network parameters

The ANN parameters are divided into two components: the basis functions and their associated parameters ($\boldsymbol{\sigma}$ and $\boldsymbol{U}$), and the output weights ($\mathbf{W}$). Brief descriptions and sensitivity analysis for these components are provided.

6.2.1.1 Basis function and its parameters

In the new RBN design proposed in Chapters 3 and 4, the basis functions are set using the orthogonal least square (OLS) method. The method sets the basis functions using the significant training cases. In this context, the significant training cases are those that contribute the most to reducing the training error (i.e., the error in the network-predicted outputs for the training cases). Within the network design, the most significant basis functions can be determined as those that contribute the most in reducing the network testing error. Hence, knowing the most significant basis function provides a direct indication of the most significant training case in terms of the effect on the network performance (i.e., the most relative reduction in the test error with respect to the other basis functions).

In order to specify the most significant basis functions, there is a need to understand the effect of each basis function on the network-predicted output. Equation 6.1 presents the calculation of the q^{th} basis function output ($h_q(\boldsymbol{x})$). All basis function outputs ($\boldsymbol{h}$) are used in weighted sums to produce the output (y_n) in Equation 6.2. The y_n presents the n^{th} output, where all N outputs are calculated using the same $\boldsymbol{h}$, but using different vectors of output weights $\boldsymbol{w}_n$.

$$h_q(\boldsymbol{x}) = \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{u}_q\|^2}{2\sigma_q^2}\right) \quad ,[q = 1, 2, .., Q] \tag{6.1}$$

$$y_n = \boldsymbol{h} \cdot \boldsymbol{w}_n \quad , \quad [n = 1, 2, .., N] \tag{6.2}$$

The variation of y_n with respect to the q^{th} basis function output h_q (i.e., the sensitivity of $y_n$with respect to h_q) can be written as follows:

$$\frac{\partial y_n}{\partial h_q} = w_{nq} \tag{6.3}$$

Therefore, a change in y_n due to a small change δh_q in the q^{th} basis function output is given as:

$$\delta y_n = \frac{\partial y_n}{\partial h_q}\delta h_q = w_{nq}\delta h_q \tag{6.4}$$

Equation 6.4 indicates that the effect of h_q on y_n can be computed, since the w_{nq} value is known from the training process. *Among the basis function outputs (**h**), h_q with the largest w_{nq} value has the greatest effect on y_n. Consequently, that basis function is the most significant one in terms of reducing the test error in the network-predicted outputs. The training cases associated with the most significant functions are also determined as the most significant cases for the network performance.* For the simulated PD tasks, determining the most significant training cases for a task would help provide suggestions for the cases to be used for the task validation. These significant cases can be considered the typical cases for the task when it is being evaluated. Moreover, knowing the significant training cases provides recommendations for potential practical training protocols when the task is performed by real warfighters. Specifically, the fighter is trained on these significant training cases instead of many more random cases.

In order to test the relationship discussed above for extracting the most significant basis function from w_{nq}, computational experiments are run in the context of task simulations. Each basis function is removed, one by one, and the effect on test error is noted. The significance of the existence of a basis function for the network is calculated by finding the increase in the test error in the network-predicted outputs when that

function is removed. *Consequently, the functions that their removal cause the largest increase in the test error are specified as the most important ones.*

6.2.1.1.1 Basis function centers ($\boldsymbol{U}$)

The basis function centers ($\boldsymbol{U}$) are set by the OLS method as the inputs of the specified significant training cases (details are in Section 3.2.3). The OLS method inherently determines this significance based on the effects on training error. *Therefore, the vectors of $\boldsymbol{U}$, which represent sets of input $\boldsymbol{x}$, can indicate the most critical inputs. The specified most critical inputs are those for which a change causes the most change in the outputs.* Among the vectors of $\boldsymbol{U}$, the input elements with the most change in value are specified as the most critical inputs for the simulated problem. The method used to extract the inputs with the most change in value from $\boldsymbol{U}$ is detailed in Section 6.2.2.

6.2.1.1.2 Basis function spreads ($\boldsymbol{\sigma}$)

The basis function spread σ_q is part of the basis function output calculation (Equation 6.1). Within the network, each σ_q has its own effect on the calculation of the network output (y_n), which can be written as a function of basis function spreads σ as follows:

$$y_n = \sum_{q=1}^{Q} \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{u}_q\|^2}{2\sigma_q^2}\right) w_{nq} \quad , \quad [n = 1, 2, .., N] \tag{6.5}$$

The variation of y_n with respect to the q^{th} basis function spread σ_q can be written as follows:

$$\frac{\partial y_n}{\partial \sigma_q} = \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{u}_q\|^2}{2\sigma_q^2}\right)\left(\frac{\|\boldsymbol{x} - \boldsymbol{u}_q\|^2}{\sigma_q^3}\right) w_{nq} \tag{6.6}$$

Then, a change in y_n due to a small change $\delta\sigma_q$ in the q^{th} basis function spread is given as

$$\delta y_n = \frac{\partial y_n}{\partial \sigma_q}\delta\sigma_q = \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{u}_q\|^2}{2\sigma_q^2}\right)\left(\frac{\|\boldsymbol{x}-\boldsymbol{u}_q\|^2}{\sigma_q^3}\right) w_{nq}\,\delta\sigma_q \tag{6.7}$$

The change δy_n is not only affected by the direct change in σ_q, but also by the change in the distance $\|\boldsymbol{x}-\boldsymbol{u}_q\|^2$ and the output weight w_{nq}. When either $\|\boldsymbol{x}-\boldsymbol{u}_q\|^2$ or w_{nq} is zero, the change of the others does not matter, since y_n will then be zero. When the value of σ_q is zero, the y_n value is undefined and δy_n is unknown. Therefore, the value of σ_q should not be zero in order to have a finite effect on y_n, and for stable network performance. In general, the effect of σ_q on y_n can be computed after the network is trained, where all parameters are known. In Equation 6.7, the values $\boldsymbol{u}_q$, σ_q, and w_{nq} are known and constant. Then, δy_n for a change in σ_q is computed for given $\boldsymbol{x}$ in all training cases to evaluate the sum of all changes. Among the basis function spreads ($\boldsymbol{\sigma}$), σ_q with the largest sum value has the greatest effect on y_n.

The calculation of the basis function spreads ($\boldsymbol{\sigma}$) depends on the size of the simulated problem (the ratio of the number of training cases to inputs) and on the distance between neighboring training cases (as shown in Equation 4.3). The ratio of number of training cases to number of inputs is constant for a given problem. Therefore, each σ_q indicates the uniqueness of the corresponding training case in terms of location (as illustrated in the example in Figure 2.3). With respect to the distance calculation (shown in Equation 4.2), if a training case is far from the closest neighboring case (each training case represents a vector with the size of inputs in the multi-dimensional training space), then the corresponding σ_q is relatively large. However, the relatively larger σ_q

does not indicate any significance with respect to the associated basis function or practical implications for the simulated task components. Larger σ_q does not necessarily accompany the more significant basis function over the others, because the OLS method creates orthogonal basis functions within a new space that differs from the original training space. Thus, the significance of a training case and its location changes every time after the addition of new basis function. The significance of the σ_q value is also changed with the associated basis function in the new orthogonal space.

Although many basis functions generally overlap to cover the same portion of the training space, the ones with larger σs cover more area in the training space. Hence, the basis function with a relatively larger σ_q produces non-zero outputs over a larger region in the training space (i.e., with larger σ_q, the non-zero basis function output is provided within a relatively large distance from its center $\boldsymbol{u}_q$). However, that does not indicate any absolute significance of that basis function over the other ones. The basis function that covers relatively more space portion than other functions does not specify any practical implications, because that basis function can overlap with many other functions that have more contributions on the produced output. In addition, the significance of either the basis function location or the associated training case that is used to create that function cannot be drawn from the spread value in the orthogonal space.

In general, σ_q plays a significant role in the network performance to provide the proper generalization for the simulated problem (see Chapter 2 for details regarding the generalization). Larger σ_q values produce wider functions, which leads to smoother output (i.e., simpler regression surface). In such a case, an under-fitting issue might occur if the resulting model is simpler than it should be. On the other hand, smaller σ_q values

produce a narrower function, which leads to less smooth output (i.e., more complex regression surface). In such a case, an over-fitting issue might occur if the resulting model is more complex than it should be (see Chapter 2 for details on under-fitting and over-fitting). Therefore, along with the setting of other network parameters, setting σ_q is critical to produce a network model that considers the most efficient trade-off between its complexity and proper performance.

In summary, the $\boldsymbol{\sigma}$ values generally indicate the space portions that various network basis functions cover relative to each other, as well as specifying the more and less smooth portions of the training space (i.e., regression hyper-surface). Although the relative significance of σ_q for the change in y_n can be computed, the significance of σ_q does not provide definite useful conclusions regarding the task components (critical inputs, outputs, or basis functions).

6.2.1.2 Output weights ($\boldsymbol{W}$)

As shown in Equation 6.2, each weight vector in $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_N]^T$) is responsible for providing the weighted values of $\boldsymbol{h}$ to produce one of the network outputs (y_n) ($\boldsymbol{y} \in R^N$ as shown in Figure 2.1). Consequently, the value of each element (w_{nq}) in the output weight vector ($\boldsymbol{w}_n$) indicates the relative significance of its associated basis function on the change in y_n. As shown in Section 6.2.1.1, the basis function that corresponds to the largest w_{nq} value has the greatest effect on y_n. In other words, that basis function is the most significant one in terms of reducing the test error in the network-predicted outputs. *In the result, the value of w_{nq} can be used to indicate the training case associated with the most significant function as the most significant case.*

With respect to analyzing the effect of individual w_{nq} on y_n, the variation of y_n with respect to the q^{th} output weight w_{nq} can be written as follows:

$$\frac{\partial y_n}{\partial w_{nq}} = h_q \tag{6.8}$$

Therefore, a change in y_n due to a small change δw_{nq} in the q^{th} output weight is given as:

$$\delta y_n = \frac{\partial y_n}{\partial w_{nq}} \delta w_{nq} = h_q \delta w_{nq} \tag{6.9}$$

The relative effect of w_{nq} for the change in y_n is directly proportional to the basis function output h_q, which changes for every training case. Then, δy_n for a change in w_{nq} is computed for given $\boldsymbol{x}$, which changes h_q, in all training cases to evaluate the sum of all changes. With this calculation, w_{nq} with the largest sum value has the greatest effect on y_n. On the other hand, since the w_{nq} value can be in positive and negative values, determining the net change in y_n depending on the value of w_{nq} is more difficult. That is because the y_n value is produced as a weighted sum of all $h_q$multiplied by their associated weights w_{nq}. Even if the weight vectors in $\boldsymbol{W}$ are compared to extract the relative change in the outputs, $\boldsymbol{w}_n$cannot be used to indicate the relative significance of the corresponding output compared to other outputs due to the independent calculation of each $\boldsymbol{w}_n$ from the other vectors.

With respect to the network, having a relatively large w_{nq} value indicates the existence of a less smooth regression surface at the location of the q^{th} basis function in the training space. In Figure 2.5, for example, when the input is located within the area that is covered by the third basis function (h_3), y_n has a larger value compared to the

other areas. The curve that represents y_n is less smooth in that portion because of the relatively large value of the weight that corresponds to h_3. The output weights with relatively large values are generally used to represent relatively less smooth parts in the hyper-surface that simulates the problem (i.e., the more complex portions of the training space). In other words, the more complex problems require more complex models, which are mainly characterized by having less smooth regression curves. On the other hand, having extremely large values for some weights relative to the rest of them is an indication of potential over-fitting issues in the ANN model.

In summary, each element (w_{nq}) in the output weight vector ($\boldsymbol{w}_n$) can represent the relative significance of its associated basis function. Having a relatively large w_{nq} value indicates the existence of a less smooth regression surface at the location of the q^{th} basis function. Thus, removing that basis function from the network model produces the most test error, because this function has the most effect on the network design complexity. Although the remaining basis functions are significant for the network, the removal of one of them can be relatively compensated for by other functions since the design complexity does not change drastically. On the other hand, the existence of relatively extreme large w_{nq} value can indicate potential poor network performance due to an over-fitting issue.

6.2.2. Task inputs

This section provides method for determining which inputs most substantially effect simulation output. The significance of a particular input is measured by calculating its change in value (its range) among the vectors of $\boldsymbol{U}$, which represents sets of inputs for

the significant training cases for the training error (based on the OLS method). The inputs with the most change in value are specified as the ones that cause the most change in the outputs. Using Equation 6.5, the variation of y_n with respect to the i^{th} input x_i can be written as follows:

$$\frac{\partial y_n}{\partial x_i} = \sum_{q=1}^{Q} \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{u}_q\|^2}{2\sigma_q^2}\right)\left(\frac{-2x_i}{2\sigma_q^2}\right) w_{nq} \tag{6.10}$$

Then, a change in y_n due to a small change δx_i in the i^{th} input is given as

$$\delta y_n = \frac{\partial y_n}{\partial x_i}\delta x_i = \left[\sum_{q=1}^{Q} \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{u}_q\|^2}{2\sigma_q^2}\right)\left(\frac{-2x_i}{2\sigma_q^2}\right) w_{nq}\right]\delta x_i \tag{6.11}$$

In Equation 6.11, the effect of δx_i on δy_n can be computed by summing all the changes that are produced from various training cases. For all *I* inputs, $\|\boldsymbol{x} - \boldsymbol{u}_q\|$, σ_q, and w_{nq} are all the same. Thus, the change in x_i value is the only component that causes the change in y_n when calculating the sum changes among the training cases. *Therefore, the change in x_i value can be directly used to indicate the input that causes the most change in y_n.*

Among all *I* problem's inputs, this work determines the three inputs that cause the biggest change in the output. These three dominant inputs are the ones with the most change in their values over the significant training cases for the training error (i.e., over $\boldsymbol{U}$ vectors). This work refers to the changes in an input value as the *range* of the input. Equation 6.12 shows the calculated range for the i^{th} input (x_i^{range}), which is measured by calculating the absolute difference between its minimum value (x_{i_min}), Equation 6.13, and maximum value (x_{i_max}), Equation 6.14, within all *Q* training cases that correspond to the basis functions.

$$x_i^{range} = \left|x_{i\_max} - x_{i\_min}\right| \quad , \quad [i = 1, 2, .., I] \tag{6.12}$$

$$x_{i\_min} = minimum\ (x_i\ ,\ \ 1 \le i \le Q) \tag{6.13}$$

$$x_{i\_max} = maximum\ (x_i\ ,\ \ 1 \le i \le Q) \tag{6.14}$$

The maximum possible value of x_i^{range} is 2, since all inputs are normalized using standardization (Section 3.2.1). Thus, there are no inputs with a much larger range of values than the others. *Within all I task inputs, inputs with the most change in value are those with the three largest x_i^{range} among the vectors of $\boldsymbol{U}$. Consequently, the three inputs with the most change in value are referred to as the most critical inputs in terms of having the greatest effect on the simulation outputs.*

6.2.3. Task outputs

In a simulated task with multiple outputs like the PD problem, the task's outputs ($\boldsymbol{y}$) with the most change in value over various conditions (training cases) can be determined as the most critical outputs. The outputs with the greatest change in their values through various training cases are those that show the most response in the simulated problem for the change in various inputs. Hence, these critical outputs are the determinants of the problem, since they are the outputs most affected by the change in the task inputs.

The change in an output (Δy_n) value is determined by calculating the average difference between the output (y_{ni}) value and its mean ($\bar{y}$) among the Q training cases that correspond to the network's basis functions as follows:

$$\Delta y_n = \frac{1}{Q}\sum_{i=1}^{Q}|y_{ni} - \bar{y}_n| \quad , \quad [n = 1, 2, .., N] \tag{6.15}$$

The outputs with the largest Δy_n are the most critical ones for the simulated task.

Based on the aforementioned discussion, although calculating the output change Δy_n among all available training cases should be is the typical option, using only the OLS-based Q training cases to calculate Δy_n saves effort, especially when there are many training cases. The use of the Q training cases to calculate Δy_n also helps the precise extraction of the most challenging outputs for the network when being trained on a task. Since the OLS method selects the training cases that provide the maximum reduction in the network output training error, the selected training cases provide a direct indication of the output training values that are responsible for the most errors in the network-predicted outputs. Thus, the use of the most important training cases is sufficient to indicate the most critical outputs.

In the PD problem, each group of outputs (R) represents the control points that are used to create the motion profile for one DOF (see Figure 4.1). The size of R varies from one task to another. *Presenting the DOF (group of outputs) with the most aggregate change in value provides more relevant conclusions regarding the task joint behaviors than depending on the change in a single control point (one output). In contrast to the evaluation of single control point, which does not provide any information about the actual joint behavior, evaluating the DOF (where each joint consists of 1 to 3 DOFs depending on the joint location in the used human model) can indicate partial behavior of an actual joint. Therefore, evaluating the change in a DOF can provide direct insight on the change that occurs in the whole joint.* Determining the changes in the value of a

DOF involves evaluating the changes in the control points (outputs) that are used to create the motion profile of that DOF. In Equation 6.16, the change in the value for a DOF ($\Delta \mathrm{DOF_d}$) is calculated as the average of the change in the *R* outputs that represent its control points.

$$\Delta DOF_d = \frac{1}{R}\sum_{r=1}^{R} \Delta y_r \quad , \quad [d = 1, 2, .., 55] \tag{6.16}$$

By presenting the most-changing DOFs, which are those with the largest ΔDOF_d, the task behavior can be better understood. The most-changing DOFs reflect the parts of the body/skeleton that are most likely to be the task determinants. The benefits to the validation effort can be enhanced by knowing the most-changing task joints.

6.3. Results

The usefulness of the network parameters for a simulated task is evaluated in this section, with the focus on the PD problem as an example. Specifically, the biomechanical analyses of both PD tasks, walking and going-prone, are indicated. The most critical training cases are determined for each task, as well as the most changed features (inputs and outputs). Then, the extracted insights about the task characteristics from the network components are evaluated and compared to the literature-based determinants (characteristics), whenever available.

6.3.1. Example 1: walking task

Basis functions

As detailed in Chapter 4, the walking task has 42 inputs and 495 outputs, and 399 training cases are used to train the network. The trained network has 74 basis functions that are selected by the OLS method and are created based on the importance of their corresponding training cases in reducing the outputs' prediction training error. For the walking task, the list of significant training cases that correspond to the basis functions along with the function spread ($\boldsymbol{\sigma}$) values are reported in Table B.1 (Appendix B). With respect to the task training cases, the most significant ones are specified using the basis functions. The most significant basis function is specified as that correspond to the largest w_{nq} value. *This is can be validated by evaluating the effect of removing each basis function on the resulting testing root-mean square error (RMSE) for the network-predicted error. Then, the basis functions that their removal produces the maximum increase on the test error are specified as the most important ones, since their removal produces the worst network performance.* Consequently, the corresponding training cases that are used to create these significant basis functions are specified as the most critical cases. For the walking task, which has RMSE equal to 0.03, Table 6.1 illustrates the three most critical basis functions. The produced RMSE when each one of these basis functions is removed is also provided in the table, as well as the corresponding training cases.

Table 6.1: The three most critical basis functions for the network prediction error in the walking task, and description of the corresponding training cases.

Largest RMSE	Basis function number	Corresponding training case (number)
0.59	5	No load, small ROMs, minimum speed (78)
0.56	54	Middle load, small ROMs, maximum speed (143)
0.52	65	Middle load, large ROMs, maximum speed (133)

Based on the reported results in Table 6.1, the most critical training case is case number 78. It is important to note that the inputs in this case have extremely small values in the training space in terms of the loading (i.e., no load), including joint range of motions (ROMs), and walking speed (see Chapter 4 for the details of the included inputs). The second case is number 143, which represents small values in terms of the ROMs, middle values for the loading, and maximum values for the walking speed. The third case, number 133, represents middle load values, large ROMs, and maximum walking speed. In general, the reported three critical training cases represent combinations of the task inputs that are located at different locations within the training space.

The critical cases for a task can be used as typical training cases in the real-life training protocols. Further, the cases can be used to improve the effort of biomechanical validation of a PD task or ANN-based simulation to assure the task is being built properly. When a new PD task is developed, its prediction success is validated by comparing the task results of some conditions with those produced by a real human being. The human performs the task under the same conditions provided to the PD and is recorded by motion capture systems. Instead of the random selection of these evaluated cases, the significant training cases provide a new reliable platform for the cases that need to be used in the validation process.

With respect to the output weights (w_{nq}), the three that have the largest absolute values are reported to compare against the drawn significant basis functions. In each w_{nq}, the n^{th} component represents the output number, while the q^{th} component represents the basis function number. The three largest absolute values are found to correspond to $w_{481,5}$ (the value equal 4.91), $w_{68,5}$ (the value equal 4.49), and $w_{71,5}$ (the value equal 3.27). When compared to the basis functions that are reported as the most significant ones, Table 6.1, all three w_{nq} correspond to the 5^{th} basis function. That function is reported as the most significant function for the network. Thus, the conclusion regarding the potential use of the w_{nq} to represent the significance of its associated basis function is satisfied in the walking task. If this conclusion is proved further on other PD tasks, the effort of removing each basis function to evaluate its significance for the network design can be saved by reporting the w_{nq} with the maximum absolute values.

With respect to the functions with the largest spread (σ) values, there are six basis functions that have the same σ value, which equal to 3.8. These functions are number 13, 33, 52, 57, 68, and 71. As reported in Table 6.1, none of these basis functions is considered one of the most significant functions for the network performance. Therefore, the σ value is not useful in indicating the importance of the associated basis function to the network design.

Inputs

Among all 42 inputs, which include the body link masses and their three-dimensional centers of mass to represent the various loading conditions on the back, head, and hand, the reduced ROM for some body DOFs, and the speed, this work specifies the three most critical inputs. Table 6.2 summarizes these three dominant inputs,

which are the three inputs with the largest x_i^{range} values (Equation 6.12). The input with the most change in value corresponds to the body segment (i.e., link) that represents the upper part of the neck's center of mass in the y-dimension. Basically, the value of the neck's center of mass in the *y*-dimension is changed when different equipment is added to the avatar's head, such as a helmet, night vision glasses, etc. The input with the second most change in value is the link that represents the lower neck's center of mass in the x-dimension. This input is changed when different loadings are carried on the back, such as a backpack, armor, radio, etc. The third input presented in Table 6.2 is the mass of the link that represents the lower part of the neck. In general, this input is directly related to the second presented input; both are changed together.

Table 6.2: The three inputs with the most change in value in the walking task.

Input ranking for most change in value	**Input range value (x_i^{range})**	**Input name**
1	1.35	Upper neck- y center of mass
2	1.18	Lower neck- x center of mass
3	0.94	Lower neck mass

Ideally, the most obvious input to change in showing the effect (response) of a changed condition in a PD task is when the loading is changed. In the Santos software, various types of equipment and loadings are attached mainly to the links that represent the upper and lower neck. Thus, the results obtained in Table 6.2 for the most critical inputs prove that notable behavior in the simulated motion task.

Outputs

With respect to the task outputs, the most changed ones are determined. Based on the value Δy_n calculated in Equation 6.15, the change in value output is evaluated for all task outputs among the Q training cases that correspond to the network's basis functions. Then, the change in value of a DOF (ΔDOF_d) is calculated (Equation 6.16), which represents the average of the change in the control points associated with the DOF. Table 6.3 presents the three DOFs with the most change in value among the simulated 55-DOF human model. The detailed ΔDOF_d values for all 55 DOFs are presented in Table B.3 (Appendix B). Eventually, the reported DOFs with the maximum change in value can refer to the biomechanical determinant joints for the simulated task.

Table 6.3: The three degrees of freedom (DOFs) with the most change in value in the walking task.

DOF ranking for most change in value	**DOF change in value (ΔDOF_d)**	**DOF name (number)**
1	0.06	Spin_Low Extension/Flexion (Q2)
2	0.054	Right_Knee Extension/Flexion (Q39)
3	0.043	Spin_MidLow Extension/Flexion (Q5)

In Table 6.3, the indicated maximum change of value is experienced in the one called "Spine Low-Extension/Flexion" (known as Q2 in Santos). The second maximum change presented in "Right_knee Extension/Flexion" (Q39) is followed by the one that represents the "Spine Mid-Low-Extension/Flexion" (Q5). These presented results for the DOFs with the most change in value in the walking task strongly match those indicated in the literature for the determinant joints. Depending on the location, each joint represents either one, two, or three DOFs. The knee joint is widely known as one of the

determinants in the walking task [1, 2]. When there is a load applied on the back, the spine joint is indicated as a determinant joint [3].

The three DOFs with the minimum ΔDOF_d values are presented in Table 6.4. The minimum value, which is zero, is reported for the "Right_Shoulder Rotation Internal/External" (Q17). This DOF experiences absolutely no change over the various task conditions because the right hand is constrained to always attach the weapon and carry it to the front side of the body. Thus, shoulder rotation in the internal/external direction is not allowed in the walking task. Thus, fixing this DOF for the task will not affect the resulting motion. The DOF with the second least change in value is reported for the "Lower_Neck Right/Left Rotation" (Q33), which has a very small value (0.002). It is obvious that this DOF is not allowed to be changed freely because none of the configurations can cause neck rotation in the walking-forward task. The third reported DOF in Table 6.4 is for the "Left_Elbow Flexion/Extension" (Q27). The left elbow is constrained to be attached to the front side of the weapon in this task. Consequently, the presented three DOFs in Table 6.4 can be eliminated or fixed in this task without affecting the whole resulting motion under various loading and configuration conditions.

Table 6.4: The three degrees of freedom (DOFs) with the least change in value in the walking task.

DOF ranking for least change in value	**DOF change in value (ΔDOF_d)**	**DOF name (number)**
1	0	Right_Shoulder Rotation Internal/External (Q17)
2	0.002	Lower_Neck Right/Left Rotation (Q33)
3	0.014	Left_Elbow Flexion/Extension (Q27)

In general, the three DOFs with the maximum change in value that are reported in this task are helpful indications for the task determinant joints, which involve the knee and spine along with other joints. Further, the DOFs with minimum change in value in this task are those that belong to joints that are either constrained in the task by being attached to the weapon or those on which the simulated motion has minimal effect. These DOFs can be fixed or eliminated from the walking task without affecting the simulation results. Thus, the PD algorithms for this task, and the corresponding ANN model, can be modified to have fewer design variables and faster optimization runs.

With respect to the output weights (w_{nq}), the three that have the largest absolute values are reported to compare against the drawn critical DOFs. The three largest absolute values are found to correspond to $w_{481,5}$ (DOF# Q48), $w_{68,5}$ (DOF# Q1), and $w_{71,5}$ (DOF# Q1). When compared to the DOFs that are reported as the most critical ones, Table 6.3, none of the three reported w_{nq} corresponds to any of the outputs that represent the critical DOFs.

6.3.2. Example 2: going-prone task

Basis functions

In the going-prone task, which is detailed in Chapter 4, there are 41 inputs and 550 outputs, and 306 training cases are used to train the network. The trained network has 38 basis functions. For the going-prone task, the list of significant training cases that correspond to the basis functions along with the function spread ($\boldsymbol{\sigma}$) values are reported in Table B.2 (Appendix B). The most significant task training cases are specified using the basis functions. The basis functions that produce the maximum RMSE for the

network-predicted error when they are removed are presented in Table 6.5. The corresponding training cases for these functions are also described in the same table. When all basis functions are included, the produced RMSE equals 0.019.

Table 6.5: The three most critical basis functions for the network prediction error in the going-prone task, and description of the corresponding training cases.

Largest RMSE	Basis function number	Corresponding training case (number)
0.89	6	Minimum load, maximum ROMs (263)
0.57	14	Middle load, minimum ROMs (41)
0.48	13	Small load, middle ROMs (162)

In Table 6.5, based on the aforementioned classification, training case number 263 is selected as the most critical one. This case has minimum values in terms of the loading and maximum values in terms of the included ROMs. The second case is number 41, which represents middle values in terms of loading and minimum values in terms of the ROMs. The third case, number 162, represents small values in terms of loading and middle values in terms of the ROMs. The reported three critical training cases, again, represent combinations of the task inputs that are located at different parts of the training space.

With respect to the output weights (w_{nq}), the three that have the largest absolute values are reported to compare against the drawn significant basis functions. The three largest absolute values of w_{nq} are found to correspond to $w_{163,6}$ (the value equal 5.81), $w_{74,6}$ (the value equal 5.41), and $w_{53,6}$ (the value equal 4.62). Similar to the trend shown in the walking task, all three w_{nq} correspond to the same 6^{th} basis function, which is reported as the most significant function for the network (Table 6.5). *Based on the results*

shown in both analyzed PD tasks, walking and going-prone, it is proven that the absolute value of w_{nq} can to be used in provide definite determination of the most significant basis function for the network prediction capability. Specifically, w_{nq} can indicates the basis function that produces the maximum reduction in the network testing error.

With respect to the basis function spread (σ) value, the three basis functions that have the largest σ values are reported. The three largest σ values are found to correspond the basis functions number 16 (σ=7.66), 11 (σ=7.2), and 13 (σ=5.57). In Table 6.5, only the basis function number 13 is reported among the most significant functions for the network performance. Along with the results reported in the walking task, this result conforms the insignificance of the σ value in reflecting the importance of the associated basis function to the network design.

Inputs

With respect to the inputs with the most change in value for the going-prone task, the three most dominant ones among the 41 inputs are illustrated in Table 6.6. These results are presented based on the calculated x_i^{range} in Equation 6.12. This task involves the same type of inputs as in the walking task, except for the walking speed. The input with the most change in value corresponds to the lower part of the neck's center of mass in the x-dimension. The second one belongs to the lower part of the neck's center of mass in the y-dimension. The first two most critical inputs are changed together when the back loading is changed. The third critical input is the center of mass (y-dimension) of the link that represents the upper part of the neck. As in the walking task, the going-prone task's most critical inputs correspond to those generally affected by changing the loading conditions.

Table 6.6: The three inputs with the most change in value in the going-prone task.

Input ranking for most change in value	Input range value (x_i^{range})	Input name
1	1.98	Lower neck- x center of mass
2	1.5	Lower neck- y center of mass
3	1.35	Upper neck- y center of mass

As with the results obtained in the walking task, the inputs with the most change in value in the going-prone task indicate that the most critical inputs to change to show the effect (response) of a changed condition in a PD task are the loading inputs. The simulated motions in this task are affected the most by changing the inputs reported in Table 6.6, which correspond to the change in loading applied on the back.

Outputs

Table 6.7 illustrates the results of the three DOFs with the most change in value in the simulated 55-DOF human model in the going-prone task. These results are presented based on the calculated ΔDOF_d in Equation 6.16. The detailed ΔDOF_d values for all 55 DOFs are presented in Table B.4 (Appendix B). In Table 6.7, the DOF with the maximum change of value is the one for "Left_Clavicle elevation (shrug)" (Q22). If the task-simulated motion in Chapter 4 is recalled, the left hand is responsible for touching the ground and holding the body before the body ends up prone on the ground. As the main joint that holds the body mass in this case, the clavicle handles most of the body mass when the hand touches the ground. That joint is the most affected one, especially under various loading conditions. The DOF with the second maximum change is the "Left_Ankle Dorsi plantar/Flexion" (Q47). The Q47 DOF, which represents part of the ankle joint, is strongly affected by the loading conditions because all body segment

masses are combined in single force at the ankle. The third one represents the "Global_Rotation Right/Left" (GR3). The global rotation is included in this task since the task requires the rotation of the whole body to be changed from the standing posture to prone on the ground.

Table 6.7: The three degrees of freedom (DOFs) with the most change in value in the going-prone task.

DOF ranking for most change in value	DOF change in value (ΔDOF_d)	DOF name (number)
1	0.044	Left_Clavicle elevation (shrug) (Q22)
2	0.038	Left_Ankle Dorsi plantar/Flexion (Q47)
3	0.037	Global_Rotation Right/Left (GR3)

In Table 6.7, all three reported DOFs present parts of the joints that are most likely to be the most affected ones (i.e., the task key joints) by the task various conditions. Although there has been no literature yet that objectively evaluates the task joint determinants, the presented DOFs with the most change in value in this work mimic the general behavior of the task. The global rotation, for instance, is the main notable characteristic in this task. That in turn supports the results obtained in this work.

The three DOFs with the minimum ΔDOF_d values are not reported for the going-prone task. Unlike the results in the walking task, the going-prone task involves 17 DOFs with absolutely no change in ΔDOF_d over the various task conditions. These DOFs include: Q1, Q3, Q4, Q6, Q7, Q9, Q10, Q12, Q33, Q34, Q35, Q36, Q38, Q41, Q43, Q45, and Q48. Such results indicate that the task is very constrained relative to the walking task. Thus, fixing or eliminating these DOFs for the task will not have any effect on the resulting motion. The reported results for the going-prone task show that the

aforementioned idea of reducing the number of design variables in the PD task can be extremely efficient for very constrained tasks like going-prone. Since many unchanged DOFs are included in the optimization problem when simulating the new going-prone motion, the optimization will be faster when fewer design variables need to be found (i.e., when the unchanged DOFs are removed from the problem).

With respect to the output weights (w_{nq}), the three that have the largest absolute values are reported to compare against the drawn critical DOFs. The three largest absolute values of w_{nq} are found to correspond to $w_{163,6}$ (DOF# Q11), $w_{74,6}$ (DOF# Q02), and $w_{53,6}$ (DOF# GR3). When compared to the DOFs that are reported as the most critical ones, Table 6.7, only the w_{nq} corresponds to the GR3 is reported. In summary, the results shown in both analyzed PD tasks, walking and going-prone, prove that the value of w_{nq} cannot be used to extract practical indications regarding the task critical outputs.

6.4. Discussion

This chapter presents a novel application for the new RBN design as a tool for simulated task analysis. The values of the network parameters, which include the basis functions, their parameters (the centers ($\boldsymbol{U}$) and spreads ($\boldsymbol{\sigma}$)), and output weights ($\boldsymbol{W}$), are shown to be useful in providing feedback about the tasks. In addition, useful suggestions are extracted for the PD task development and validation processes, as well as recommendations for practical training protocols for the simulated tasks. In general, the new methods introduced in this chapter show that the parameters of a trained ANN can be used to study human performance and extract unique characteristics of a simulation task that are not provided by any other prediction tools. For each presented PD task, the

critical training cases, inputs, and outputs are specified. Hence, the dominant characteristics of a task can be determined.

The chapter presents a unique opportunity to develop a new tool for biomechanical analyses of the simulated PD tasks using the ANN's parameters. It is found that output weights ($\boldsymbol{W}$) can be used to determine the most critical basis functions that cause the greatest reduction in the network test error. Then, the critical basis functions can specify the most significant training cases that are responsible for the proper performance achieved by the network. The inputs with the most change in value can be extracted from $\boldsymbol{U}$ in order to determine the dominant inputs. The outputs with the most change in value and their corresponding key body degrees of freedom (DOFs) for a motion task can also be specified using the training cases that are used to create the network's basis functions.

Determining the most significant training cases for a task would help provide suggestions for the cases to be used for the task validation. These significant cases can be considered the typical cases for the task when it is being evaluated. In addition, the task development and optimization running time might be improved, because the significant training cases can replace the randomly selected seed motions in the optimization cost function (see Chapter 4 for details on the PD optimization problem). Along with the potential benefits that are gained by studying the task's significant training cases, the cases help investigate the most critical task input(s), as well as indicate the potentially most important outputs. That is especially needed for problems like PD tasks because they involve prediction of a relatively large number of outputs (in the order of hundreds).

Among all PD inputs, which include 41 and 42 inputs for the walking and going-prone tasks, respectively, the body link masses and their three-dimensional centers of mass to represent the various loading conditions on the back, head, and hand, the reduced ROM for some body DOFs, and the speed in the walking task, this work finds that the PD tasks' most critical inputs are those representing the back and head loadings. Although having various combinations of other inputs is critical for successful task simulation, the focus should be mainly on having more varied combinations of the most critical inputs. Specifically, most training cases need to involve various equipment sizes and masses, and with symmetric and asymmetric distribution to change the centers of mass of different body links. In contrast to changing all task inputs in the same manner, the less-critical inputs can be kept fixed or minimally changed, and the dominant inputs are changed when creating new conditions to be added to the task training library. With these considerations, a more organized approach might be developed for the best combination of inputs when the training cases are collected for a new PD task. Consequently, fewer training cases are used for the same or improved network design, and less time is consumed in collecting the training cases. This trend of having fewer training cases with more efficient combinations of inputs that produce similar ANN prediction results can enhance those results produced in the example shown in Figure 4.9. In that example, adding more training cases was not helpful in all cases because of the random collection of the added cases.

The proposed approach of depending on the calculation of the change in value of the output (Δy_n) to extract the associated DOFs with the most change in values as an indication of the critical key joints for the presented PD tasks is shown to be effective.

The results from both tasks were promising since the joints specified as the potential key joints match both what is found in the literature and the task's natural biomechanical behavior. The main benefit of knowing these key joints for a task is that it provides the biomechanics community with a tool to extract the determinant joints for a new motion task. Instead of evaluating all body joints, the biomechanical analysis is only performed on the determinant joints. That in turn saves time and effort.

In future work, the results regarding the task-related features can be investigated further and validated. Other PD tasks might also be studied. Furthermore, the training cases indicated as the significant ones might replace the original seed motion to evaluate their effects on the optimization running times. The network performance could also be evaluated after being retrained using new sets of training cases with a focus on collecting the cases with more variations among the reported key inputs. Moreover, the new network parameter analyses that are applied on the PD motion tasks might be populated not just over other digital-human-modeling problems but for other applications as well. Further, although the presented DOFs with the maximum and minimum changes in values provides valuable biomechanical insight about the simulated tasks, these DOFs do not completely reflect the joints with maximum and minimum changes. Depending on its location in the body and the simulated human model, each joint consists of single or multiple DOFs. Hence, a more general evaluation of the key joints can be performed in future work by combining the changes in values of the DOFs that represent the same body joint.

CHAPTER VII

DISCUSSION

The main focus of the thesis work is the development of new artificial neural network (ANN) algorithms for real-time dynamic motion prediction of a digital human model (DHM). Various challenges related to the motion problem specifications are overcome, and new ANN approaches are proposed to overcome the existing network limitations. This chapter provides a summary of the new approaches and contributions. General discussion and conclusions are then provided for the new algorithm's evaluation when applied on experimental and practical problems. Finally, potential ideas and open issues are illustrated for future research work.

7.1. Summary

Artificial neural networks have been used successfully in various practical problems. Though extensive improvements on different types of ANNs have been made to improve their performance, each ANN design still experiences its own limitations. In general, typical ANN models experience limited performance when applied in applications with large-scale size, limited available training data, or both.

The existing DHMs are mature enough to provide accurate and useful results for different tasks and scenarios under various conditions. There is, however, a critical need for these models to run in real time, especially those with large-scale problems like motion prediction. Predicting motion problems can be computationally demanding. It takes time (minutes to hours) to predict a single task, even with small changes in the task

conditions. Hence, this work addresses that need using a new radial-basis network (RBN) design that is capable not only of providing highly accurate real-time motion results, but of providing them with minimal training.

The RBN is a powerful type of ANN to be investigated for predicting highly complicated problems like DHM motion. The RBN has been selected in this work because of its advantages when predicting regression problems and its fast and successful training process, especially in large-scale applications. The RBN design, however, has some limitations. Therefore, this work presents a new algorithm for the development of a new RBN design to address the needs. Specifically, this means creating a single network design that provides improved prediction of large-scale motion problems, even with a reduced number of training cases. The design incorporates multiple training stages with approaches to facilitate automation of the whole training process with minimal heuristics. Methods for constraint implementation within the new design are also provided. In addition, the design is used as a new tool for simulated task analysis.

In the current ANNs, there are gaps in the state of the art with respect to prediction models for large-scale problems. In this context, the term “large-scale” refers to the ratio between the number of outputs and the number of inputs and/or training cases. Specifically in DHM motion prediction problems, the large-scale problem could be referred to as the number of outputs. There is a critical need, especially in the field of DHM, to develop a single model that is capable of providing instant realistic motion prediction of full-body DHM.

The RBN is the fundamental ANN model that is used in this work, and its various training techniques are illustrated in Chapter 2. That chapter also details the current RBN

design deficiencies, which mainly include: 1) generally poor performance when predicting large-scale problems like DHM motion, 2) limited prediction capability when the network is trained with minimal available training cases, and 3) network parameters setups that involve many heuristics, especially the basis function centers ($\boldsymbol{U}$) and spreads ($\boldsymbol{\sigma}$).

All proposed opportunities for the aforementioned deficiencies are considered in new methodologies presented in this work. The work develops a new RBN design that outperforms other models, especially when used for large-scale problems with fewer training cases. To that end, this work pursues the following objectives:

1. Design a new ANN to provide robust and improved performance with large-scale problems with a reduced number of training cases.
2. Develop training approaches that facilitate automation of the training process with involvement of minimal heuristics.
3. Use the new ANN design to provide accurate real-time prediction of motion tasks for full DHM under various task conditions.
4. Develop new methods for constraint implementation within the ANN design to improve the results and satisfy the critical constraints.
5. Use the ANN as a new tool to extract useful biomechanical information about the predicted DHM tasks.

Based on the presented research objectives, this work develops new methodologies for designing an RBN with an optimal training process to improve the network performance (i.e., the most possible accuracy in the predicted network outputs) for different applications. The improved performance is especially applicable for those

applications with reduced available training data like the motion prediction of DHM. Based on the RBN training process, the new design involves multi-stage training techniques for determining all necessary network parameters.

The new RBN design outperforms other typical ANN and regression models. The new design in this work is conceptually similar to the knowledge-based neural network (Towell & Shavlik, 1994), which is a hybrid learning method for an ANN, but the algorithms and training procedures used in the designs are different (details provided in Chapter 1). Furthermore, this work proposes and evaluates new approaches for constraint implementation that are conceptually similar to those proposed in the literature but that might not affect the results produced from the new RBN design. In addition, although the evaluation is not successful in all predictive dynamic (PD) tasks, it is shown that the new approaches can relatively maintain the speed of ANN calculations when applied on some PD tasks. Unlike the existing methods, the new approaches can be applied for any task and any type of constraint.

Other applications and analyses in the DHM predicted motion problems are performed in this work by using the new RBN design to provide insights regarding the inherent parameters in the predicted tasks. The design is used to draw biomechanics feedback about the predicted task. Questions regarding the most critical training cases and significant task inputs and outputs can be addressed. Moreover, this work focuses on implementing the new RBN in the Santos software environment to be automatically trained and run to provide real-time motion prediction, and to possibly be populated for the prediction of other problems. Some work (Ishu, van Der Zant, Becanovic, & Ploger, 2004) pointed out the need for automatic methods to run the ANN quickly; this requires

careful selection and modification of the approach incorporated in the new RBN design. Such requirements are considered in the algorithms used in the new design to facilitate automation of the whole training process with minimal heuristics.

With the new developments detailed in the previous chapters, this thesis made the following contributions:

1. Improved RBN performance when predicting a task with any number of training cases.
2. Introduced a modified orthogonal least squares (OLS) algorithm for determining basis function centers (***U***) and initial connection weights (***W***). Unlike the original OLS method, the modified method has better termination criteria that assists in setting more proper number of basis functions for improved network design. The initial ***W*** values can speed up the coverage of the optimization problem when calculating the optimal ***W***.
3. Integrated the OLS approach and an optimization-based approach.
4. Introduced objective calculation of all necessary network parameters that are task independent.
5. Proposed a modified design for the newly developed RBN design, called Opt_RBN, for improved performance in large-scale problems with minimal training data.
6. Applied the new modified Opt_RBN design for real-time prediction of predictive dynamic (PD) tasks for full DHM.
7. Illustrated two new approaches for constraint implementation to be incorporated within the new RBN design.
8. Improved the accuracy of the new RBN-predicted outputs and visual motion.

9. Used the ANN and its associated parameters as a tool for simulated task analysis. Analysis of the network parameters can lead to indication of the critical task inputs, outputs, and training cases.

7.2. Conclusion

The main contribution introduced in this work is the development of a new RBN design, called Opt_RBN, which is detailed in Chapter 3. The new RBN design overcomes the poor performance of ANNs when used in applications that have limited numbers of training cases available. The new design "Opt_RBN" was tested on four experimental problems, and the results were compared with those from three models: linear regression, feed-forward back-propagation network (FFN), and RBN. The results showed that Opt_RBN outperforms the other models in all examples. In addition, the results of comparing the new design with the RBN at different numbers of training cases suggested better response for the Opt_RBN to produce the smallest possible error. Then, Opt-RBN was evaluated and compared with RBN on two real-world regression problems. In general, Opt_RBN evaluation showed substantial outperformance when trained with fewer training cases. The network showed stable performance when trained with fewer training cases for all presented problems.

The new double termination criteria in the OLS method and the quadratic cost function used in the optimization step (see Chapter 3) guarantee that the Opt_RBN design demonstrates high robustness and stability to provide improved performance. The robust and stable behaviors of the new design are illustrated by its results in all presented

examples, as well as when the design is evaluated with multiple training and test samples in the experimental problems.

The new design proposes a smarter ANN that is capable of improved learning rather than needing more training data. The new design opens new fields for the use of ANNs in applications with limited numbers of training data, such as digital human modeling. The design is introduced with a focus on improving the prediction ability for a unique problem, which is the regression problem with reduced available training sets. The use of training algorithms with minimal heuristics takes the new RBN design to a higher domain of prediction quality that none of the competing methods have achieved. That prediction quality is facilitated by the use of OLS to set the inputs of the significant training case, which are selected from all available training data, as $\boldsymbol{U}$, and the optimization to find the optimal $\boldsymbol{\sigma}$ and $\boldsymbol{w}$ values.

The new Opt_RBN design has some limitations, which include: 1) the design outperformance compared to other competing models could decline when more training data are available because the design might experience over-fitting issue when being trained using more training data, and 2) when the design is being simulated to predict large-scale problem in terms of the number of outputs (more than 200 outputs), the design experiences a CPU memory issue when running the optimization step in its training process (Section 3.2.4).

Chapter 4 leveraged the work on development of the new RBN, Opt_RBN, for PD applications. The original proposed Opt_RBN design is modified to work for large-scale PD problems in terms of the number of outputs. Although the Opt_RBN design improves the prediction results for application with a reduced number of training cases,

applying the new design on the large-scale PD application experiences some difficulty. The Opt_RBN experiences a memory issue when running the optimization step in its training process (Section 3.2.4) to predict all PD outputs from a single network model. Nonetheless, the new RBN design and its training process proposed in Chapter 3 should still be the typical choice when predicting any regression application with reduced training sets. The modified steps should only be used for large-scale applications similar to PD.

The modified Opt_RBN design was successfully implemented, and its performance was investigated on mathematical simulations. Then, its capability to provide real-time prediction of two common PD tasks, walking and going prone, was evaluated. The evaluation results of the new network were acceptable objectively and subjectively. Compared to typical RBN design, the Opt_RBN had relatively small prediction errors for approximately 500-700 outputs from a single network model. Although the presented modification to the Opt_RBN is driven by its use with PD, the consequent ANN design can be used with a broad range of large-scale problems; PD is simply a well-studied example problem for the proposed developments. The new proposed ANN design can be used for general applications in various large-scale engineering and industrial fields that experience delay issues when running computational tools that require a massive number of procedures and a great deal of memory.

Although the new network training process took longer than that in the typical RBN design, both networks run in a fraction of a second for the test cases. Given the improvements in the results and the problem sizes, the training times for the new network

are acceptable. Furthermore, the training time is not as important as the run time for test cases for most practical applications.

To check the prediction sensitivity of the modified Opt_RBN at various numbers of training cases, it was trained and evaluated with various numbers for a walking task, and its results were compared with those obtained from the RBN. Besides the superiority of Opt_RBN's performance over RBN's, the analysis introduced a tool that can be used as guidance to balance collecting the proper number of training cases for the PD tasks with the design of a network with acceptable results. If Opt_RBN is used to predict a new PD task, the performed error analysis at various numbers of training cases could provide an appropriate approximation of the number of training cases required to achieve the necessary accuracy—specifically, the balance (i.e., trade-off) between the proper prediction errors and number of necessary training cases. That in turn could be critical for saving effort, since collecting each PD case is computationally costly. As an example from the presented walking task, training the network with 399 cases instead of 918, where both produce similar prediction errors, could save approximately 55% of the time consumed in collecting the training cases.

The PD motion results produced by the new modified RBN in Chapter 4 were highly accepted and accurate. However, such motion results can violate some constraints within the simulated PD task. Therefore, Chapter 5 worked toward improving the performance of the new network design by introducing new approaches to satisfy the potentially violated constraints. Specifically, since most PD tasks include a relatively large number of constraints (on the order of hundreds or thousands), the new approaches focused on implementing the constraints that are difficult to satisfy even with the highly

accurate predictions from ANN designs. For example, when different weapons are used for the same avatar and task, the hands' locations might be off from where they should be on the weapon. In some other tasks like jumping up on a box and climbing stairs, hands and feet are expected to be at specific locations at specific times over the motion profile.

There were two main proposed approaches for the implementation of PD contact constraints within the new RBN design. The first one was called the constrained network design (CND) method and incorporates constraints within the network training process. The second one was called the locally adaptive network outputs (LANO) and is applied after the network provides its outputs. The details of both approaches were discussed, as well as their advantages and limitations. The CND method was not evaluated for any PD task, because the method requires all design variables in the large-scale PD task to be found in a single optimization run. That cannot be done since the software experiences memory issues during such a task. Therefore, the method was not evaluated on the PD task, but is expected to help reduce the constraint violations, if any exist. The method could also enhance the general network-prediction ability and improve the violation in other excluded constraints like the joint angle and torque limits of various degrees-of-freedom (DOFs).

The results of implementing the LANO method and applying it for the PD tasks were evaluated. Along with satisfying the constraints, the network performance with the method was improved over that without the method. The method's success was emphasized in terms of running times and prediction errors, especially for the jumping-on-the-box task where the method satisfies the constraints within 2-3 seconds. On the other hand, the LANO method experienced slow running time when it was applied for the

walking task, because the method's running time differs from one PD task to another or for various versions of the task. The task could have different versions because the task developer keeps updating the task based on user feedback to improve the task and fix some odd results. In order to solve the method's running issue, two modified LANO methods were proposed. The methods were shown to improve the accuracy level and running time for the walking task.

The use of the new constraint implementation methods (CND and LANO) can be expanded to be applied for all ANN designs, including those that do not accept the implementation of constrained optimization methods. Many existing ANN designs include unconstrained optimization algorithms and cannot be modified to include constraints. Therefore, the constrained optimization can be changed to an unconstrained one to apply the new presented methods in such designs. That was especially needed for the CND method since it is applied within the network training process. To address that need, a method to change the CND constrained optimization problem to an unconstrained problem was illustrated.

In Chapter 6, the calculated known RBN parameters were used to analyze the characteristics of the simulated PD tasks. A novel application for the new RBN design as a tool for simulated task analysis was presented. The network parameters, which mainly included basis functions, basis function centers ($\boldsymbol{U}$), and the output connection weights ($\boldsymbol{W}$), were shown to be useful in providing feedback about the studied tasks. In addition, useful suggestions were extracted for the task development and validation processes, as well as recommendations for practical training protocols of the simulated PD tasks. On the other hand, although the relative significance of the basis function spread (σ_q) for the

change in the output (y_n) can be computed, the significance of σ_q does not provide definite useful conclusions regarding the task components (critical inputs, outputs, or basis functions).

The basis functions that cause the greatest reduction in the network test error, which can be determined using $\boldsymbol{W}$, are used to specify the most significant training cases for the proper network performance. Knowing the most significant training cases would help provide robust suggestions on the cases to be used for the task validation and specify the real-life cases to be produced in the motion capture laboratory. Moreover, by providing feedback about a task's general behavior, the significant cases can have a great impact on improving the task implementation. Furthermore, knowing the significant training cases provides recommendations for potential practical training protocols when the task is performed by real warfighters. Specifically, the fighter is trained on these significant training cases instead of many more random cases. Along with the potential benefits that are gained by studying the task's significant training cases, the cases help investigate the most critical task input(s), as well as indicate the potentially most important outputs. That is especially needed for problems like PD tasks because they involve prediction of a relatively large number of outputs (in the order of hundreds).

The task inputs with the most change in value can be extracted from $\boldsymbol{U}$ in order to determine the dominant inputs. The simulated PD walking and go-prone tasks were presented as examples; their results showed that the most significant inputs for the network-predicted outputs were those representing the back and head loadings. Therefore, the recommendation for those tasks was to focus on collecting the training cases that involve various equipment sizes and masses, and on symmetric and

asymmetric distribution to change the centers of mass of different body links. With these considerations, a more organized approach might be developed for the best combination of inputs when the training cases are collected for a new PD task. With the focus on having cases with more variations for the important inputs, the performance of the trained network would be improved.

The last part of Chapter 6 focused on extracting the most critical outputs, which are indicated as those with the most change in their values. For the presented PD tasks, the average change in the values for each group of outputs that represent one DOF are also calculated. Hence, the DOFs that are associated with the greatest average changes are extracted in order to indicate the critical key joints for the PD tasks. The results from both tasks were promising since the joints specified as the potential key joints match both what is found in the literature and the task's natural biomechanical behavior. The main benefit of knowing these key joints for a task is that it provides the biomechanics community with a tool to extract the determinant joints for a new motion task. Instead of evaluating all body joints, the biomechanical analysis is only performed on the determinant joints. That in turn saves time and effort.

7.3. Future work

The successful use of the new RBN design for PD problems presented in this thesis could be expanded to study more DHM-related tasks. For example, the presented application of predicting the stresses on the knee joint (see Chapter 3) might be expanded to other joints with consideration of extra inputs and outputs. The new RBN design can

also be applied for new challenging large-scale applications within and beyond the DHM field.

With the completion of the new RBN design in this thesis, some pitfalls might arise for the use of the design for problems beyond PD. Although the design performance was validated on both experimental and practical problems, it is common for predictive models like ANN to experience difficulties in producing comparable high-quality results when applied on other practical applications. That is necessary indeed because each problem has its own characteristics. For instance, a problem might be smaller and less complicated than the PD, but its available training data may be more distorted and noisy. Investigating methods to clean the training data in such a case is more critical than which ANN model is used. Specifically, future work would include the following:

1. Improving the proposed network training process to be completely heuristics-free. In the new network design, the new termination criterion, which is the main sensitivity parameter in the network, still involves heuristics by setting the tolerance values. Other statistical criterions like computing the variance of the residuals among the training data could be incorporated and analyzed (Chen, Hong, Harris, & Sharkey, 2004; Kerschen, Worden, Vakakis, & Golinval, 2006; Walter & Pronzato, 1997).
2. Investigating the use of the new modified Opt_RBN design to involve the prediction of other PD outputs like ground reaction forces on the feet, joint torque, etc.
3. Evaluating the new RBN design capability when predicting other PD problems with various numbers of training cases. Then, general selection criteria can be drawn for the optimal number of training cases to be used to train the ANN in new PD tasks. In addition, the optimal number of training cases can be specified for the inputs-to-

outputs ratio. The same protocol might be followed for the use of ANN in other DHM applications.

4. Evaluating the proposed modified Opt_RBN design for other large-scale practical problems that experience delay issues when running tools that are computationally expensive in terms of time and CPU memory.
5. Improving the modified Opt_RBN design to achieve better prediction results for the PD outputs that were predicted with the highest errors. As initial steps, these outputs can be checked for whether they are the same outputs in different tasks and conditions or are random.
6. Incorporating other PD constraints in the network and evaluating the results in terms of the optimization running times and resultant motion quality. In addition, alternative constraint implementation approaches might be investigated, such as considering the cost function to be one of the constraints, which could reduce the optimization running time. Instead of optimizing the difference between the network output and the final motion to the lowest possible value, the difference is set to a small value that can be obtained quickly.
7. Expanding the use of the LANO method for constraint implementation towards other PD tasks. The method was shown to be effective in terms of the results and fast running time. On the other hand, implementation of other constraints using this method does not necessarily preserve its running speed. Constraints like joint torque limits might require more time to be satisfied.

In Chapter 6, the new novel study of connecting the network parameters with their implications on the task critical components could greatly enhance future work for both

the ANN and DHM communities. That is because the ANN is typically used to provide predictions without analyzing its parameters and means to the simulated task. In addition, the DHM field is still developing and requires more effort to understand the general human behaviors for the effort of modeling new tasks and validating existing ones.

Based on the presented tool for the use of ANN in simulation analysis, the new extracted results regarding the task-related features can be investigated further and validated. Other PD tasks might also be studied. Furthermore, the training cases indicated as the significant ones might replace the original seed motion to evaluate their effects on the optimization running times. The network performance could also be evaluated after being retrained using new sets of training cases with a focus on collecting the cases with more variations among the reported key inputs. Moreover, the new network parameter analyses that are applied on the PD motion tasks might be populated not just over other DHM problems but for other practical applications as well.

As a long-term goal, based on the presented results for the evaluated PD tasks in Chapter 4, which show that some task outputs have either minor changes or no changes over various task conditions, the ANN might help in developing new tools in the future to improve the PD task-development process. For example, when a PD task is being developed, the task could have a reduced number of design variables (i.e., eliminate the unchanged outputs) or fix the unchanged DOF. That in turn would save development effort and task running time. The ANN can help in that effort by reducing the number of design variables (i.e., control points) for a task after the network prediction capability is evaluated to check for the control points with minor changes. Moreover, the potential

long-term work from this thesis is and its related issues are proposed in the following points:

1. Investigating alternative optimization approaches to solve the problem of CPU memory when optimizing the network parameters to simulate large-scale problems. Unlike the PD problem, in which the parameters can be divided to be found in multiple optimization runs, other problems might not allow for such division.
2. The use of ANN as a total replacement of the PD algorithm when predicting a PD task with no violated constraints, and under any conditions and scenarios. The PD algorithm is complex and not only predicts the joint control points, but also calculates other outputs like joint torques, which are critical to be found. To solve such an issue, multiple ANNs can be trained to provide various types of outputs. Once the PD calculates the joint control points, it takes less time to calculate all other outputs. That is always true because calculating the other outputs is dependent on the control points. Hence, prediction of the other outputs should be less challenging.
3. Considering the time step, which represents the order of the control points (outputs) in the predicted motion profile, as input to reduce the ratio between inputs and outputs. For example, when the task involves 6 control points for each DOF, the network is trained to run 6 times, where only the time step input is changed, to provide the 55 by 6 outputs for the whole motion profile. This approach also produces 6 training cases from each existing training case, which might be helpful.
4. Expanding the use of ANNs to provide feedback about the simulated PD problems. The ANN might be involved in the detection of failed cases. Since the PD problem is too complicated and nonlinear, the optimization sometimes reaches locally optimal

solutions that satisfy all the constraints but might visually look strange. During the task development process, the ANN can detect the failed case produced from the optimization algorithm of the task if the case is within the training grid. In such cases, the detected failed cases can be inspected to fix the task formulations.

BIBLIOGRAPHY

Abdel-Malek, K., Arora, J., Yang, J., Marler, T., Beck, S., Swan, C., . . . Rahmatalla, S. (2006). *Santos: A physics-based digital human simulation environment.* Paper presented at the Proceedings of the Human Factors and Ergonomics Society Annual Meeting.

Arora, J. (2004). *Introduction to Optimum Design*: Academic Press.

Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *Information Theory, IEEE Transactions on, 44*(2), 525-536.

Bataineh, M. (2012). Artificial neural network for studying human performance. *M.S. thesis, University of Iowa, 2012.*

Bataineh, M., & Marler, T. (2015). Neural network for regression problems with reduced training tets. *Neurocomputing, under submission.*

Bataineh, M., Marler, T., & Abdel-Malek, K. (2012). *Using artificial neural networks for prediction of dynamic human motion*. Paper presented at International Summit on Human Simulation, Florida, USA.

Bataineh, M., Marler, T., & Abdel-Malek, K. (2013). Artificial neural network-based prediction of human posture. Paper presented at *Digital Human Modeling and Applications in Health, Safety, Ergonomics, and Risk Management. Human Body Modeling and Ergonomics* (pp. 305): Springer.

Beale, M. H., Hagan, M. T., & Demuth, H. B. (2001). Neural network toolbox for use with Matlab user's guide version 4. *The mathworks.*

Bianchini, M., Frasconi, P., & Gori, M. (1995). Learning without local minima in radial basis function networks. *Neural Networks, IEEE Transactions on, 6*(3), 749.

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern Recognition and Machine Learning* (Vol. 1): Springer New York.

Bishop, C. M., & Roach, C. M. (1992). Fast curve fitting using neural networks. *Review of Scientific Instruments, 63*(10), 4450.

Björck, Å. (1967). Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT Numerical Mathematics, 7*(1), 1.

Bouzerdoum, A., & Pattison, T. R. (1993). Neural network for quadratic optimization with bound constraints. *Neural Networks, IEEE Transactions on, 4*(2), 293.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123.

Broomhead, D. S., & Lowe, D. (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks.* (NO. RSRE-MEMO-4148). ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM).

Buhmann, M. D. (2003). *Radial basis functions: theory and implementations* (Vol. 12): Cambridge University Press.

Cao, J., Lin, Z., Huang, G.-B., & Liu, N. (2012). Voting based extreme learning machine. *Information Sciences, 185*(1), 66.

Chakraborty, K., Mehrotra, K., Mohan, C. K., & Ranka, S. (1992). Forecasting the behavior of multivariate time series using neural networks. *Neural Networks, 5*(6), 961-970.

Chen, S., Billings, S. A., & Luo, W. (1989). Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control, 50*(5), 1873.

Chen, S., Cowan, C. F. N., & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *Neural Networks, IEEE Transactions on, 2*(2), 302.

Chen, S., Hong, X., Harris, C. J., & Sharkey, P. M. (2004). Sparse modeling using orthogonal forward regression with PRESS statistic and regularization. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 34*(2), 898-911.

Cherkauer, K. J. (1996). *Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks.* Paper presented at the Working notes of the AAAI workshop on integrating multiple learned models.

Cochocki, A., & Unbehauen, R. (1993). *Neural networks for optimization and signal processing*: John Wiley & Sons, Inc.

Collobert, R., Bengio, S., & Bengio, Y. (2002). A parallel mixture of SVMs for very large scale problems. *Neural Computation, 14*(5), 1105.

Deming, W. E. (1944). *Statistical Adjustment of Data*: New York.

Deng, W., Zheng, Q., & Chen, L. (2009). *Regularized extreme learning machine.* Paper presented at the Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on (pp. 389-395). IEEE.

Drucker, H., Schapire, R., & Simard, P. (1993). Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence, 7*(04), 705.

Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern Classification*: John Wiley & Sons.

Fausett, L. (1994). *Fundamentals of neural networks: architectures, algorithms, and applications*: Prentice-Hall Inc.

Fiacco, A. V., & McCormick, G. P. (1990). *Nonlinear programming: sequential unconstrained minimization techniques* (Vol. 4). Siam.

Fletcher, R. (2013). *Practical Methods of Optimization*: John Wiley & Sons.

Frank, R. J., Davey, N., & Hunt, S. P. (2001). Time series prediction and neural networks. *Journal of Intelligent and Robotic Systems, 31*(1-3), 91.

Freund, Y., & Schapire, R. E. (1995). *A desicion-theoretic generalization of on-line learning and an application to boosting.* Paper presented at the Computational Learning Theory.

Friedman, J. H. (1997). On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery, 1*(1), 55.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*(1), 1-58.

Gholizadeh, S., Salajegheh, E., & Torkzadeh, P. (2008). Structural optimization with frequency constraints by genetic algorithm using wavelet radial basis function neural network. *Journal of Sound and Vibration, 312*(1), 316-331.

Gill, P. E., Murray, W., & Saunders, M. A. (2002). SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization, 12*(4), 979-1006.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural networks architectures. *Neural Computation, 7*(2), 219-269.

Golub, G. H., & Van Loan, C. F. (2012). *Matrix Computations* (Vol. 3): JHU Press.

Grant, M., Boyd, S., & Ye, Y. (2008). CVX: Matlab software for disciplined convex programming: Available from www.stanford.edu/~boyd/cvx/.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural Network Design* (Vol. 1): Pws Boston.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural Network Design*: Pws Pub. Boston.

Han, J., Kamber, M., & Pei, J. (2006). *Data Mining: Concepts and Techniques*: Morgan kaufmann.

Hansen, L. K., Liisberg, C., & Salamon, P. (1992). *Ensemble methods for handwritten digit recognition.* Paper presented at the Neural Networks for Signal Processing [1992] II., Proceedings of the 1992 IEEE-SP Workshop.

Hansen, L. K., & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*, 993.

Hartman, E. J., Keeler, J. D., & Kowalski, J. M. (1990). Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation, 2*(2), 210.

Haykin, S., & Network, N. (2004). A comprehensive foundation. *Neural Networks, 2*(2004).

Haykin, S. S., Haykin, S. S., Haykin, S. S., & Haykin, S. S. (2009). *Neural Networks and Learning Machines* (Vol. 3): Prentice Hall New York.

He, P., & Jagannathan, S. (2007). Reinforcement learning neural-network-based controller for nonlinear discrete-time systems with input constraints. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 37*(2), 425.

Hecht-Nielsen, R. (1989). *Theory of the backpropagation neural network.* Paper presented at the Neural Networks, 1989. IJCNN., International Joint Conference on.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence, 40*(1), 185-234.

Hock, W., & Schittkowski, K. (1983). A comparative performance evaluation of 27 nonlinear programming codes. *Computing, 30*(4), 335.

Hong, P., Wen, Z., & Huang, T. S. (2002). Real-time speech-driven face animation with expressions using neural networks. *Neural Networks, IEEE Transactions on, 13*(4), 916.

Hornik, K. (1993). Some new results on neural network approximation. *Neural Networks, 6*(8), 1069.

Huang, G.-B. (2003). Learning capability and storage capacity of two-hidden-layer feedforward networks. *Neural Networks, IEEE Transactions on, 14*(2), 274-281.

Huang, G.-B., & Chen, L. (2008). Enhanced random search based incremental extreme learning machine. *Neurocomputing, 71*(16), 3460.

Huang, G.-B., & Siew, C.-K. (2004). *Extreme learning machine: RBF network case.* Paper presented at the Control, Automation, Robotics and Vision Conference, 2004. ICARCV 2004 8th.

Huang, G.-B., Wang, D. H., & Lan, Y. (2011). Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics, 2*(2), 107.

Huang, G.-B., Zhou, H., Ding, X., & Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 42*(2), 513.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2004). Paper presented at the Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing, 70*(1), 489.

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting, 22*(4), 679-688.

Inman, V. T., & Eberhart, H. D. (1953). The major determinants in normal and pathological gait. *The Journal of Bone & Joint Surgery, 35*(3), 543-558.

Isaksson, M., Jalden, J., & Murphy, M. J. (2005). On using an adaptive neural network to predict lung tumor motion during respiration for radiotherapy applications. *Medical Physics, 32*(12), 3801.

Ishu, K., van Der Zant, T., Becanovic, V., & Ploger, P. (2004). *Identification of motion with echo state network.* Paper presented at the Oceans'04. Mtts/Ieee Techno-Ocean'04.

Jung, E. S., & Park, S. (1994). Prediction of human reach posture using a neural network for ergonomic man models. *Computers & Industrial Engineering, 27*(1), 369-372.

Kennedy, M. P., & Chua, L. O. (1988). Neural networks for nonlinear programming. *Circuits and Systems, IEEE Transactions on, 35*(5), 554-562.

Kerschen, G., Worden, K., Vakakis, A. F., & Golinval, J.-C. (2006). Past, present and future of nonlinear system identification in structural dynamics. *Mechanical Systems and Signal Processing, 20*(3), 505-592.

Kim, J. H., Xiang, Y., Bhatt, R., Yang, J., Chung, H.-J., Patrick, A., . . . Abdel-Malek, K. (2008). *Efficient ZMP formulation and effective whole-body motion generation for a human-like mechanism.* Paper presented at the ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

Kim, J. H., Xiang, Y., Yang, J., Arora, J. S., & Abdel-Malek, K. (2010). Dynamic motion planning of overarm throw for a biped human multibody system. *Multibody System Dynamics, 24*(1), 1-24.

Kim, Y. H., & Lewis, F. L. (1999). Neural network output feedback control of robot manipulators. *Robotics and Automation, IEEE Transactions on, 15*(2), 301.

Kleinbaum, D., Kupper, L., Nizam, A., & Rosenberg, E. (2013). *Applied regression analysis and other multivariable methods*: Cengage Learning.

Kohavi, R., & Wolpert, D. H. (1996). *Bias plus variance decomposition for zero-one loss functions.* Paper presented at the Icml (pp. 275-283).

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE, 78*(9), 1464-1480.

Kohonen, T. (2001). *Self-Organizing Maps* (Vol. 30): Springer.

Koike, Y., & Kawato, M. (1995). Estimation of dynamic joint torques and trajectory formation from surface electromyography signals using a neural network model. *Biological Cybernetics, 73*(4), 291.

Köthe, G., & Garling, D. J. H. (1969). *Topological Vector Spaces* (Vol. 462): Springer.

Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. *Advances in Neural Information Processing Systems*, 231.

Kun, A., & Miller Iii, W. T. (1996). *Adaptive dynamic balance of a biped robot using neural networks.* Paper presented at the Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on (Vol. 1, pp. 240-245). IEEE.

Kwon, H.-J., Xiang, Y., Bhatt, R., Rahmatalla, S., Arora, J. S., & Abdel-Malek, K. (2014). Backward walking simulation of humans using optimization. *Structural and Multidisciplinary Optimization*, 1-11.

Lan, Y., Soh, Y. C., & Huang, G.-B. (2009). Ensemble of online sequential extreme learning machine. *Neurocomputing, 72*(13), 3391.

Lapedes, A., & Farber, R. (1987). Nonlinear signal processing using neural networks. *Prediction and System Modelling* (No. LA-UR-87-2662; CONF-8706130-4)..

Lázaro-Gredilla, M., & Figueiras-Vidal, A. R. (2010). Marginalized neural network mixtures for large-scale regression. *Neural Networks, IEEE Transactions on, 21*(8), 1345-1351.

Lewis, F. L., Yesildirek, A., & Liu, K. (1996). Multilayer neural-net robot controller with guaranteed tracking performance. *Neural Networks, IEEE Transactions on, 7*(2), 388.

Liang, N.-Y., Saratchandran, P., Huang, G.-B., & Sundararajan, N. (2006). Classification of mental tasks from EEG signals using extreme learning machine. *International journal of neural systems, 16*(01), 29.

Lloyd, S. (1982). Least squares quantization in PCM. *Information Theory, IEEE Transactions on, 28*(2), 129.

Looney, C. G. (1997). *Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists*: Oxford University Press Inc.

Maa, C. Y., & Shanblatt, M. A. (1992). Linear and quadratic programming neural network analysis. *Neural Networks, IEEE Transactions on, 3*(4), 580.

MacQueen, J. (1967). *Some methods for classification and analysis of multivariate observations.* Paper presented at the Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability.

Mathia, K., & Clark, J. (2002). *On neural network hardware and programming paradigms.* Paper presented at the Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on (Vol. 3, pp. 2692-2697). IEEE.

Miller III, W. T., Glanz, F. H., & Kraft III, L. G. (1990). Cmas: An associative neural network alternative to backpropagation. *Proceedings of the IEEE, 78*(10), 1561-1567.

Miller, W. T., Werbos, P. J., & Sutton, R. S. (1995). *Neural Networks for Control*: MIT press.

Moerland, P., & Fiesler, E. (1997). Neural network adaptations to hardware implementations. *Handbook of Neural Computation, 1*, 2.

Moody, J., & Darken, C. (1988). *Learning with localized receptive fields*: Yale Univ. Department of Computer Science.

Moody, J., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation, 1*(2), 281.

Park, J., & Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks. *Neural Computation, 3*(2), 246.

Patterson, D. W. (1998). *Artificial Neural Networks: Theory and Applications*: Prentice Hall PTR.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation, 1*(2), 263-269.

Perrone, M. P., & Cooper, L. N. (1992). When networks disagree: Ensemble methods for hybrid neural networks: DTIC Document.

Platt, J. (1991). A resource-allocating network for function interpolation. *Neural Computation, 3*(2), 213.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE, 78*(9), 1481.

Powell, M. J. D. (1983). Variable metric methods for constrained optimization *Mathematical Programming The State of the Art* (pp. 288): Springer.

Powell, M. J. D. (1987). *Radial basis functions for multivariable interpolation: a review.* Paper presented at the Algorithms for Approximation.

Press, W. H. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*: Cambridge University Press.

Priddy, K. L., & Keller, P. E. (2005). *Artificial Neural Networks: An Introduction* (Vol. 68): SPIE Press.

Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research, 6*, 1939.

Rahmatalla, S., Xiang, Y., Smith, R., Meusch, J., & Bhatt, R. (2011). A validation framework for predictive human models. *International journal of human factors modelling and simulation, 2*(1), 67-84.

Reed, R. D., & Marks, R. J. (1998). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*: Mit Press.

Ripley, B. D. (2007). *Pattern Recognition and Neural Networks*: Cambridge University Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*. (No. ICS-8506). CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE.

Saha, A., & Keeler, J. D. (1990). *Algorithms for better representation and faster learning in radial basis function networks.* Paper presented at the Advances in Neural Information Processing Systems 2.

Sanner, R. M., & Slotine, J. J. (1992). Gaussian networks for direct adaptive control. *Neural Networks, IEEE Transactions on, 3*(6), 837.

Schaefer, H. H., & Wolff, M. P. (1999). *Locally Convex Topological Vector Spaces*. (pp. 36-72). Springer New York.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197.

Sharkey, A. J. C. (1999). Multi-Net Systems *Combining Artificial Neural Nets* (pp. 1): Springer.

Snelson, E., & Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems, 18*, 1257.

Specht, D. F. (1991). A general regression neural network. *Neural Networks, IEEE Transactions on, 2*(6), 568.

Stakem, F., & AlRegib, G. (2008). *Neural Networks for Human Arm Movement Prediction in CVEs.* Paper presented at the Proceedings of 3DPVT.

Statnikov, A., Aliferis, C. F., Tsamardinos, I., Hardin, D., & Levy, S. (2005). A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. *Bioinformatics (Oxford, England), 21*(5), 631. doi: 10.1093/bioinformatics/bti033

Stinchcombe, M., & White, H. (1989). *Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions.* Paper presented at the Neural Networks, 1989. IJCNN., International Joint Conference on (pp. 613-617). IEEE.

Sultan, S., & Marler, T. (2012). Multi-scale Human Modeling for Injury Prevention. *2nd International Conference on Applied Digital Human Modeling*.

Sun, Z.-L., Choi, T.-M., Au, K.-F., & Yu, Y. (2008). Sales forecasting using extreme learning machine with applications in fashion retailing. *Decision Support Systems, 46*(1), 411.

Suresh, S., Saraswathi, S., & Sundararajan, N. (2010). Performance enhancement of extreme learning machine for multi-category sparse data classification problems. *Engineering Applications of Artificial Intelligence, 23*(7), 1149.

Suresh, S., Venkatesh Babu, R., & Kim, H. J. (2009). No-reference image quality assessment using modified extreme learning machine classifier. *Applied Soft Computing, 9*(2), 541.

Ting, K. M., & Witten, I. H. (2011). Issues in stacked generalization. *J. Artif. Intell. Res.(JAIR), 10*, 271-289.

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence, 70*(1), 119-165.

Trippi, R. R., & Turban, E. (1992). *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance*: McGraw-Hill, Inc.

Twomey, J. M., & Smith, A. E. (1998). Bias and variance of validation methods for function approximation neural networks under conditions of sparse data. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 28*(3), 417-430.

Van Heeswijk, M., Miche, Y., Lindh-Knuutila, T., Hilbers, P. A. J., Honkela, T., Oja, E., & Lendasse, A. (2009). Adaptive ensemble models of extreme learning machines for time series prediction. *Artificial Neural Networks–ICANN 2009* (pp. 305): Springer.

van Heeswijk, M., Miche, Y., Oja, E., & Lendasse, A. (2011). GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing, 74*(16), 2430.

Walter, E., & Pronzato, L. (1997). Identification of parametric models. *Communications and Control Engineering*.

Wasserman, P. D. (1993). *Advanced Methods in Neural Computing*: John Wiley & Sons Inc.

Watrous, R. L. (1988). Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. *Technical Reports (CIS)*, 597.

White, H. (1989). Learning in artificial neural networks: A statistical perspective. *Neural Computation, 1*(4), 425-464.

Widrow, B., Rumelhart, D. E., & Lehr, M. A. (1994). Neural networks: Applications in industry, business and science. *Communications of the ACM, 37*(3), 93-105.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation, 1*(2), 270-280.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks, 5*(2), 241.

Xia, Y. (1996). A new neural network for solving linear and quadratic programming problems. *Neural Networks, IEEE Transactions on, 7*(6), 1544.

Xia, Y., Leung, H., & Wang, J. (2002). A projection neural network and its application to constrained optimization problems. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 49*(4), 447.

Xia, Y., & Wang, J. (1998). A general methodology for designing globally convergent optimization neural networks. *Neural Networks, IEEE Transactions on, 9*(6), 1331.

Xia, Y., & Wang, J. (2005). A recurrent neural network for solving nonlinear convex programs subject to linear constraints. *Neural Networks, IEEE Transactions on, 16*(2), 379.

Xiang, Y., Arora, J. S., & Abdel-Malek, K. (2010). Physics-based modeling and simulation of human walking: a review of optimization-based and other approaches. *Structural and Multidisciplinary Optimization, 42*(1), 1-23.

Xiang, Y., Arora, J. S., & Abdel-Malek, K. (2011). Optimization-based prediction of asymmetric human gait. *Journal of Biomechanics, 44*(4), 683-693.

Xiang, Y., Arora, J. S., & Abdel-Malek, K. (2012). Hybrid predictive dynamics: a new approach to simulate human motion. *Multibody System Dynamics, 28*(3), 199-224.

Xiang, Y., Arora, J. S., Rahmatalla, S., & Abdel-Malek, K. (2009). Optimization-based dynamic human walking prediction: One step formulation. *International Journal for Numerical Methods in Engineering, 79*(6), 667-695.

Xiang, Y., Arora, J. S., Rahmatalla, S., Marler, T., Bhatt, R., & Abdel-Malek, K. (2010). Human lifting simulation using a multi-objective optimization approach. *Multibody System Dynamics, 23*(4), 431-451.

Xiang, Y., Chung, H.-J., Kim, J. H., Bhatt, R., Rahmatalla, S., Yang, J., . . . Abdel-Malek, K. (2010). Predictive dynamics: an optimization-based novel approach for human motion simulation. *Structural and Multidisciplinary Optimization, 41*(3), 465.

Yang, S., & Wang, Q. (2000). Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *Neural Networks, IEEE Transactions on, 11*(2), 474.

Yoo, J.-H., Hwang, D., Moon, K.-Y., & Nixon, M. S. (2008). *Automated human recognition by gait using neural network.* Paper presented at the Image Processing Theory, Tools and Applications, 2008. IPTA 2008. First Workshops on.

Zhang, B., Horváth, I., Molenbroek, J. F., & Snijders, C. (2010). Using artificial neural networks for human body posture prediction. *International Journal of Industrial Ergonomics, 40*(4), 414-424.

Zhang, G., Eddy Patuwo, B., & Y Hu, M. (1998). Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting, 14*(1), 35.

Zhang, R., Huang, G.-B., Sundararajan, N., & Saratchandran, P. (2007). Multicategory classification using an extreme learning machine for microarray gene expression cancer diagnosis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 4*(3), 485.

Zhang, S., & Constantinides, A. G. (1992). Lagrange programming neural networks. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, 39*(7), 441.

Zhang, Y., & Li, Z. (2009). Zhang neural network for online solution of time-varying convex quadratic program subject to time-varying linear-equality constraints. *Physics Letters A, 373*(18), 1639.

Zhang, Y., & Wang, J. (2002). A dual neural network for convex quadratic programming subject to linear equality and inequality constraints. *Physics Letters A, 298*(4), 271.

Zhang, Y., Wang, J., & Xia, Y. (2003). A dual neural network for redundancy resolution of kinematically redundant manipulators subject to joint limits and joint velocity limits. *Neural Networks, IEEE Transactions on, 14*(3), 658.

Zhang, Y., & Wu, L. (2008). Weights optimization of neural network via improved BCO approach. *Progress In Electromagnetics Research, 83*, 185.

Zhou, Z.-H., Jiang, Y., Yang, Y.-B., & Chen, S.-F. (2002). Lung cancer cell identification based on artificial neural network ensembles. *Artificial Intelligence in Medicine, 24*(1), 25.

Zhou, Z.-H., Wu, J., & Tang, W. (2002). Ensembling neural networks: many could be better than all. *Artificial Intelligence, 137*(1), 239.

Zhu, Q.-Y., Qin, A. K., Suganthan, P. N., & Huang, G.-B. (2005). Evolutionary extreme learning machine. *Pattern Recognition, 38*(10), 1759.

APPENDIX A

TABLES OF TRAINING CASES FOR THE PROBLEM OF MULTI-SCALE HUMAN MODELING FOR INJURY PREVENTION

Table A.1: All 25 training cases and 3 test cases for the problem of predicting the knee forces in the multi-scale human modeling system (walking task).

	Case No.	Inputs				Outputs		
		Angle (deg)	GRF (N)	Comp Force (N)	Shear Force (N)	Stress (MPa)	Strain (%)	Contact Pre. (MPa)
Training Cases	1	-9.249	929.371	1403.287	-91.912	36.57	3.001	13.62
	2	-9.889	853.45	1330.358	-92.136	34.39	2.953	13.42
	3	-8.593	779.062	1255.265	-93.604	33.37	2.912	13.19
	4	-6.345	719.322	1204.403	-29.731	17.75	2.099	6.97
	5	-3.089	737.589	1069.118	-46.42	16.11	2.172	7.35
	6	-4.327	740.215	1007.859	-39.526	15.14	2.001	6.68
	7	-8.627	745.904	982.167	-105.696	28.19	2.811	13.72
	8	-17.978	615.467	1155.365	-200.543	25.57	5.015	11.92
	9	-47.072	0	546.691	74.991	24.87	0.672	4.22
	10	-54.264	0	625.593	63.761	70.35	0.543	6.52
	11	-55.277	0	737.708	48.094	56.12	3.892	5.22
	12	-49.331	0	816.652	51.361	18.43	1.276	5.49
	13	-36.9984	0	550.895	-8.399	17.89	0.799	5.69
	14	-8.809	0	117.679	-46.179	10.47	1.215	6.14
	15	-1.511	0	119.392	-31.115	4.25	1.323	3.25
	16	-0.847	0	236.328	12.998	5.83	0.311	1.11
	17	-1.665	171.73	305.235	25.476	7.35	0.857	1.23
	18	-3.991	740.47	1020.99	-40.089	15.098	2.212	7.51
	19	-5.981	740.33	995.28	-61.4	21.05	2.48	8.48
	20	-7.118	741.38	1284.14	-92.8	35.6	2.904	13.205
	21	-12.249	756.17	1010.22	-142.6	27.9	3.49	13.45
	22	-5.736	0	117.94	-41.65	6.98	1.287	4.54
	23	-10.986	0	142.098	-42.051	10.001	1.025	7.51
	24	-21.363	0	234.99	-41.589	10.72	0.667	3.41
	25	-34.004	84.423	338.987	38.889	12.57	1.843	4.07
Test Cases	1	-16.05	774.311	1038.043	-180.978	27.28	4.955	13.12
	2	-21.363	0	234.993	-41.589	10.72	0.667	3.41
	3	-4.158	709.491	1146.389	-36.801	16.32	2.129	7.13

Table A.2: All 25 training cases and 3 test cases for the problem of predicting the knee forces in the multi-scale human modeling system (stairs ascent task).

	Case No.	Inputs				Outputs		
		Angle (deg)	GRF (N)	Comp Force (N)	Shear Force (N)	Stress (MPa)	Strain (%)	Contact Pre. (MPa)
Training Cases	1	-74.371	261.514	633.264	214.648	40.31	2.618	18.38
	2	-61.988	218.771	315.209	-224.29	49.11	6.156	33.13
	3	-10.019	615.912	913.478	111.225	27.29	2.791	12.63
	4	-29.836	707.33	1033.593	-119.684	29.48	2.888	13.37
	5	-0.387	826.192	1301.548	30.841	26.34	1.757	7.17
	6	-2.301	772.238	1275.569	47.706	28.99	1.151	7.23
	7	-1.315	772.238	1244.253	69.076	32.15	1.164	6.39
	8	-3.258	772.238	1252.824	-2.441	21.54	1.566	5.27
	9	-6.228	772.238	1302.082	-27.308	19.49	2.182	7.24
	10	-4.882	572.541	757.559	-78.532	14.37	2.231	7.78
	11	-0.022	591.738	717.611	-35.225	8.88	2.662	6.91
	12	-23.326	0	281.865	29.953	20.18	1.248	8.84
	13	-58.031	0	546.781	63.601	30.48	0.805	4.63
	14	-75.681	0	742.376	46.952	19.93	0.923	6.47
	15	-81.719	0	262.291	-34.674	9.53	1.632	4.03
	16	-79.802	0	191.915	-30.225	7.67	1.439	3.55
	17	-77.593	0	154.615	-25.048	6.28	1.041	3.03
	18	-68.224	236.704	529.82	5.981	42.97	3.961	22.85
	19	-57.366	593.097	345.091	-222.123	49.34	4.821	26.32
	20	-45.934	725.153	450.65	-126.63	48.82	4.731	21.95
	21	-22.118	553.958	730.82	14.96	31.08	3.017	14.46
	22	-3.196	0	598.37	5.091	14.92	1.892	7.34
	23	-2.198	586.091	739.81	-49.827	11.98	2.451	7.18
	24	-64.272	0	640.743	58.812	28.005	0.981	5.02
	25	-46.566	0	429.35	55.921	28.93	0.712	3.01
Test Cases	1	-5.017	772.238	1264.706	-9.317	20.94	1.731	5.76
	2	-29.836	0	311.571	39.168	23.91	0.238	1.25
	3	-75.272	0	476.067	46.409	18.97	2.489	6.14

APPENDIX B

TABLES OF NETWORK PARAMETERS VALUES FOR SIMULATED PREDICTIVE DYNAMIC TASKS

Table B.1: The network basis functions, basis function spread ($\boldsymbol{\sigma}$) values, and their corresponding original training cases for the walking task.

Basis function ranking	Basis function spread (σ)	Original training case number
1	1.71	72
2	1.71	93
3	1.71	99
4	2.1	115
5	1.71	78
6	1.71	95
7	2.98	42
8	2.98	41
9	2.6	138
10	2.98	20
11	2.98	21
12	3.05	36
13	3.81	15
14	2.98	58
15	2.1	113
16	1.71	74
17	2.1	116
18	3.05	137
19	2.98	16
20	3.05	57
21	2.1	109
22	2.1	117
23	2.1	114
24	1.71	75
25	3.05	25
26	2.6	135
27	1.71	96
28	1.71	71
29	1.71	73
30	1.71	67
31	2.6	12
32	2.6	33
33	3.81	4
34	2.6	54
35	2.1	120
36	3.05	141
37	2.6	30

Table B.1: Continued.

38	1.71	94
39	1.71	88
40	2.6	51
41	3.05	136
42	3.05	46
43	3.05	130
44	2.98	142
45	2.6	9
46	2.98	59
47	1.71	92
48	2.98	38
49	3.05	50
50	3.05	134
51	2.98	17
52	3.81	211
53	3.25	8
54	2.98	143
55	3.05	232
56	3.05	253
57	3.81	216
58	3.05	29
59	3.05	31
60	3.05	131
61	3.05	237
62	3.05	258
63	3.25	10
64	3.05	49
65	3.05	133
66	1.71	80
67	1.71	101
68	3.81	7
69	3.5	32
70	2.1	121
71	3.81	1
72	3.05	43
73	3.05	22
74	3.05	127

Table B.2: The network basis functions, basis function spread ($\boldsymbol{\sigma}$) values, and their corresponding original training cases for the go-prone task.

Basis function ranking	Basis function spread (σ)	Original training case number
1	1.77	11
2	3.55	16
3	1.8	30
4	0.65	49
5	1.77	266
6	5.57	263
7	1.77	164
8	0.65	207
9	1.77	113
10	1.77	62
11	7.2	28
12	1.8	234
13	5.57	162
14	2.23	41
15	2.23	36
16	7.66	29
17	2.23	50
18	0.65	304
19	0.65	202
20	1.17	27
21	0.65	151
22	0.65	253
23	0.65	100
24	2.23	51
25	4.83	15
26	1.17	129
27	3.1	7
28	2.23	35
29	4.61	13
30	0.64	43
31	2.95	1
32	2.23	305
33	1.8	81
34	1.7	26
35	0.59	24
36	2.28	32
37	2.28	47
38	2.23	291

Table B.3: The full-body 55-DOFs and their change in value (ΔDOF_d) in the walking task

DOF number	DOF change in value (ΔDOF_d)
1	0.011
2	0.011
3	0.003
4	0.025
5	0.036
6	0.022
7	0.032
8	0.06
9	0.027
10	0.023
11	0.043
12	0.021
13	0.023
14	0.038
15	0.014
16	0.028
17	0.042
18	0.019
19	0.014
20	0.009
21	0.012
22	0.018
23	0
24	0.08
25	0.002
26	0.008
27	0.009
28	0.016
29	0.009
30	0.013
31	0.017
32	0.005
33	0.014
34	0.002
35	0.005
36	0.007
37	0.023
38	0.021
39	0.002
40	0.015
41	0.014
42	0.025
43	0.035
44	0.016
45	0.054
46	0.028
47	0.018
48	0.023

Table B.3: Continued.

49	0.022
50	0.037
51	0.018
52	0.023
53	0.016
54	0.031
55	0.012

Table B.4: The full-body 55-DOFs and their change in value (ΔDOF_d) in the go-prone task.

DOF number	**DOF change in value (ΔDOF_d)**
1	0.014
2	0.026
3	0.014
4	0.027
5	0.021
6	0.037
7	0
8	0.036
9	0
10	0
11	0.033
12	0
13	0
14	0.027
15	0
16	0
17	0.032
18	0
19	0.019
20	0.017
21	0.027
22	0.023
23	0.013
24	0.019
25	0.007
26	0.013
27	0.009
28	0.044
29	0.028
30	0.031
31	0.034
32	0.027
33	0.032
34	0.001
35	0.006
36	0.008
37	0.003

Table B.4: Continued.

38	0.006
39	0
40	0
41	0
42	0
43	0.027
44	0
45	0.027
46	0.029
47	0
48	0.014
49	0
50	0.029
51	0
52	0.024
53	0.038
54	0
55	0.019