Use Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission to download and/or print my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

_____

Signature

_____

Date

EMG Feature-Based Approach toward a Robust Artificial Neural Networks Analysis of Human Finger Motion

By

Abduljaleel (AJ) Alriyadh

A thesis
submitted in partial fulfillment
of the requirements for the degree of

Master of Science in the Department of

Measurement and Control Engineering

IDAHO STATE UNIVERSITY

August 2016

Committee Approval

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of **Abduljaleel (AJ) Alriyadh** find it satisfactory and recommended that it be accepted:

X _____

Dr. Schoen, Marco: major advisor

X _____

Dr. Devine, Nancy: co-advisor

X _____

Dr. Stuffle, Gene: graduate faculty representative

Acknowledgements

Success is no accident; it is rather the product of hard work, perseverance, determination and, most of all, the support of the loved ones. I consider this work as the recompense of all the nights I spent without sleep, the recompense of standing solid against all the obstacles that showed up to hinder my progress. Behind the scenes of this achievement and deep between the lines of this thesis, stand a number of people who made my success possible—the people I can't thank enough for their support, love, and motivation.

The first person on my list is my wife, Dr. Hadeel F. Alkhatabi. She bestowed on me one of my best gifts in life when she recently agreed to marry me. Although she knew she had to deal with a long distance marriage, she didn't hesitate to offer her love and support. Despite all her responsibilities as a doctor and the amount of time and effort she put everyday into saving lives, she carried, throughout all the trouble, our five weeks old son, Elyas. I can only imagine how hard this was for her. Nevertheless, all what an ambitious and strong women provided to me is endearment, and comprehension. Moreover, as this research is related to human biology, M.D Alkhatabi has always been a source of knowledge.

I wouldn't be writing this document if it wasn't for my parents. No matter how hard I try, it remains rather impossible to see a way to return the favor to them. My mother is going through very rough health issues. My head hits the pillow every night uncertain whether I am going to hear from her the day after or not. Being unemployed and single, she never needed support more than she does today. Nevertheless, she never stopped encouraging me to chase my goals overseas with her beautiful smile and warm

wishes for a long life full of success and dreams to come true. My father has never stopped providing the guidance and support I needed to get through this stage of my life. As an international student, I faced many financial issues, which, without his help, could've been, very likely, be the reason for me to never pursue my degree. In addition to that, he never scrimped his smile and love to make sure I am following, in his shoe, the path to my ambition. My parents never failed to demonstrate an example of dedication, strength, and fondness—values I would do my best to pass to my newborn. I believe, as they have always told me, that my success in life is their ultimate wish and here I lay this work inside their widely open arms of warmth.

I also want to take this opportunity to express my gratitude to my best friends, Waleed Yosef and Mohammed Marzoq. Being bounded by the love of friends is just as important as being surrounded by family. Mohammed and Waleed did more than being supportive; without their phone calls, the cold long nights of frustration and disappointment would not serve as motivation to keep me chasing my goals and try different approaches to achieve the results of this research. Waleed never held back reminding me how smart and determined he thinks I am, while Mohammed kept on fixing my nodded vision to interchange the desperation of today with the hope of tomorrow.

My deepest gratitude to my advisor, Professor Schoen, Marco. Despite his busy schedule, he never hesitated to provide guidance to me and this research. Professor Schoen, hasn’t stopped encouraging, leading, and motivating me throughout my degree. I remember the many times I entered his office all disappointed and ready to give up. He always would look me in the eye and remind me that there will be a moment when “I will

be walking on the ceiling" out of happiness. Here I am writing my thesis with a big smile. Professor Schoen isn't only my advisor; he is rather a mentor, role model, and a friend. I am honored to work under his supervision, without which I wouldn't have learned the knowledge I am proud of today.

Finally, I can't wrap up my acknowledgements, without thanking Prof. Ken Bosworth, Prof. Alba Perez, and Prof Herbert Jaeger. They provided another source of many of what I have learned throughout my experience.

The road to success is hard to navigate, but with drive and passion anything become possible. Without the help of my friends, family, and professors, none of this would have become a reality. I am both happy and honored with this opportunity to express my sincere gratitude to these people and to them, I dedicate this achievement.

Table of Contents

List of Figures

List of tables

Abstract

Surface electromyography signals (sEMG) are known to be nonstationary and highly noisy [1]. Hence, classification of the signal in relation to human motion can be difficult. Based on other studies (reviewed later), it is believed that there exists a correspondence between sEMG signals and human limb motion. This research explores this relationship.

The study goals of this work contribute to the funded research in which Idaho State University (ISU) is participating, ARWED. The project targets stroke victims who suffer partial loss in their motor abilities. The goal is to develop an augmented reality device (ARWED) that helps to train and retrain patients.

A robust approach to identify the intended motion using only sEMG signals is achieved. Previous research in the field succeeded in classifying sEMG in relation to kinematical variables. Using kinematical variables provide more information, which, as a result, makes classification and analysis an easier task. Instead, this study targeted the mapping of sEMG signals without provided information about the kinematical variables.

Artificial Neural Networks (ANN), along with seventeen features, are able to map out the intended motion. The data of twenty participants are collected for the purpose of validating the achieved results. For each participant, a separate neural network is trained to predict the intended motion based on information extracted from different sEMG signals.

The proposed approach is tested on twenty individuals and five different finger motions. It is valid in deducing the five motions. The performance (mean square error) of

the different training algorithms is compared. Levenberg-Marquardt (LM) Algorithm with an average performance of 0.0461, Bayesian Regularization (BR) Algorithm with an average performance of 0.03105, and Scaled Conjugate Gradient (SCG) with an average performance of 0.1398 have all shown reliability in identifying the different types of motion. The linear correlation between the predicted and the target outputs is also computed. The scores are 0.9898 for LM, 0.99305 for BR, and 0.96753 for SCG.

Chapter 1: Introduction

1.1 What is an EMG signal?

Electromyography (EMG) is the study of muscle activity through analysis of myoelectric signals [2]. Electrodiagnostic studies, under which EMG analysis is categorized, started in the nineteenth century. It wasn't, however, until 30-40 years ago, when these studies had a significance impact [3]. The reason perhaps is the later development of computing power and machine analysis.

EMG studies are used in a variety of areas such as medical research, rehabilitation, ergonomics, and sport science. The use of EMG in medical research involves orthopedics, surgery, functional neurology, and posture analysis. In the field of rehabilitation, EMG helps in post-surgery, neurological rehabilitation, physical therapy, and active training therapy. For ergonomics, EMG is used to study the analysis of demand, risk prevention, ergonomics design, and product certification. As for sport science, EMG research contributes to areas such as biomechanics, movement analysis, athletes' strength training, and sport rehabilitation [2].

1.2 The origin of EMG

The smallest element by which the muscle activities are controlled is called a motor unit. These motor points are made of a "single horn cell, its axon, and all the muscle fibers it innervates" [2]. All fibers within a unit act as one. Figure 1.2.1 shows the internal structure of a motor unit. Under neurological control, an ionic difference, between the inner and outer spaces of the horn cell, creates an electrical potential. This potential (ranges between -80 to -90 mille volts when muscle is not contracted) is

maintained through physiological processes. This phenomenon is called the resting potential [2].

When a motion is intended, chemical processes take place causing the potential to rapidly change from -80 up to + 30 mille volts. This process is called the action potential, which, afterwards, spreads through the muscle fibers into a tubular system resulting in motion. The full action potential process goes through a number of stages: depolarization, repolarization, and after-hyperpolarization. Each stage is caused by the occurrence of chemical processes. Figure *1.2.2* shows the signal as it passes through the three stages [2]. The EMG signal is the result of the action potential and, mainly, the depolarization and repolarization processes.

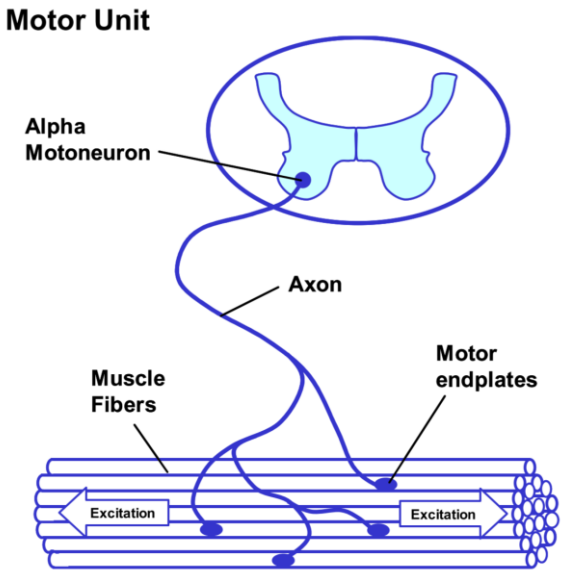


Figure 1.2.1: internal structure of motor units [2]

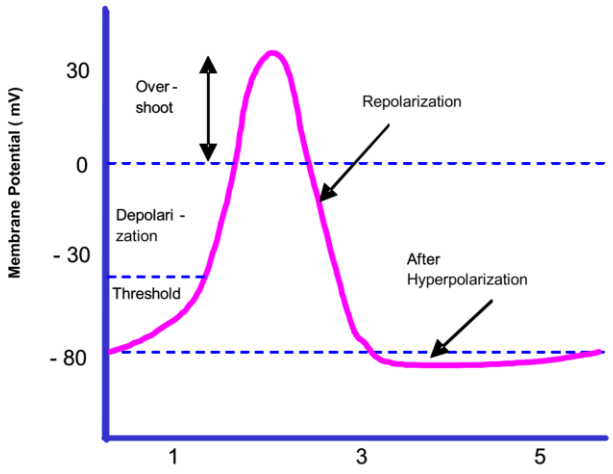


Figure 1.2.2: activation potential *[2]*.

1.3 Artificial Neural Networks

Humans possess one of the most complicated, yet capable, exiting control system. We perceive the world thorough our sensory stations, such as sight, hearing, smell, taste, and touch. Based on the physical variable(s) perceived, one signal or more travels through the nervous system to the brain, which, in turn, based on complex analysis, make decisions, which translate to different outputs such as motion. The capabilities of this control network have, throughout the years, inspired the founders of control theory.

As you are reading this report, some 10^{11} neurons are complexly communicating with each other. They are working together to facilitate your listening, breathing, motion and thinking. Some of our neural structures were born with us and others were established by life experience [4]. The connection between these neurons is highly complicated (approximately 10^4 connections per neuron) [4]. The question that Warren McCulloch and Walter Pitts asked in the 1940s: is it possible to construct a smaller set of artificial neurons? [4]. The answer is yes.

Artificial neural networks are inspired by their biological counterparts. A neuron has three main components (see Figure *1.3.1*): the dendrites, the cell body, and the axon. The dendrites, a tree-like connections, carries the electrical signals into the cell body, where the signals get summed and processed to be, then, sent out through the axon to other neurons. The point of contact between the axon of a cell and dendrite of another cell is called the synapse. It is worth mentioning that even though the biological neural networks are much slower than electrical circuits, they are able to perform, in many tasks, much faster. The reason is due to the massive parallel structure of the biological neural networks, at which all the neurons fire at the same time [4].

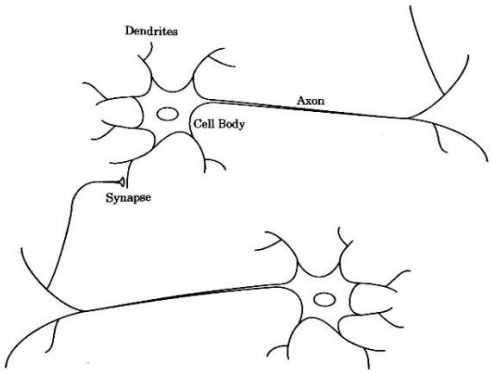


Figure 1.3.1: The biological Neural cell from which the ANN were inspired *[4]*.

Figure *1.3.2* shows the simplest structure of an artificial neuron, the single-input. A scaler input is multiplied by its weight, w, and summed with a bias, b. The resultant is then fed to a transfer function, f, which can be picked to be anything, to produce the output. Examples of transfer functions are sigmoid, hard limit, and linear. Relating this structure to the biological neuron discussed earlier, the input corresponds to the dendrites, the weight to the strength of the synapse, the summation and the transfer function to the cell body, and the output, a, corresponds to the axon of the cell.

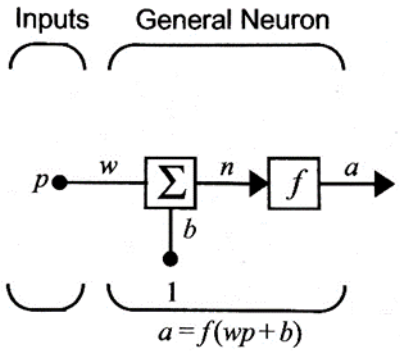


Figure 1.3.2: Single-Input neuron model *[4]*.The output of this model is $a = f(wp + b)$. Where p is the input, w is the output, b corresponds to the bias, and, f, is the transfer function by which output of the neuron is derived. All components of the output are scalers in this case

Generally speaking, more complex neural structures are more capable. A single-input neuron can be extended to a multiple-input neuron (see Figure *1.3.3*), where the input and the weights become vectors instead of scalers. As shown in Figure *1.3.4*, a multiple-inputs neuron also can be extended to a single layer of neurons—with multiple inputs. Moreover, a single layer can be extended to multiple layers (see Figure *1.3.5*). The complexity of the artificial neural network can be built up as desired to achieve more complicated tasks. It is, however, worth mentioning that too complex structures can be troublesome as, with larger ANNs, "irrelevant statistical fluctuations in the training data will be learnt by the model. That leads to poor generalization on test data" [5].

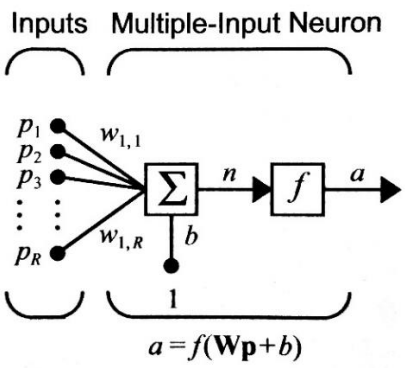


Figure 1.3.3: : Multiple-input neuron *[4]*. the input, p, and the weights, w, are now vectors instead of scalers. The output of this model is $a = f(\boldsymbol{Wp} + b)$. the vectors, W and p, are bolded to represent that they are not scalers anymore.

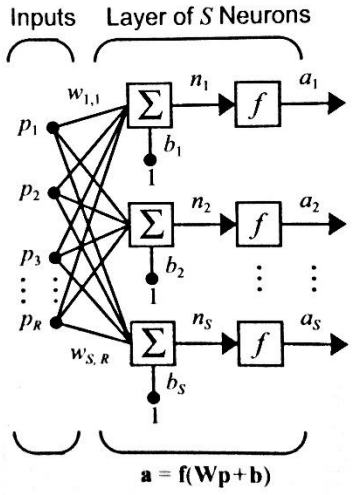


Figure 1.3.4: : A layer of neurons each with multiple inputs and one output *[4]*. The output of this model, $\boldsymbol{a} = \boldsymbol{f}(\boldsymbol{Wp} + \boldsymbol{b})$, has all its components bolded to represent that they are vectors.

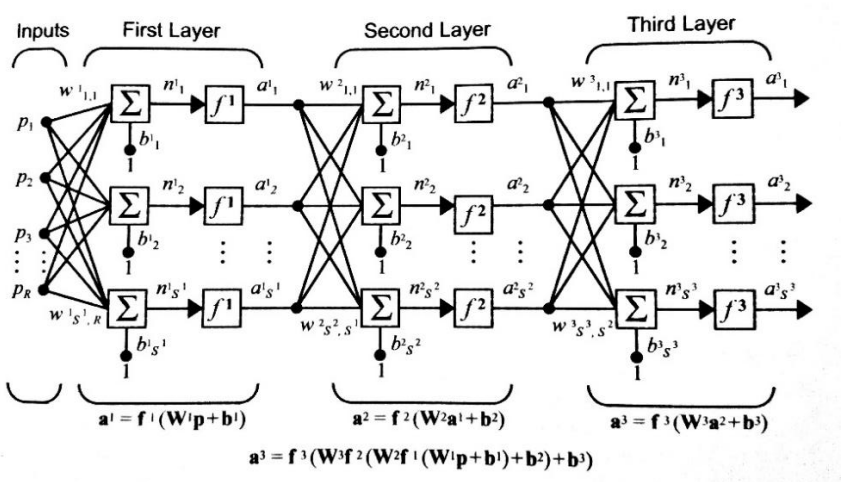


Figure 1.3.5: : Multiple-layers-of-neurons model *[4]*. Here only three layers are shown. The number, however, can be extended to whatever number of layers is desired

There are two major types of neural networks: feedforward and recurrent. All the examples we have looked at so far are feedforward, where the input is piped all the way till the end, i.e. it doesn't get reused at any stage of the process. Recurrent neural network (RNN), by contrast, are ones with feedback, where some of the outputs get connected to the input throughout the process.

An example of an RNN is shown in Figure *1.3.6*. The output of the delay block is the input signal delayed by one-time step. This specific arrangement of a recurrent neuron work such that the future outputs are computed from past values with inputs being initial condition.

RNNs are more powerful than feedforward NNs as they possess the ability to exhibit temporal behavior. In another word, feedforward NN can only implement static input-output mapping, while RNN's can deal with dynamical systems, where time is involved. The feedforward neural networks, however, have a huge amount of literature as they are being investigated by most scholars and researchers, which might be thought of as an advantage.

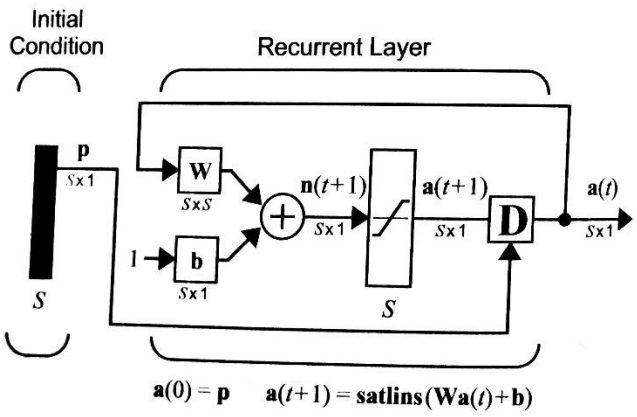


Figure 1.3.6: Recurrent ANN model *[4]*. The delay block takes inputs as initial conditions and shift its input by one which determines the output.

The amount of interest and money being invested in ANN is big and is becoming even bigger [4]. Many applications involve the use of ANNs. Below is a very small list:

1. Automotive: Automatic guidance systems, fuel injection control, automatic breaking systems, …etc.
2. Defense: Weapon steering, target tracking, facial recognition, object discrimination, … etc.
3. Medical: Breast cancer cell analysis, EEG, and EMG analysis, hospital quality improvement, … etc.
4. Robotics: trajectory control, forklift robot, manipulator controllers, …etc.
5. Speech: speech recognition, vowel classification, text to speech synthesis, …etc.
6. Aerospace: High performance autopilots, flight path simulation, aircraft control system, …etc.

Artificial Neural Networks (ANNs) is one very important topic, to which analysis of the biological human control system has led. One can think of ANNs as small artificial brains built to learn and accomplish desired tasks.

1.3.1 Backpropagation Neural Networks

Backpropagation is an abbreviation of "Backward Propagation of Error" [6]. It is a method used to train ANNs. Because the error (difference between predicted and target outputs) is eventually minimized, the algorithm is usually conjoined with an optimization technique.

Backpropagation training involves two phases: the feedforward and the feedback propagation (See Figure 1.3.7) [7]. During the first phase, the teacher-input (a vector in most cases) is fed to a network built with random weights. The output is computed using the values produced by the hidden layer, while the weights of each neuron remain unchanged. In the second phase, the error is computed by subtracting the network predicted output from the teacher-input (target training signal). The error (collected in a vector) is then back-propagated through the hidden layer(s) and minimized using techniques that result in updating the weights of the network. This procedure is repeated until a terminating condition is met (i.e. satisfactory result is achieved). An example of a termination condition would be achieving a value of a mean square error that is smaller than a preset threshold. Figure 1.3.7 shows the mechanism chart of the backpropagation training method. As explained in section 3.4, This study uses the backpropagation algorithm to train different ANNs. The difference between the training approaches is the optimization technique. LM, BR, and SCG are nothing but different methods to minimize the error vector(s). Next, the algorithms of LM, BR, and SCG are explained.

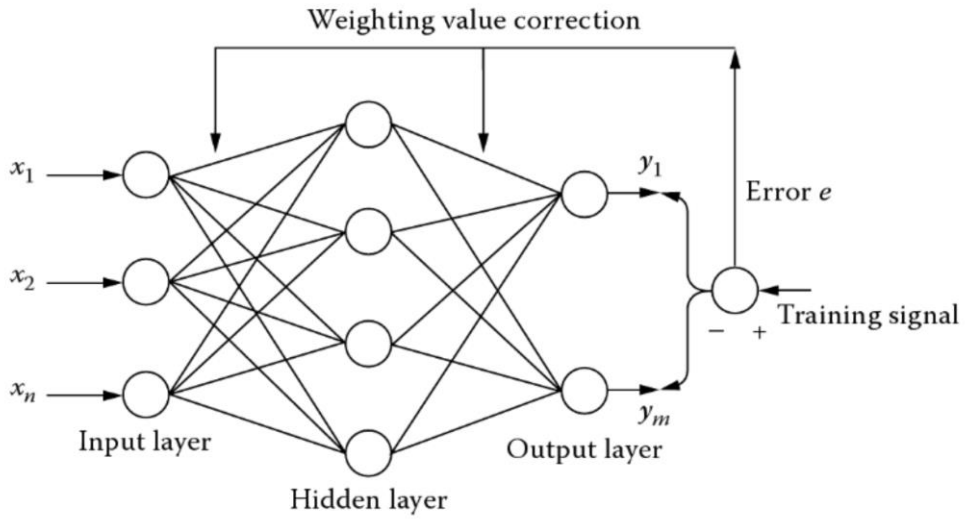


Figure 1.3.7: Backpropagation mechanism chart [7]. For an 'n' input neurons and an 'm' output neurons, the error is produced and back propagated through the hidden layer(s). the weights of the hidden and output neurons are then updated.

1.3.2 Levenberg-Marquardt Backpropagation

Levenberg-Marquardt (LM) is an optimization algorithm. It solves the problem of minimizing non-linear functions [8]. The method is sometimes referred to as the Damped Least Square (DLS) algorithm [9]. The method was first published in 1944 by Kenneth Levenberg and later rediscovered in 1963 by Donald Marquardt [10], [11]. Levenberg-Marquardt algorithm blends the steepest decent and the Gauss-Newton algorithms to produce a faster, more robust optimization technique [9], [8].

The steepest decent algorithms—also called the error backpropagation (EBP)—is one of the most significant methods when it comes to neural networks training [8]. A major issue, however, is born with the discovery of EPB: the method has a very slow convergence rate [8]. Two main reasons cause this problem. First, the step size is constant. This means that, logically, in places where the gradient is steep, a large step size is going to lead the search away from the minimum. Similarly, in places where the

gradient is gentle, a small step size will lead the search to take more time in a place it does not need to be. Figure 1.3.8 shows the search process for small and large step size constants. The second reason comes up when the error function does not have the same curvature in all directions. A function similar to the famous Rosenbrock (shown in Figure 1.3.9) has the classic problem of "error valley" and might slow down the search for an optimum even further [12]. Both of these reasons introduce inefficiency to the EBP algorithm.

The performance, in terms of the discussed issues, of the steepest decent algorithm can significantly be improved using the Gauss-Newton (GN) method [8]. The latter uses the second derivative of the error function to evaluate the curvature and adjust the step size accordingly [8]. In another word, by looking at the second order derivative, Gauss-Newton algorithm finds the proper step size as the gradient of the error changes between steep and gentle. The issue of this method is that it converges only when the second-order approximation is reasonable [8]. Otherwise, the algorithm may mostly diverge.

Levenberg-Marquardt algorithm merges the EBP and the GN. The result is an improved performance where the features of both algorithms are inherited; speed of GN and stability of EPB. LM is more robust than GN as it has the ability to converge in more complex surfaces (of the error function). Similarly, LM beats EBP in speed [8]. The main concept of the algorithm is that it switches to the steepest decent algorithm where the curvature of the error function is complex [8]. When the curvature is proper for quadratic approximation, LM uses GN algorithm [8]. Next, the algorithm is derived.

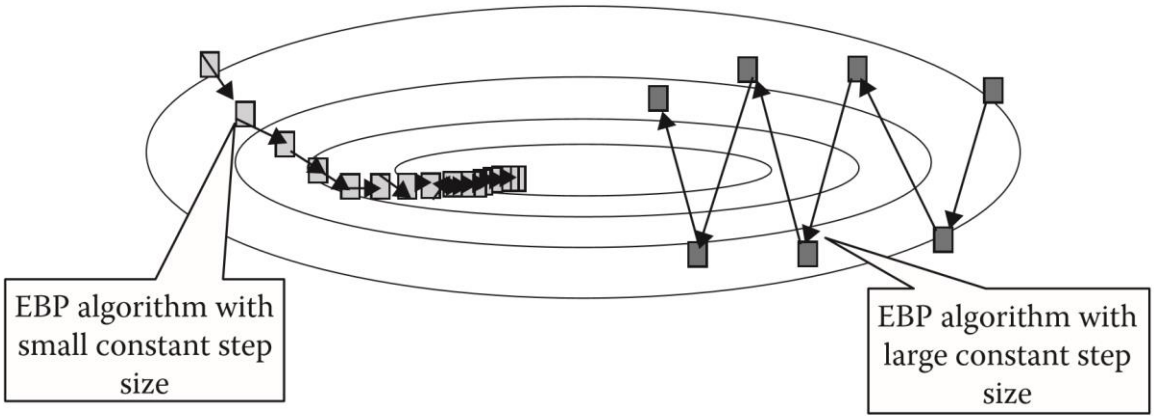
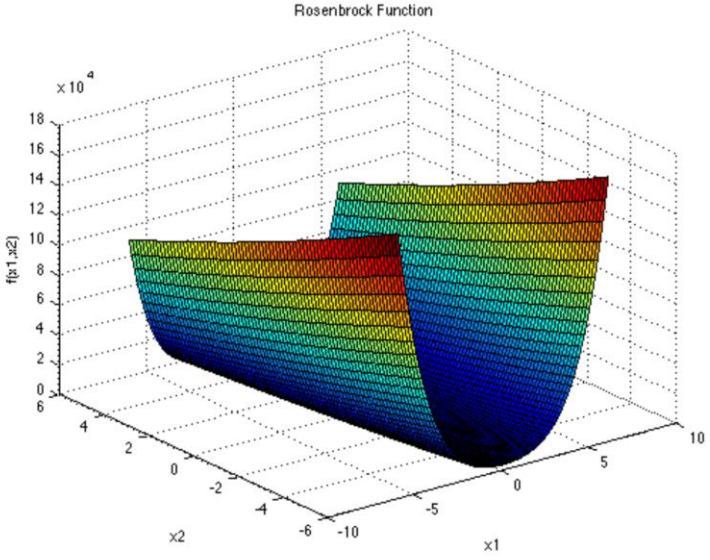


Figure 1.3.8: The EBP with different step size constants. Large step size constants result in an oscillation that slows down the search in places where the gradient is steep. In the other hand, small step size constants, may also slow down the search where the gradient is gentle.



$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

Figure 1.3.9: The Rosenbrock, a popular testing function for optimization problems. It is a unimodal function and the optimum minimum lies in a narrow parabolic valley. It is also called the banana/ valley function [13].

As previously described, the Levenberg-Marquardt algorithm breed the EBP and the GN methods. Therefore, the algorithm, in this section, is derived through these

methods. The following derivation is adopted from [8]. A list of the commonly used indices follows:

- p represents the number of patterns.
- m represents the number of outputs.
- i and j are weight indices.
- k is the index iterations.

The sum of square error is defined in (17). The formula generalizes to all training patterns and network outputs.

$$E(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} \boldsymbol{e_{p,m}}^2 \tag{17}$$

where $\boldsymbol{x}$ is the input vector, $\boldsymbol{w}$ is the weight vector, and $\boldsymbol{e_{p,m}}$ is the training error at an output, m, when applying a pattern, p, and is defined as:

$$\boldsymbol{e_{p,m}} = \boldsymbol{d_{p,m}} - \boldsymbol{o_{p,m}} \tag{18}$$

where $\boldsymbol{d}$ is the target output and $\boldsymbol{o}$ is the predicted (by the network) output vector.

The steepest decent algorithm uses the first-order derivative of the error function. Normally, it is called the gradient, $\boldsymbol{g}$, which is defined in (19).

$$\boldsymbol{g} = \frac{\partial E(\boldsymbol{x}, \boldsymbol{w})}{\partial \boldsymbol{w}} = \left[\frac{\partial E}{\partial \boldsymbol{w_1}} \ \frac{\partial E}{\partial \boldsymbol{w_2}} \ldots \frac{\partial E}{\partial \boldsymbol{w_N}}\right] \tag{19}$$

Knowing, $\boldsymbol{g}$, the weights can be updated using the following:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha \boldsymbol{g}_k \tag{20}$$

where α, is the learning constant (the search step size).

Around the minimum point, the gradient vector will have very small values and therefore, very tiny weight changes. In another word, an asymptotic convergence would be the desired result of a training process involving the steepest gradient decent algorithm.

Newton's method is the first look at the second order derivative. The method assumes that all the gradient components $\boldsymbol{g}_1, \dots, \boldsymbol{g}_N$, are functions of weights and all weights are linearly independent.

$$\begin{cases} g_1 = F_1(w_{\mathbf{1}}, w_{\mathbf{2}}, \dots, w_{\boldsymbol{N}}) \\ g_2 = F_2(w_{\mathbf{1}}, w_{\mathbf{2}}, \dots, w_{\boldsymbol{N}}) \\ \qquad\qquad . \\ \qquad\qquad . \\ \qquad\qquad . \\ g_N = F_N(w_{\mathbf{1}}, w_{\mathbf{2}}, \dots, w_{\boldsymbol{N}}) \end{cases} \tag{21}$$

where $F_1, \dots, F_N$, nonlinearly relates weights to corresponding gradient components.

Expanding each component in (21) using Taylor series and taking the first-order approximation result in (22).

$$\begin{cases} g_1 \approx g_{1,0} + \dfrac{\partial g_1}{\partial w_1}\Delta w_1 + \dfrac{\partial g_1}{\partial w_2}\Delta w_2 + \cdots + \dfrac{\partial g_1}{\partial w_N}\Delta w_N \\ g_2 \approx g_{2,0} + \dfrac{\partial g_2}{\partial w_1}\Delta w_1 + \dfrac{\partial g_2}{\partial w_2}\Delta w_2 + \cdots + \dfrac{\partial g_2}{\partial w_N}\Delta w_N \\ \qquad\qquad . \\ \qquad\qquad . \\ \qquad\qquad . \\ g_N \approx g_{N,0} + \dfrac{\partial g_N}{\partial w_1}\Delta w_1 + \dfrac{\partial g_N}{\partial w_2}\Delta w_2 + \cdots + \dfrac{\partial g_N}{\partial w_N}\Delta w_N \end{cases} \tag{22}$$

From (19), it could be determined that:

$$\frac{\partial g_i}{\partial w_j} = \frac{\partial \left(\frac{\partial E}{\partial w_j}\right)}{\partial w_j} = \frac{\partial^2 E}{\partial w_i \, \partial w_j} \tag{23}$$

Combining (23) and (22) produce:

$$\begin{cases} g_1 \approx g_{1,0} + \frac{\partial^2 E}{\partial {w_1}^2}\Delta w_1 + \frac{\partial^2 E}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial w_1 \partial w_N}\Delta w_N \\ g_2 \approx g_{2,0} + \frac{\partial^2 E}{\partial w_2 \partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial {w_2}^2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial w_2 \partial w_N}\Delta w_N \\ \vdots \\ g_N \approx g_{N,0} + \frac{\partial^2 E}{\partial w_N \partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial w_N \partial {w_2}^2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial {w_N}^2}\Delta w_N \end{cases} \tag{24}$$

In order to find the minimum of a total error function, E, each element of the gradient vector is equalized to zero:

$$\begin{cases} 0 \approx g_{1,0} + \frac{\partial^2 E}{\partial {w_1}^2}\Delta w_1 + \frac{\partial^2 E}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial w_1 \partial w_N}\Delta w_N \\ 0 \approx g_{2,0} + \frac{\partial^2 E}{\partial w_2 \partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial {w_2}^2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial w_2 \partial w_N}\Delta w_N \\ \vdots \\ 0 \approx g_{N,0} + \frac{\partial^2 E}{\partial w_N \partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial w_N \partial {w_2}^2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial {w_N}^2}\Delta w_N \end{cases} \tag{25}$$

Combining (19) with (25) results in:

$$\begin{cases} -\frac{\partial E}{\partial w_1} = -g_{1,0} \approx \frac{\partial^2 E}{\partial {w_1}^2}\Delta w_1 + \frac{\partial^2 E}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial w_1 \partial w_N}\Delta w_N \\ -\frac{\partial E}{\partial w_2} = -g_{2,0} \approx \frac{\partial^2 E}{\partial w_2 \partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial {w_2}^2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial w_2 \partial w_N}\Delta w_N \\ \vdots \\ -\frac{\partial E}{\partial w_N} = -g_{N,0} \approx \frac{\partial^2 E}{\partial w_N \partial w_1}\Delta w_1 + \frac{\partial^2 E}{\partial w_N \partial {w_2}^2}\Delta w_2 + \cdots + \frac{\partial^2 E}{\partial {w_N}^2}\Delta w_N \end{cases} \tag{26}$$

Since there are N equations for N parameters, all the Δw_i can be calculated and, therefore, the weights can be updated iteratively. Equation (26) can be rewritten as:

$$\begin{bmatrix} -g_1 \\ -g_2 \\ . \\ . \\ . \\ -g_N \end{bmatrix} = \begin{bmatrix} -\frac{\partial E}{\partial w_1} \\ -\frac{\partial E}{\partial w_2} \\ . \\ . \\ . \\ -\frac{\partial E}{\partial w_N} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 E}{\partial {w_1}^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial {w_2}^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_N} \\ & . & & \\ & . & & \\ & . & & \\ \frac{\partial^2 E}{\partial w_N \partial w_1} & \frac{\partial^2 E}{\partial w_N \partial {w_2}^2} & \cdots & \frac{\partial^2 E}{\partial {w_N}^2} \end{bmatrix} \times \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ . \\ . \\ . \\ \Delta w_N \end{bmatrix} \tag{26}$$

From (26) the square matrix is called the Hessian matrix:

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 E}{\partial {w_1}^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial {w_2}^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_N} \\ & . & & \\ & . & & \\ & . & & \\ \frac{\partial^2 E}{\partial w_N \partial w_1} & \frac{\partial^2 E}{\partial w_N \partial {w_2}^2} & \cdots & \frac{\partial^2 E}{\partial {w_N}^2} \end{bmatrix} \tag{27}$$

As a result, we have

$$-\boldsymbol{g} = \boldsymbol{H}\Delta\boldsymbol{w} \tag{28}$$

And

$$\Delta\boldsymbol{w} = -\boldsymbol{H}^{-1}\boldsymbol{g} \tag{29}$$

The weights can now be updated using (30)

$$\boldsymbol{w_{k+1}} = \boldsymbol{w_k} - \boldsymbol{H_k}^{-1}\boldsymbol{g_k} \tag{30}$$

Compared to the steepest decent, this method requires computation of the second-order derivative for every gradient component. This process could be very complicated. As

a result, the Gauss-Newton Algorithm simplifies the process by introducing the Jacobian matrix, $\boldsymbol{J}$, which is expressed as:

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \cdots & \frac{\partial e_{1,1}}{\partial w_N} \\ \frac{\partial e_{1,2}}{\partial w_1} & \frac{\partial e_{1,2}}{\partial w_2} & \cdots & \frac{\partial e_{1,2}}{\partial w_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial e_{1,M}}{\partial w_1} & \frac{\partial e_{1,M}}{\partial w_2} & \cdots & \frac{\partial e_{1,M}}{\partial w_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial e_{P,1}}{\partial w_1} & \frac{\partial e_{P,1}}{\partial w_2} & \cdots & \frac{\partial e_{P,1}}{\partial w_N} \\ \frac{\partial e_{P,2}}{\partial w_1} & \frac{\partial e_{P,2}}{\partial w_2} & \cdots & \frac{\partial e_{P,2}}{\partial w_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial e_{P,M}}{\partial w_1} & \frac{\partial e_{P,M}}{\partial w_2} & \cdots & \frac{\partial e_{P,M}}{\partial w_N} \end{bmatrix} \tag{31}$$

By integrating (17) and (19), the gradient vector becomes:

$$\boldsymbol{g} = \frac{\partial E}{\partial w_{\boldsymbol{i}}} = \frac{\partial \left(\frac{1}{2} \sum_{p=1}^{P} \sum_{m=1}^{M} \boldsymbol{e_{p,m}}^2 \right)}{\partial w_{\boldsymbol{i}}} = \sum_{p=1}^{P} \sum_{m=1}^{M} \left(\frac{\partial e_{p,m}}{\partial w_i} e_{p,m} \right) \tag{32}$$

Combining (31) and (32) produce the relationship between the Jacobian matrix and the gradient vector in (33).

$$\boldsymbol{g} = \boldsymbol{J}\boldsymbol{e} \tag{33}$$

where $\boldsymbol{e}$, is the error vector of the form:

$$\boldsymbol{e} = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ . \\ . \\ e_{1,M} \\ . \\ . \\ e_{P,1} \\ e_{P,2} \\ . \\ . \\ e_{P,M} \end{bmatrix} \tag{34}$$

Substituting (17) into (18) produce the formula to calculate the elements in the Hessian matrix. The formula is described as:

$$h_{i,j} = \frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{\partial^2 \left(\frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} \boldsymbol{e_{p,m}}^2\right)}{\partial w_i \partial w_j} = \sum_{p=1}^{P}\sum_{m=1}^{M}\left(\frac{\partial e_{p,m}}{\partial w_i}\frac{\partial e_{p,m}}{\partial w_j} + S_{i,j}\right) \tag{35}$$

Where $S_{i,j}$ is:

$$S_{i,j} = \sum_{p=1}^{P}\sum_{m=1}^{M}\left(\frac{\partial e_{p,m}}{\partial w_i}\frac{\partial e_{p,m}}{\partial w_j}\right) \tag{36}$$

The basic assumption of Newton's method is that $S_{i,j}$ is very small [8]. Therefore, the relationship between the Hessian matrix, $\boldsymbol{H}$, and the Jacobian matrix, $\boldsymbol{J}$, is:

$$\boldsymbol{H} \approx \boldsymbol{J}^T\boldsymbol{J} \tag{37}$$

Combining (30), (33), and (37), the weight updating formula is presented as:

$$\boldsymbol{w_{k+1}} = \boldsymbol{w_k} - (\boldsymbol{J_k}^T\boldsymbol{J_k})^{-1}\boldsymbol{J_k}\boldsymbol{e_k} \tag{38}$$

It is noticeable that the Gauss-Newton algorithm is better than the standard Newton's algorithm in the sense that the Hessian matrix is not needed for the calculation. Gauss-Newton algorithm, however, still has the problem of convergence discussed earlier

in this section. In another word, from a mathematical point of view, $\boldsymbol{J^T J}$ may not be invertible. This is where the Levenberg-Marquardt algorithm come to use.

The Levenberg-Marquardt algorithm introduces a new approximation to the Hessian matrix. The approximation makes sure that $\boldsymbol{J^T J}$ is always invertible. The new approximation is presented as:

$$\boldsymbol{H} \approx \boldsymbol{J^T J} + \mu \boldsymbol{I} \tag{39}$$

where μ, is the combination coefficient (always positive) and $\boldsymbol{I}$, is the identity matrix.

The new approximation makes certain that the main diagonal of the (39) is positive (larger than zero). As a result, the approximated matrix will always be invertible. Combining (38) and (39) produces the formula for updating the weights for the LM method:

$$\boldsymbol{w_{k+1}} = \boldsymbol{w_k} - (\boldsymbol{J_k}^T \boldsymbol{J_k} + \mu \boldsymbol{I})^{-1} \boldsymbol{J_k e_k} \tag{40}$$

It is mentioned earlier in this section that the LM method "blends both GN and EBP algorithms." This can be seen as the combination coefficient is swept from very small values to very large values. If μ is close to zero, then (40) is approaching (38), which is the Gauss-Newton's algorithm. Similarly, if μ is very large, (40) can be interpreted as the learning coefficient of the steepest decent method:

$$\alpha = \frac{1}{\mu} \tag{41}$$

LM is a fast and stable training approach [8]. It, however, because it uses the Jacobian matrix, assumes that performance is a Mean of Squared Errors (MSE) or a Sum of Squared Errors (SSE) [14]. This limits the performance function to be one of the two,

which may not lead to the best generalization. Next, the Bayesian regularization backpropagation method is described.

1.3.3 Bayesian Regularization Backpropagation

One large issue of designing neural networks is overfitting. This problem refers to the ability of a trained neural network to generalize to a new set of data (different than the one used for training). In neural network design, a concept, often referred to as the Ockham's Razor concept, states that "for a network to be able to generalize, it should have fewer parameters than there are data points in the training set" [4]. In another word, to achieve good generalization, the complexity of a trained neural network needs to match the complexity of the targeted system for classification [4], [5], [15], [7]. Therefore, it is important to figure out the number of free parameters (weights and biases) which, in turn, affect the number of neurons used in a network. The idea is that if the network is too complex, it might learn unnecessary patterns, which do not generalize to all the population (new data sets).

There are many approaches to this problem—all of which attempt to find the simplest network to fit the data [4]. According to the authors of [4], two methods have proven solid in terms of practicality: early stopping and regularization. Both of these methods tackles the problem by restricting the magnitude of the weights (free parameters), which, in turn, restricts the complexity of the network. The two approaches reduce the complexity of the network without changing the number of neurons with which the training started. Regularization is the category under which the Bayesian algorithm lie.

The main idea of the method is to introduce a term, to the error function, which penalizes the complexity of the neural network. This term smooths the error function and is called the regularization term. The sum squared error, alternatively, may be represent as:

$$E = \sum_{q=1}^{Q} \left(\boldsymbol{t}_q - \boldsymbol{a}_q\right)^T - \left(\boldsymbol{t}_q - \boldsymbol{a}_q\right) \tag{42}$$

where $\boldsymbol{t}_q$, is the target output and $\boldsymbol{a}_q$ is the anticipated (predicted) output by the network and q is the number of sample points.

The regularization term is written as:

$$\beta E_D + \alpha E_w = \beta \sum_{q=1}^{Q} \left(\boldsymbol{t}_q - \boldsymbol{a}_q\right)^T - \left(\boldsymbol{t}_q - \boldsymbol{a}_q\right) + \alpha \sum_{i=1}^{n} {x_i}^2 \tag{43}$$

Here $\frac{\alpha}{\beta}$ controls the complexity (smoothness) of the output. As $\frac{\alpha}{\beta}$ increases, the smoother the output is.

Penalizing the sum squared error is similar to reducing the number of neurons in the network [4]. Changing the weights of an ANN affects the slope of the network's output. For example, take the neural network constructed in Figure 1.3.10. The predicted output of the network is presented in Figure 1.3.11. Changing one of the weights from zero to two changed the slope of the network's output. As the weight took larger values, the slope of the output function has changed. As a result, the output of the network is more likely to overfit the targeted output. In conclusion, regularizing the weights restricts the complexity of an ANN and, therefore, produces a fit that is more capable to be generalized.

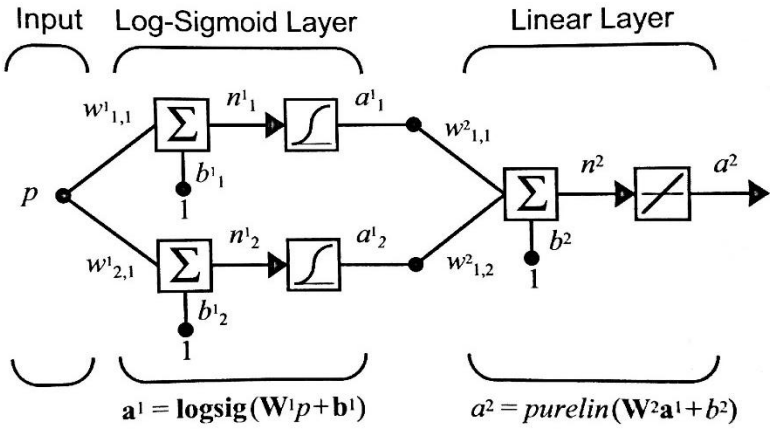


Figure 1.3.10:Neural Network structure [4].

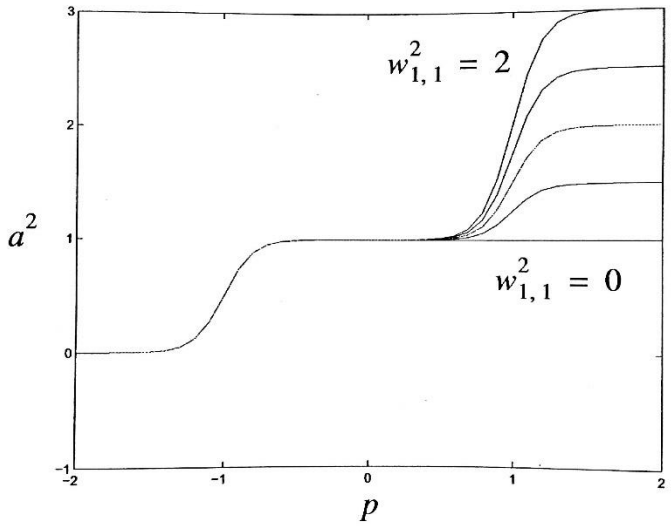


Figure 1.3.11: The output of the network in Figure 1.3.10. Here a, is the network's output and p is the teacher input. Changing one of the weights has led to a large change in the slop of the output function [4].

The key to success in the regularization technique is choosing the best regularization ratio, $\frac{\alpha}{\beta}$, [4]. In Figure 1.3.12, a 1-20-1 ANN is trained. Twenty-one noisy data points are used in the example [4]. The targeted function is a sine wave. It is

obvious that the regularization ratio of 0.01 produced the best fit. Above and below this value, the fits were either too smooth or over-fitted.

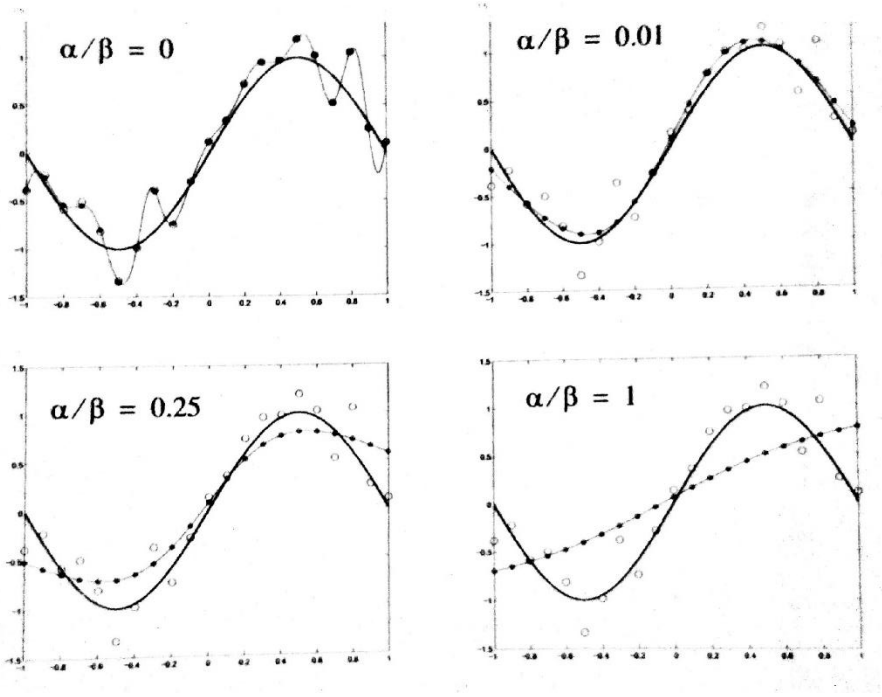


Figure 1.3.12:The effect of the regularization factor on neural filling example. The bold line is the targeted function. The big hollow circles represent the noisy samples and the think line is the response of the network to the data set. The example and figures are adopted from [4].

Several techniques can be used to decide the best value for the regularization ratio. The one used for this study, is the Bayesian Regularization (BR) algorithm. The algorithm inherited the name of the founder of the corner stone by which the concept is build, Thomas Bayes [4]. He found the famous Bayes' Theorem that computes the conditional probability of an even 'A' given a condition 'B' has occurred. The formula of his finding is recalled in (44).

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{44}$$

In 1992, David MacKay developed an approach to use Bayes' theorem in neural network training [16]. The approach brings Neural Network training to the statistical

probability framework. This framework is useful in many aspects other than the regularization ratio selection [4]. Hagan, Demuth, Beale, and Jesus divides the framework into two levels. The analysis follows, are adopted from [4].

Level I begins with assuming that the weights of an artificial neural network are random variables. Next, the conditional probability of the weights given the data points is formed. Finally, the weights which maximize the formula are found and chosen. This probabilistic relation is presented as:

$$P(\boldsymbol{x}|D,\alpha,\beta,M) = \frac{P(D|\boldsymbol{x},\beta,M)P(\boldsymbol{x}|\alpha,M)}{P(D|\alpha,\beta,M)} \tag{45}$$

where, $\boldsymbol{x}$, is the vector containing all the weights and biases in the network, D, is the teacher input/output α and β are the parameters associated with the density functions, $P(D|\boldsymbol{x},\beta,M)$ and $P(\boldsymbol{x}|\alpha,M)$. M is the architecture of the designed network (i.e. the number of layers and neurons).

Because of the importance of this concept, each term in (45) is looked at next. The probability density of the data, $P(D|\boldsymbol{x},\beta,M)$, given the weight vector, $\boldsymbol{x}$, the parameter, β, and the network model, M, is presented in (47). The target output of a system is represented as:

$$\boldsymbol{t}_q = \boldsymbol{g}(p_q) + \varepsilon_q \tag{46}$$

where $g(.)$, is some unknown function, ε_q, is a noise source, $\boldsymbol{t}_q$, is the target output, and p_q, is a corresponding input.

Assuming the noise term in (46) is independent and has Gaussian distribution, the probability density becomes:

$$P(D|\boldsymbol{x},\beta,M) = \frac{1}{Z_D(\beta)}\, e^{-\beta E_D} \tag{47}$$

where $\beta = {}^{1}/_{2\sigma_\varepsilon{}^2}$, $\sigma_\varepsilon{}^2$ is the variance of each element of ε_q, E_D is the squared error as defined in (42), and:

$$Z_D(\beta) = (2\pi\sigma_\varepsilon{}^2)^{N/2} = \left({}^{\pi}/_{\beta}\right)^{N/2} \tag{48}$$

As discussed earlier, the weights are selected such as (47) is maximized. In this case (Gaussian Distribution), maximizing the function requires minimizing the squared error E_D.

The second term in (45) is $P(\boldsymbol{x}|\alpha,M)$. It includes the knowledge of the weights prior to collecting data. Therefore, this term is called the prior density. If we assume that the weights are very small (close to zero), a zero-mean prior density may be selected:

$$P(\boldsymbol{x}|\alpha,M) = \frac{1}{Z_w(\alpha)}\, e^{-\beta E_w} \tag{49}$$

where $\alpha = {}^{1}/_{2\sigma_w{}^2}$, $\sigma_w{}^2$ is the variance of each of the weights, E_w is the sum squared of the weights as defined in (43), and:

$$Z_w(\alpha) = (2\pi\sigma_w{}^2)^{n/2} = ({}^{\pi}/_{\alpha})^{n/2} \tag{50}$$

where n, is the number of weights and biases in the network.

The last term in (45) is called the evidence: $P(D|\alpha,\beta,M)$. This normalizing term does not involve the weights vector, $\boldsymbol{x}$. Therefore, since the goal is to maximize (45), it is not of concern at this point.

Putting everything together (i.e. (45), (47), and (49)) the density function can be rewritten as:

$$P(\boldsymbol{x}|D,\alpha,\beta,M)=\frac{\frac{1}{Z_w(\alpha)}\frac{1}{Z_D(\beta)}\exp[-(\beta E_D+\alpha E_w)]}{Normalizing\ factor} \tag{51}$$

$$=\frac{1}{Z_F(\alpha,\beta)}\exp\big(-F(\boldsymbol{x})\big)$$

where $Z_F(\alpha,\beta)$ is a function of α and β and $F(\boldsymbol{x})$ is the regularization performance index as defined (43). Recall that the goal of this framework is to maximize the density function, $P(\boldsymbol{x}|D,\alpha,\beta,M)$. In order to do that, the performance index $F(\boldsymbol{x})=\beta E_D+\alpha E_w$.

The performance index, which is used regularize the weights in a network, is derived from the Bayesian Theorem. This statistical framework granted actual meanings to the parameters α and β. For example, β is inversely proportional to the variance. If the variance of the noise is large, this would lead β to be small. As a result, the performance ratio, α/β, is large, which, in turn, will force the weights to be small and the overall function to be smooth. On the other hand, α is inversely proportional to the variance in the prior distribution. In the case of large value of this variance, α will be small. This is an indication of uncertainty with regards to the chosen weights (i.e. the weights might be large). As a result, the performance ratio will be small and, therefore, more variation allowance for the network (i.e. more complexity). To recall this discussion, see Figure 1.3.12.

Level II of the framework provides the analysis to estimate the parameters α and β, from the data. Using the Bayesian analysis, $P(\alpha,\beta|D,M)$ can be written as:

$$P(\alpha,\beta|D,M)=\frac{P(D|\alpha,\beta,M)P(\alpha,\beta|M)}{P(D|M)} \tag{52}$$

This formula has the same format as (45). Following the same logic, (52) is maximized by maximizing $P(D|\alpha,\beta,M)$. This term, however, is identical to the normalizing factor (evidence) of (45). With that in mind, $P(D|\alpha,\beta,M)$ can be solved for:

$$P(D|\alpha,\beta,M)=\frac{\left[\frac{1}{Z_D(\beta)}\exp(-\beta E_D)\right]\left[\frac{1}{Z_w(\alpha)}\exp(-\alpha E_w)\right]}{\frac{1}{Z_F(\alpha,\beta)}\exp\big(-F(\boldsymbol{x})\big)} \tag{53}$$

$$=\frac{Z_F(\alpha,\beta)}{Z_D(\beta)Z_w(\alpha)}$$

From (48) and (50) $Z_D(\beta)$ and $Z_w(\alpha)$ can be found. $Z_F(\alpha,\beta)$, in the other hand, is unknown. However, since the objective function has a quadratic shape around its minimum point, $\boldsymbol{x}^{MP}$, (where the gradient is zero), a second order Taylor series expansion can be used to solve for $Z_F(\alpha,\beta)$:

$$F(x)\approx F(x^{MP})+\frac{1}{2}(x-x^{MP})^T H^{MP}(x-x^{MP}) \tag{54}$$

where H^{MP}, is the hessian matrix of $F(x)$ evaluated at x^{MP}. Substituting (54) into (51) provides the solution:

$$Z_F(\alpha,\beta)\approx(2\pi)^{n/2}(\det((H^{MP})^{-1}))^{1/2}\exp(-F(x^{MP})) \tag{55}$$

Substituting (55) into (53) provides the optimum solution for α and β at the minimum point:

$$\alpha^{MP}=\frac{\gamma}{2E_w(x^{MP})}\ and\ \beta^{MP}=\frac{N-\gamma}{2E_D(x^{MP})} \tag{56}$$

where $\gamma=n-2\alpha^{MP}tr(H^{MP})^{-1}$ is the effective number performance and n, is the total number of parameters in the network (weights and biases).

1.3.4 Scaled Conjugate Gradient Backpropagation

The Scaled Conjugate Gradient (SCG) algorithm is developed with the claim that large-scale problems are best handled by a group of optimization techniques called the "conjugate gradient methods" [17], [18], [19]. The scaled conjugate gradient algorithm is similar to the LM method in the sense that it avoids the calculation of the Hessian matrix (second order derivative) [4]. As discussed in the LM algorithm, the calculation of the Hessian matrix can be very complicated and may abuse the computation power. If the gradient (first derivative) has n elements, the Hessian (second derivative has n^2 elements [4]. The conjugate gradient method provides quadratic termination without computing the Hessian.

The steepest decent algorithm, as discussed earlier, performs linear searches at each iteration. The directions of each search are orthogonal. It turns out that searching in this direction does not provide the best (fastest) results. The conjugate direction is one way of guaranteeing quadratic termination (solves the problem).

It can be shown that if a sequence of exact linear searches is made along any set of conjugate direction, the minimum of any quadratic function with n parameters can be found in a maximum of n searches [20], [21]. The solution, therefore, is to find a way to force the search in the conjugate direction(s).

A set of vectors $\{\boldsymbol{p}_k\}$ is mutually conjugate with respect to a positive definite Hessian Matrix, H, if and only if:

$$\boldsymbol{p}_k{}^T \boldsymbol{H} \boldsymbol{p}_j = 0 \qquad k \neq j \tag{57}$$

Similar to orthogonal vectors, there are an infinite number of conjugate vectors spanning an n-dimensional space. This, however, includes the hessian matrix. Therefore, the conjugacy condition in (57) must be stated without $\boldsymbol{H}$. Recall that for quadratic functions:

$$\nabla \mathrm{F}(\mathrm{x}) = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{d} \tag{58}$$

$$\nabla^2 \mathrm{F}(\mathrm{x}) = \boldsymbol{H} \tag{59}$$

The combination of these formulas presents the change in gradient at iteration $k + 1$:

$$\Delta g_k = g_{k+1} - g_k = (\boldsymbol{H}\boldsymbol{x}_{k+1} + \boldsymbol{d}) - (\boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{d}) = \boldsymbol{H}\Delta\boldsymbol{x}_k \tag{60}$$

$$\boldsymbol{x}_k = (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \alpha_k \boldsymbol{p}_k \tag{61}$$

where α_k is chosen to minimize F(x) in the direction $\boldsymbol{p}_k$.

Now the conjugacy condition can be restated as:

$$\alpha_k \boldsymbol{p}_k{}^T \boldsymbol{H}\boldsymbol{p}_j = \Delta\boldsymbol{x}_k{}^T \boldsymbol{H}\boldsymbol{p}_j = \Delta\boldsymbol{g}_k{}^T \boldsymbol{p}_j = 0 \qquad k \neq j \tag{62}$$

(62) restates the conjugacy condition in terms of change in the gradient at successive iterations. In words, the search directions will be conjugate if they are orthogonal to the change in gradient. It is worth to mention that $\boldsymbol{p}_0$ is arbitrary and $\boldsymbol{p}_1$ can be any vector that is orthogonal to $\Delta\boldsymbol{g}_0$. Therefore, there are an infinite number of conjugate vectors. It is common to start in the steepest decent direction:

$$\boldsymbol{p}_0 = -\boldsymbol{g}_0 \tag{63}$$

Next, at each iteration, a vector $\boldsymbol{p}_k$, that is orthogonal to $\{\Delta\boldsymbol{g}_0, \Delta\boldsymbol{g}_1, \ldots\ldots, \Delta\boldsymbol{g}_{k-1}\}$. This procedure can be simplified to iterations of the form [21]:

$$\boldsymbol{p}_k = -\boldsymbol{g}_k + \beta_k \boldsymbol{p}_{k-1} \tag{64}$$

β_k is a scaler that can be chosen in different methods. All of which produce equivalent results when it comes to quadratic functions [21]. The most common methods are [21] :

$$\beta_k = \frac{\Delta \boldsymbol{g}_{k-1}{}^T \boldsymbol{g}_k}{\Delta \boldsymbol{g}_{k-1}{}^T \boldsymbol{p}_{k-1}} \tag{65}$$

$$\beta_k = \frac{\Delta \boldsymbol{g}_k{}^T \boldsymbol{g}_k}{\Delta \boldsymbol{g}_{k-1}{}^T \boldsymbol{g}_{k-1}} \tag{66}$$

$$\beta_k = \frac{\Delta \boldsymbol{g}_{k-1}{}^T \boldsymbol{g}_k}{\Delta \boldsymbol{g}_{k-1}{}^T \boldsymbol{g}_{k-1}} \tag{67}$$

Chapter 2: Literature Review

2.1 Detection and Processing of EMG

The action stage of the generation of EMG result in an electric dipole [22]. This flow of electrical potential travels through the muscle fibers, which allows its detection (typically) using "bipolar electrodes configuration and differential amplification" [2]. Figure *2.1.1*illustrates the use of electrodes to detect the EMG signal through skin tissues.

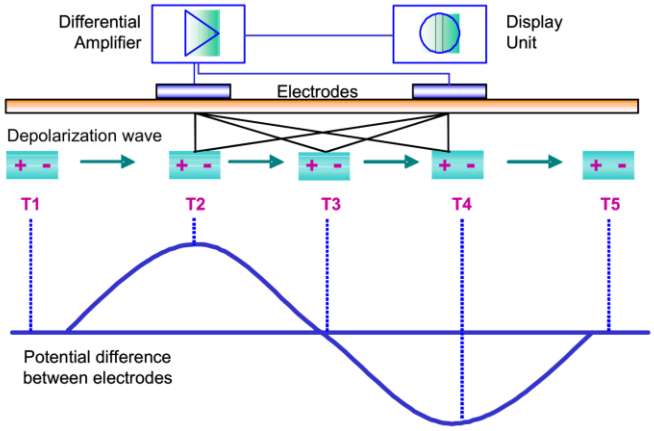


Figure 2.1.1: illustration of using bipolar electrodes and differential amplification to detect EMG signal. The depolarization waves an electric dipole (two opposite charges. Since the skin tissue is conductive, the electric dipole can be easily detected using bipolar electrodes *[23]*.

The interest in EMG detection and processing started with the eldest documented experiment. In 1666, Francesco Redi discovered a highly specialized muscle in the electric ray fish [24]. Throughout the years, the machines to process surface electromyography (sEMG) signals became stronger in terms of their computing power. Companies such as Noraxon and Mathworks developed hardware and software tools to help detect and process sEMG signals [25], [26]. Hardware kits such as Desktop DTS and Telemyo DTS by Noraxon are developed to precisely detect the signal, while software such as the signal processing toolbox and the Arduino support package by Mathworks provide advanced tools to process the signal [27], [28].

2.2 Classification of EMG signal

Artificial Neural Networks (ANNs) approaches to identify sEMG signals has recently surfaced. It makes sense for the two to correlate as ANNs reassemble their biological counterparts. Many algorithms can be used to train an artificial neural network. Previous literature successfully tested training algorithms such as Time Delay Neural Networks (TDNN), NueCube spiking neural networks (SNN), backpropagation based neural networks (BNN) [29], [30], [31]. Because it provides more information, most studies use kinematical features to enhance their classification of the EMG signal [29], [32], [33]. Classification using neural networks, has been broadly explored and, in many cases, has led to success in mapping out the motion and motion intend. The accuracy, however, of trained networks deteriorates when used on different subjects, and/or the same subject, but at different times [34].

Several attempts approach the classification problem using fuzzy logic. Fuzzy logic has the ability to imply human-expert decision making on given inputs [15]. Based on programmed premises and conclusions (rules), the fuzzy system makes decision similar to what a human (expert in the subject) would take [15]. Some research combines fuzzy logic with other techniques to help classification of the signal. These techniques vary between feature extraction [35], hybrid neuro-fuzzy analysis [36], and others.

Current studies developed features to be extracted from the EMG signal. This approach is aimed to minimize the number of channels needed for successful classification of the myoelectric signal (sEMG) [37]. A number of studies have analyzed the features and looked into finding the optimal combination [38].

Chapter 3: Proposed Approach

3.1 Overview

This study attempts to detect, analyze and classify an sEMG signal, collected from seven different locations of the forearm. First, an sEMG is recorded and preprocessed. Next, a total of seventeen features are extracted from each channel. Finally, four different ANNs are trained to classify the sEMG signal based on the extracted features. Finally, the results are analyzed and compared.

3.2 Detection and preprocessing

A box is built in the laboratory using sensing circuitry boards, potentiometers, and a power supply circuit. The box functions as an electromyogram (sEMG sensor). The total amount of channels it can support is ten. This, however, can be increased by upgrading the box to handle another set of channels (up to twenty).

Inside the box; two circuit-boards, similar to the one in Figure 3.2.1, serve as the core sensing elements. A dipolar sEMG signal is sensed with two electrodes which are positioned in a proximity of each other. The differential input is, first, amplified using a non-inverting terminal instrumentation amplifier (LM324). Figure 3.2.2shows the circuit diagram of the instrumentational amplifier. Next, the amplified signal is sent to a notch filter to eliminate the noise caused by the electronic components such as the power supply [39]. The circuit diagram of the designed notch filter is shown in Figure 3.2.3. Finally, a Chebyshev type II passband filter is used to reduce the noise caused by the interference of other internal (biological) signals. The passband is built using a high-pass filter cascaded with a low-pass filter.

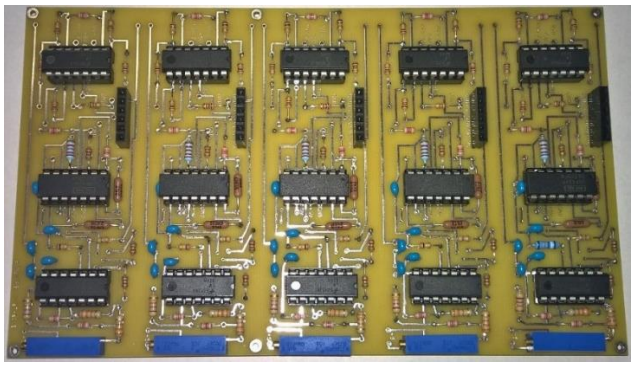


Figure 3.2.1: The sensing circuitry. Each vertical set of components represent one channel resulting five channel per board.

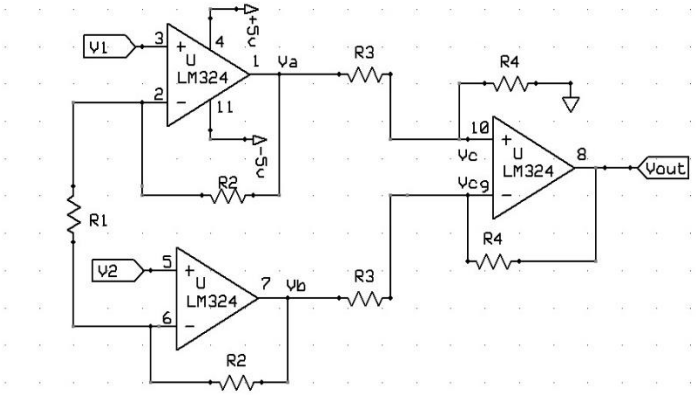


Figure 3.2.2: Instrumentational amplifier diagram. Adopted from [39].

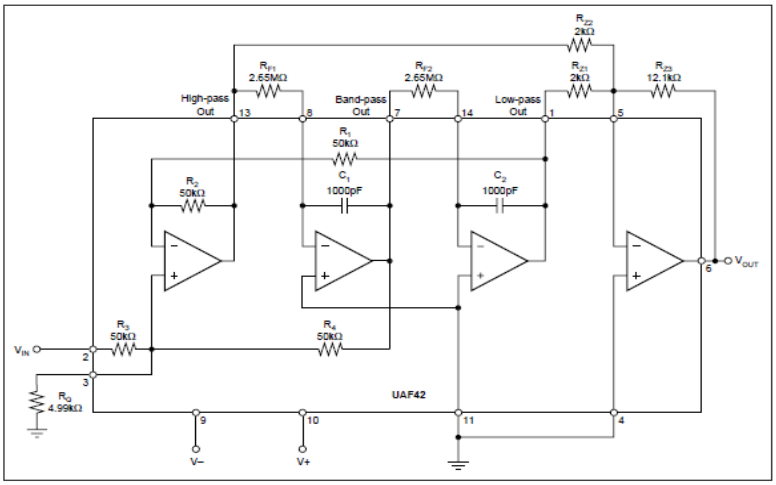


Figure 3.2.3: designed circuit diagram of the notch filter

On the sides of the box, ten potentiometers are connected to the two boards (See Figure 3.2.4). The potentiometers serve to position the zero line of the sEMG signal. The

reason is the limitation on the Arduino microcontroller, which only reads values within the range 0_v to 5_v.

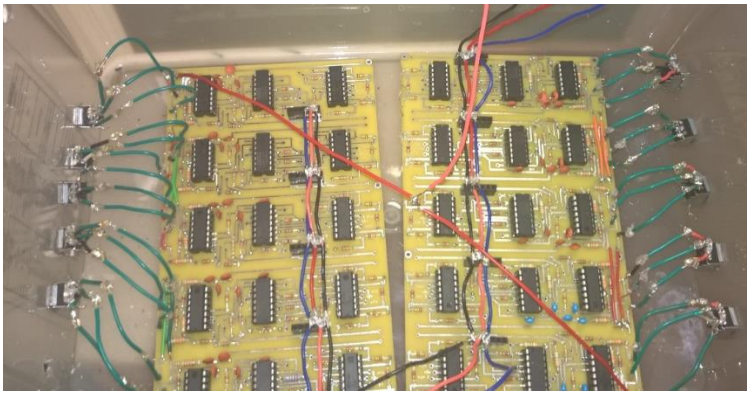


Figure 3.2.4: Inside the box. The green wires are used to connect the potentiometers with the boards.

In order to power the circuit boards, a $+5_v$ and a -5_v DC is needed. Therefore, a voltage divider and regulator is used. Figure 3.2.5and Figure 3.2.6 show the circuit diagram and the actual board respectively. U3 is a 10_v regulator, while U1 and U2 are both 5_v regulators. In order to reduce the amount of noise caused by the AC power, a 12_v DC battery is used to drive the circuitry.

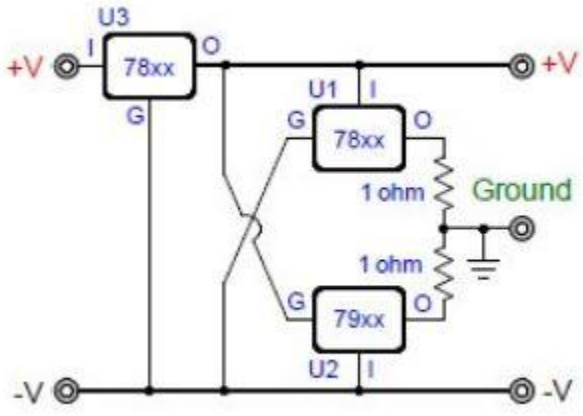


Figure 3.2.5: Circuit diagram of the power supply. U3 is a 10v regulator, while U1 and U2 are both 5v regulators [39]

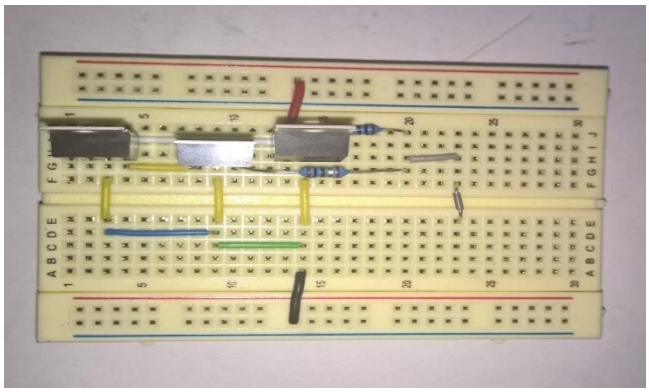


Figure 3.2.6: The actual power supply circuit. To the far left, is the 10v regulator and the two 5v regulators are to the right.

An Arduino MEGA 2560 along with SIMULINK™ (by MATHWORKS) are used to collect and record the sEMG signals. The output of each channel is connected to a different Arduino analog pin. Next, the signal is sent to SIMULINK™ for collection. Figure 3.2.7 shows the SIMULINK™ model for a single channel. In order to record the data for seven channels, the model, shown in Figure 3.2.7, is replicated six more times.

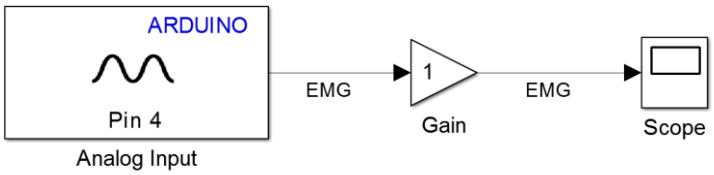


Figure 3.2.7: SIMULINK model for a single channel. The "ARDUINO" box receives the signal coming out from one of the channels at one of the circuit boards and send it to a "gain" block. The gain is used, as necessary, to strengthen the signal. Finally, a "scope" sinks the signal and displays it.

The final step of preprocessing is offset removal. After collecting an sEMG signal, the mean of each channel is subtracted. For a random variable, X, and a number of scalar observations, N, the mean is calculated by:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} X_i. \qquad (1)$$

Once the mean is calculated, it gets subtracted resulting a mean of zero. This process is called detrending. Next, the data is ready for the features to be extracted.

3.3 Features extraction

Seventeen features are adopted in this study to extract the dynamics from an sEMG signal. The hypothesis is that with more features, a larger amount of information is extracted and used for training. In pursuit of that, this study kicked off with seventeen features. Some of the features are developed and others are adopted. Each of the features is designed as a sliding window with a width of 100 sample points. The size of the window is picked randomly. Next, the features used in this research are described.

3.3.1 Band power

Band power is "the sum of the absolute squares of its time-domain samples divided by the signal length" [40]. It is, in another word, the square of the root mean square value (RMS). This feature has not been reported in previous research. For a random variable, X, and a number of samples, N, Band Power is given by:

$$BP = \frac{1}{N}\sum_{i=1}^{N}|X_i|^2. \tag{2}$$

3.3.2 Integrated EMG [38]

Integrated EMG (IEMG) is the integral (summation for discrete signals) of the absolute value of an EMG signal. It is "related to the sEMG sequence firing point" [38]. IEMG is given by (3).

$$IEMG = \sum_{i=1}^{N}|X_i|. \tag{3}$$

3.3.3 Mean Absolute value [38]

The mean absolute value (MAV) of an EMG is the sum of the absolute values of a scalar finite number of samples, X, divided by the total number of samples, N. MAV is one of the most commonly used features in identifying sEMG due to its simplicity [38]. The feature is given by:

$$MAV = \frac{1}{N}\sum_{i=1}^{N}|X_i|. \tag{4}$$

3.3.4 Modified Mean Absolute Value 1 [38]

The Modified Mean Absolute Value 1 (MMAV1) is an extension of MAV. For this feature, a weighting index is added. Samples within the range 25% to 75% of the EMG signal are weighted more than of those outside the range. This feature assumes, and tries to avoid, the bias in measurements, which might affect the first and the last 25% of the samples. MMAV1 is given by (5).

$$MMAV1 = \frac{1}{N}\sum_{i=1}^{N}\omega_i|X_i|, \tag{5}$$

where,

$$\omega_i = \begin{cases} 1 & if\ 0.25N \le i \le 0.75N \\ 0.5 & Otherewise \end{cases}$$

3.3.5 Modified Mean Absolute Value 2 [38]

The Modified Mean Absolute Value 2 (MMAV2) shares the same goal with MMAV1. MMAV2 is designed to have a smoother transition than MMAV1. Equation (6) describes this feature.

$$MMAV1 = \frac{1}{N}\sum_{i=1}^{N} \omega_i |X_i|, \tag{6}$$

where,

$$\omega_i = \begin{cases} 1, & if\ 0.25N \le i \le 0.75N \\ \dfrac{4i}{N}, & if\ i < 0.25N \\ \dfrac{4(i-N)}{N}, & if\ i > 0.75N \end{cases}$$

3.3.6 Mean Absolute Deviation [41]

The Mean Absolute Deviation (MAD) of a set of data is the average distance between a sample point and the mean. The mathematical form of this feature is given below:

$$MAD = \frac{1}{N}\sum_{i=1}^{N} |X_i - \bar{X}|, \tag{7}$$

where $\bar{X}$ is the mean of the population.

3.3.7 Mean

The mean is a measure of the central tendency of a random variable. It is given by (8). This feature has not been reported in previous studies.

$$Mean = \frac{1}{N}\sum_{i=1}^{N} X_i, \tag{8}$$

3.3.8 Mean Frequency

The mean frequency is a measure of the mean normalized frequency of the power spectrum of a time domain signal [42]. The power spectrum is a measure of a signal's

"energy per unit time" [43]. It is generated using the Discrete Fourier Transform (DFT). This feature, too, has not been reported in previous research.

3.3.9 Median Frequency

Similar to mean frequency, the median frequency is a normalized estimate that is calculated from the power spectrum of a given signal. The process of calculating the median frequency is similar to the mean frequency, except that the median is the value computed at the end instead of the mean. This feature, also, has not been reported in previous research.

3.3.10 Root Mean Square [38]

The Root Mean Square (RMS) is, as the name suggests, the square root of the mean of a signal raised to the second power [44]. RMS is sometimes called the quadratic mean. This feature has been studied in number of research related to sEMG [45]. The mathematical formula is given by (9).

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} {X_i}^2}, \tag{9}$$

3.3.11 Slope Sign Change [38]

The slope sign change (SSC) is an indicator of the slope sign change over three consecutive segments. Previous research used this method in classification of sEMG signal [38]. SSC is described as follows:

$$SSC = \sum_{i=2}^{N-1} \{f[(X_i - X_{i-1})(X_i - X_{i+1})]\}, \tag{10}$$

where,

$$f(x) = \begin{cases} 1, & if\ x \geq Threshold \\ 0, & otherwise \end{cases}$$

3.3.12 Simple Square Integral [38]

This feature uses the "energy of sEMG signals" [38]. The Simple Square Integral (SSI) is the sum of the squares of the absolute values of the samples (for discrete time). A number of studies used the feature to classify sEMG signals. SSI is described by (11).

$$SSI = \sum_{i=1}^{N} |X_i|^2, \tag{11}$$

3.3.13 Standard Deviation

The standard deviation (STD) of a signal is a measure of how far the observations, of a sample, are from the mean [46]. This feature is not reported in previous studies related sEMG classification. STD is expressed as follows:

$$STD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N} |X_i - \mu|^2}, \tag{12}$$

where, μ is the mean of the random variable, X.

3.3.14 Variance of EMG [38]

This feature uses the power of sEMG. Previous studies have defined the Variance of EMG (VAR) differently. Generally, the variance is the mean value of the square of the deviation of that variable. However, mean of EMG signal is close to zero. In consequence, variance of EMG can be calculated by:

$$VAR = \frac{1}{N-1}\sum_{i=1}^{N} {X_i}^2, \tag{13}$$

3.3.15 Willison Amplitude [38]

Willison Amplitude (WAMP) detects the number of times the difference between two adjacent segments exceeds a predefined threshold. “WAMP is related to the firing of motor unit action potentials (MUAP) and the muscle contraction level” [38]. WAMP is mathematically defined by (14).

$$WAMP = \sum_{i=1}^{N-1} f(|X_i - X_{i+1}|), \tag{14}$$

where, $f(x) = \begin{cases} 1, & if\ X \geq Threshold \\ 0, & otherewise \end{cases}$

typically, the value of the threshold is chosen between 10 and 100 mV [38]. WAMP is reported in studies related to sEMG classification [38].

3.3.16 Waveform Length [38]

This feature measures the length of the waveform over a time segment. It is reported in previous research and showed significance in sEMG analysis [38]. the waveform length (WL) is described as follows:

$$WL = \sum_{i=1}^{N-1} f(|X_{i+1} - X_i|). \tag{15}$$

3.3.17 Zero Crossing [47]

Zero Crossings (ZC) is a measure of the number of times an sEMG signal intercepts with the zero line. It is a very common feature that has been used in a number of studies related to sEMG. ZC is described by (16).

$$WL = \sum_{i=1}^{N-1} sgn(-X_i X_{i+1}), \tag{16}$$

where $sgn(x) = \begin{cases} 1, & if\ X > 0 \\ 0, & otherewise \end{cases}$

3.4 Artificial Neural Network Training

After extracting the seventeen features described above, they are fed to an ANN for training. For each sEMG channel, seventeen features are computed. This will result, for seven channels, in a total of 119 inputs to the ANN. Dual Layer Feedforward (DLF) ANNs are used to achieve the results of this study. Levenberg-Marquardt (LM) backpropagation, Bayesian Regularization (BR) backpropagation, and Scaled Conjugate Gradient (SCG) backpropagation are the training methods used in this research.

When the input and output are known to the ANN during the training process, the method is called Supervised Training (ST) [4]. The concept of ST is to develop an ANN that can realize the model of a system by mapping a given input set to its corresponding output set. The mapping process is achieved by adjusting the weights and biases of the layers of neurons [4]. The different methods of adjusting the weights and biases are called training algorithms. The detailed discussion and derivation of the three training algorithms, mentioned above, are explained in chapter one (Introduction).

Using these algorithms three different DLFs are trained. For each participant and for each data set collected from each experiment, three ANNs are developed and trained. The input to the neural networks, as clarified earlier, is the seventeen extracted features for each of the seven channels (total of 119 vectors) conjoined in one matrix of size: $nObservations \times 119$.

The structure of the artificial neural network is shown in Figure 3.4.1. As discussed earlier, 119 input vectors are fed to the ANN. Twenty neurons are used inside the hidden layer, while the output layer contained only one neuron.

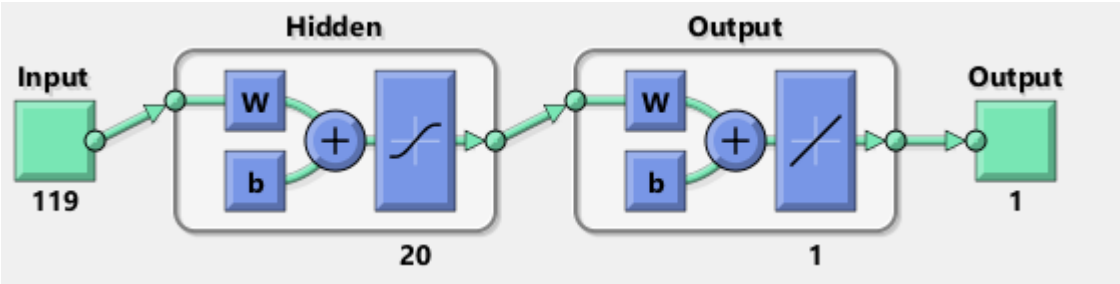


Figure 3.4.1: The neural network structure. The input, from seventeen features for seven channels, is fed using 119 nodes. The hidden layer is built with 20 neurons, and the output layer contains only one neuron.

3.5 Experiment setup

Twenty testing subjects participated in the research. The participants were asked to sit in a comfortable position, and make sure their arm muscles are not stressed. The participant is then asked to identify their dominant arm. The electrodes are then placed in random positions around their forearm (front and backside). Figure 3.5.1 and Figure 3.5.2 show the placement of electrodes around a participant forearm.

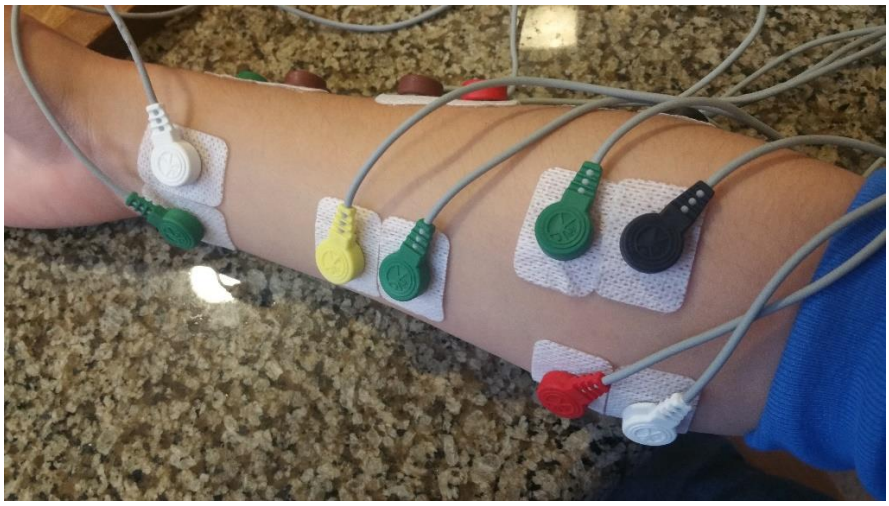


Figure 3.5.1:The placement of electrodes on the flexor side

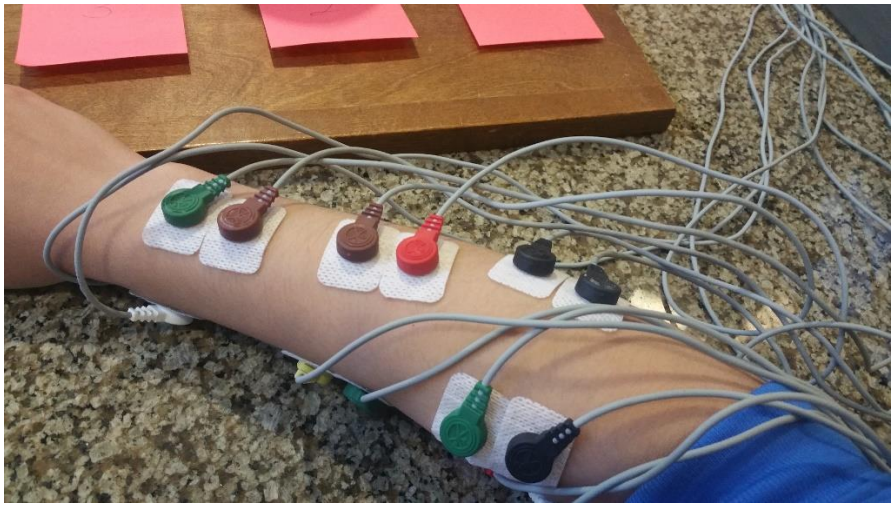


Figure 3.5.2:The placement of electrodes on the extensor side

The participant is then educated about five finger motions and asked to practice doing them as the operator randomly mention the numbers related to the motions (one to five). Figure 3.5.3 illustrates the five finger motions targeted in this research. Five objects are used for the motions to be performed on: three stress spheres and a disk. In front of each object, a number is printed indicating the label of the motion.

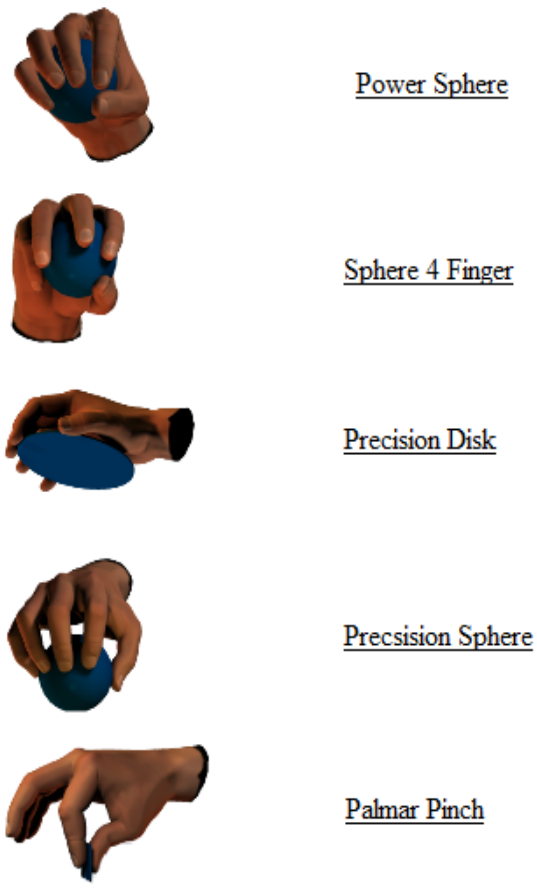


Figure 3.5.3: The five targeted finger motions

The collection of data starts by running the SIMULINK™ model (shown in the appendix). In there, the motions are coded to occur in a random manner. As this takes place, the numbers are displayed to the operator who, in turn, recites it to the participant to perform. Each experiment is designed to run for 30 seconds and each motion (presented as a number) is programed to appear at least twice. The procedure of the experiment is triplicated.

Chapter 4: Results and Discussion

4.1 Results

The final stage of the process, is data analysis. The collected data are analyzed based on two main criteria: performance and regression. For each of the twenty participants, the data set, from seven channels, is split into three subsets: training, validation and testing. Next, the split data are used to train three different neural networks, each applying a different learning algorithm.

Each experiment, through which the sEMG data are collected, is triplicated. Therefore, for each of the twenty test subjects, three data sets are recorded. For each of the three data sets, three neural networks are trained. For each trained ANN, the data is split into three sub-sets (training, validation, and testing). For each of the sub-sets, the performance and regression is calculated. In another word, the results presented in this section are based on the analysis of 1,080 individual values (performance and regression).

4.1.1 Performance

The difference (error) between predicted and target outputs occurs due to randomness or not accounting for information [48]. One way to quantify this difference is by computing the Mean Squared Error (MSE). MSE measures the average of the square of errors or deviations (the difference between the predicted and target values). It is the risk function by which the quadratic loss is quantified. The performance of a neural network training algorithm is inversely proportional to MSE (i.e. the lower the value of MSE, the better the performance). MSE is defined as:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(\widehat{\boldsymbol{Y}}_n - \boldsymbol{Y}_n)^2 \tag{68}$$

For each training algorithm, the performance is determined by calculating the average MSE. Table I shows the average performance for the training, validation and testing sets. In this contest the BR algorithm scored best with an average performance of 0.0045 for the training sets, 0.0125 for the validation sets, and 0.0576 for the testing sets. In competition with that, the LM algorithm comes second with a performance of 0.0146, 0.0547, and 0.0690 for the same three observations sets respectively. Finally, the SCG algorithm comes last with an average performance of 0.1248, 0.1425, and 0.9641 for the training, validation, and testing sets, respectively. These values, as shown in Table I, are averaged to further illustrate the performance of the three algorithms. LM, BR, and SCG has an average performance of 0.0461, 0.0248, and 0.1398, respectively. With no significant difference, BR algorithm performed slightly better than the other training algorithms.

Table I: Comparison of the training algorithms in terms of performance. The performance is computed according to (68).

Algorithm / Data Set	**Training**	**Validation**	**Testing**	**Average**
Levenberg-Marquardt	0.0146	0.0547	0.0690	0.0461
Bayesian Regularization	0.0045	0.0125	0.0576	0.0248
Scaled Conjugate	0.1248	0.1425	0.1521	0.1398

Throughout each learning process, the performance (MSE) of the data is traced. Figure 4.1.1 shows the performance plot for the training, validation, and testing data sets for one of the participants. In every training process, the error (MSE), as expected, drops for all the sets. At one point, the validation and testing MSE start to increase (i.e. performance drops). When this scenario takes place, it indicates the occurrence of overfitting. Even though the error for the training data set might still be decreasing, the learning process stops when the validation and testing performance exceeds a threshold. The same case is demonstrated in Figure 4.1.1: the validation and testing error started to deviate upward (worsen) at about the 5th epoch. The training continued giving the error the chance to drop again. However, when it didn't, the training was terminated.

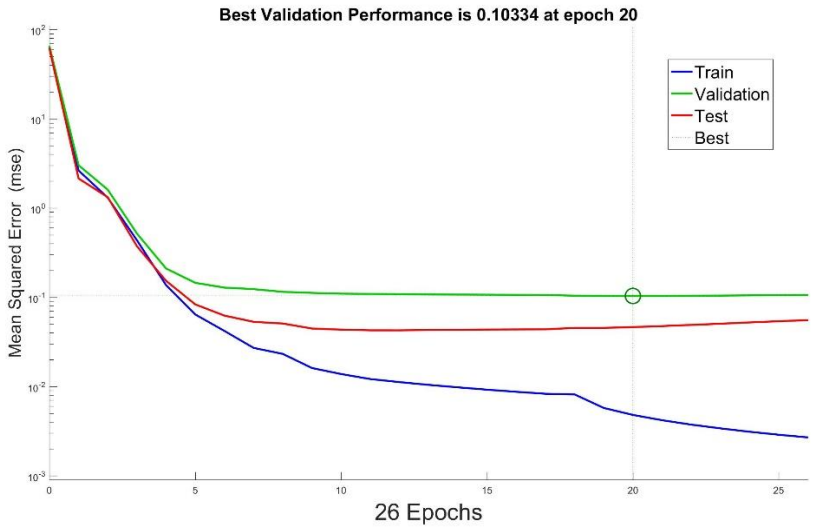


Figure 4.1.1:Performance plot for a testing subject. The MSE is traced as the weights of an LMNN are updated. The training, validation, and testing data sets are presented in blue, green, and red, respectively. The dotted line and the circle indicates the epoch at which the best performance is identified. In this case it is the 20th epoch.

4.1.2 Correlation

The measure of dependence or association of any two random variables (two data sets) is, in statistics, called correlation [49]. A very common example is the correlation

(dependence or association) between the demand and the price of a certain product. Correlation is represented by a number that indicates the strength of the relationship between a pair of random variables. There are many types of correlation coefficients. Some of which are linear and others are nonlinear (more robust). The most common is the Pearson product-moment correlation coefficient, which is a measure of the linear correlation between a pair of random variables.

For this study, the Pearson correlation coefficient is used to analyze the data. After each learning process, and for each test subject, the correlation is measured between the predicted and the target outputs. A value between -1 and +1 indicates the association between the predicted and target data. A value of '-1' represents a strictly negative correlation (i.e. as one variable increases, the other decreases). If a value of '0' is returned, the two outputs are not correlated. A value of '+1' indicates a strict positive correlation (i.e. both variables increase and decrease together). These values are attained by [50]:

$$\rho(X,Y) = \frac{1}{N-1}\frac{\sum_{i=1}^{N}(X_i-\mu_X)(Y_i-\mu_Y)}{\sigma_X\sigma_Y}, \tag{69}$$

where μ_X and σ_X are, respectively, the mean and standard deviation of the predicted output, X, μ_Y and σ_Y are the mean and standard deviation of the target output, Y.

Alternatively, (69) can be defined in terms of the covariance of X and Y [50]:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X\sigma_Y}, \tag{70}$$

where $cov(X,Y) = \sum_{i=1}^{N}(X_i-\mu_X)(Y_i-\mu_Y)/N-1$

Table II illustrates average correlation values calculated from each neural network. Similar to the performance, the correlation value is calculated for each of the data sets (training, validation, and testing) in each individual learnt ANN. Furthermore, these values are averaged to present the overall finding. Similar to their performance, the learnt artificial neural networks produced a predicted output that is highly linearly positively correlated to the target outputs. This result is expected as their performance values demonstrated high accuracy. The BR algorithm has an average correlation of 0.9990, 0.9975, and 0.9871 for the training, validation, and the testing data sets, respectively. The LM algorithm scored 0.9968, 0.9880, and 0.9846 for the same order of the data sets. Finally, the SCG-trained ANNs produced an average correlation of 0.9716, 0.9669, and 0.9641 for the training, validation, and testing data sets, respectively. These data are further average to result in 0.9945 correlation for the BR algorithm, 0.9898 for the LM algorithm, and 0.9675 for the SCG algorithm. The data demonstrates a significance linear correlation between the predicted and target outputs. For a little difference, the BR algorithm, again, has shown a better accuracy in estimating the target outputs.

Table II: Average correlations for the trained neural networks calculated according to (69)

Algorithm / Data Set	**Training**	**Validation**	**Testing**	**Average**
Levenberg-Marquardt	0.9968	0.9880	0.9846	0.9898
Bayesian Regularization	0.9990	0.9975	0.9871	0.9945
Scaled Conjugate	0.9716	0.9669	0.9641	0.9675

Linear regression, similar to the linear correlation coefficient, is a method of modeling the relationship between two or more random variables. This approach, however, is graphical. This approach demonstrates a visual approach into identifying the correspondence of one data set to another. In the case of this research, the targeted question is, how close are the predicted and target outputs? Figure 4.1.2 illustrates the linear regression of a trained ANN. For each of three data sets, a linear regression model is plotted. The figure shows a close linear relationship between the predicted and targeted outputs, which further proof the significance of the attained results.

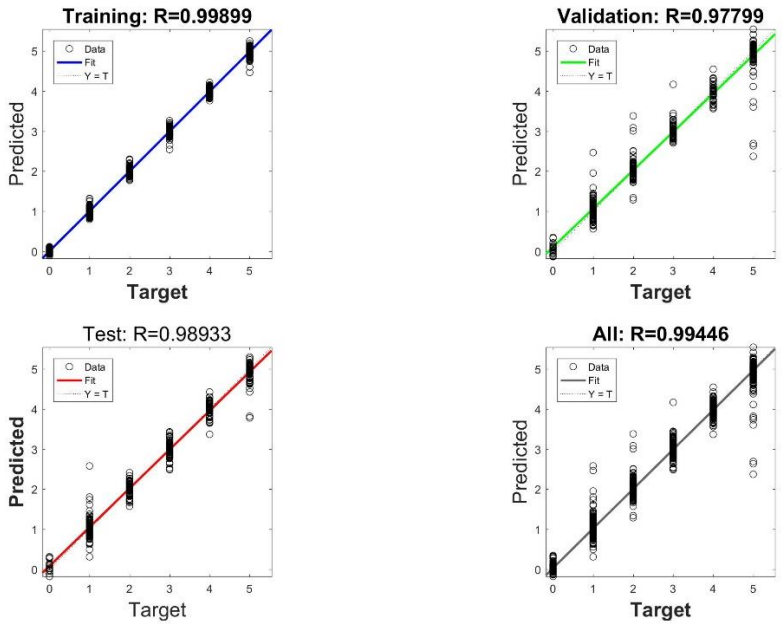


Figure 4.1.2: The regression line of a trained ANN for one of the test subjects. The predicted output is plotted against the target output for each of the data sets. In the bottom right corner of the graph, a linear regression model is created for the entire data.

4.2 Discussion

4.2.1 Ability to identify the motion

The outcome of this study has shown significance. As discussed in the results section, performance and correlation analysis have shown that a trained neural network,

with the seventeen features, is able to identify an intended motion with high precision. In fact, all trained neural networks succeeded in identifying the intended motion at all times. In another word, the ability of a trained neural network to identify an intended motion, for each test subject, is a 100%. Figure 4.2.1 shows the predicted output, for a trained neural network, plotted against the target output for the same ANN. It is obvious that the trained neural network is able to identify the intended motion. The predicted output of the neural network (orange line) is, very precisely, following the target output (blue line). The small error that shows up in the results is the cause of the oscillation of the ANN's output at steady points. This, however, does not affect the ability of the network to identify the intended motion. Figure 4.2.2 shows the plot for the error function for the same train artificial neural network. In both of the figures below, the horizontal axis represents the samples' number (i.e. first sample, second sample… etc.), while the vertical axis represents the value of the function (i.e. motion in Figure 4.2.1 and noise level in Figure 4.2.2).

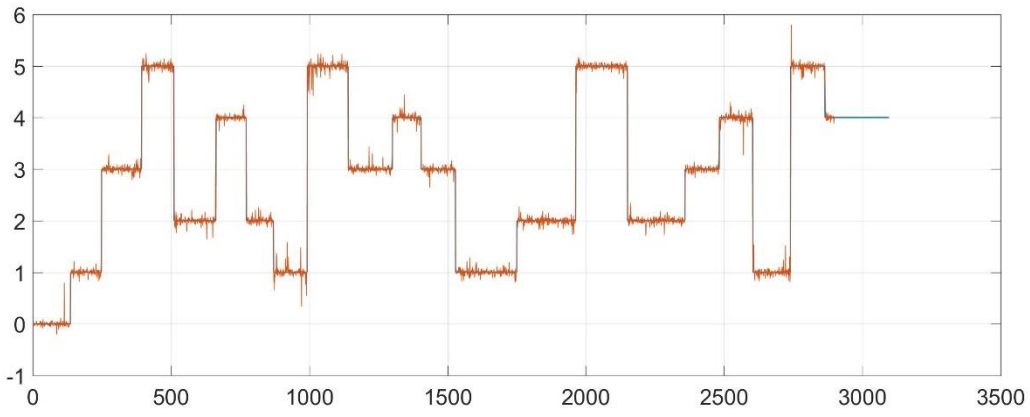


Figure 4.2.1: the predicted output for one of the trained ANNs plotted against the target (true) output. The orange function is the predicted output, while the blue function is the target output.

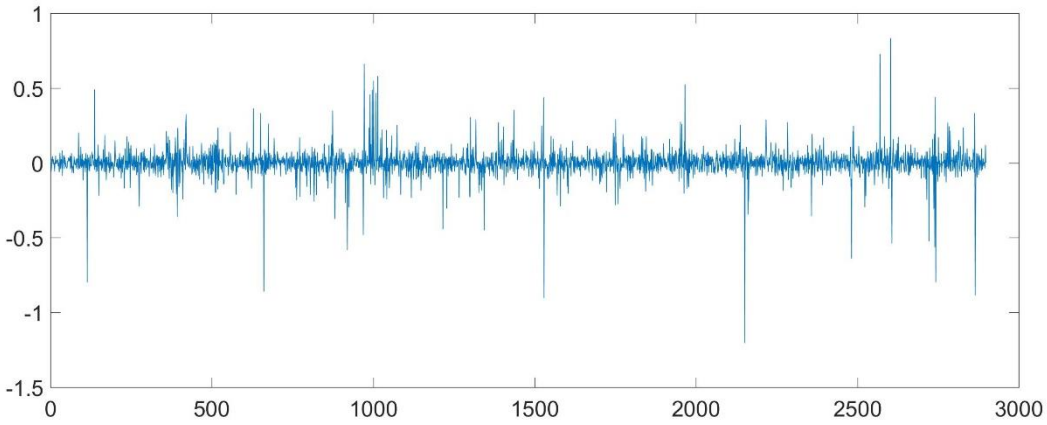


Figure 4.2.2:The error function for the same neural network in Figure 4.2.1.

4.2.2 Robustness

In addition to the neural network's ability to identify the intended motion, the proposed approach has shown its resistance to noise. The large number of features provides the neural network with enough information with which the neural network is always able to identify the corresponding motion during the data collection stage, despite the finding that the amount of noise within every participant's sEMG signal differed. Some sEMG signals contained more noise than the other. For example, an sEMG signal, such as the one shown in Figure 4.2.3 contains a large amount of noise compared to the one shown in Figure 4.2.4 The trained neural networks, using the two signals, however, succeeded in identifying the intended motion. An ANN is trained, using the signal in Figure 4.2.3. The performance for this particular ANN is 0.0457, and the regression is 0.9903. On the other hand, the sEMG signal in Figure 4.2.4 produced an ANN with a performance of 0.0374 and a regression of 0.9921. The training algorithm for the results shown in both figures are the same (LM). The predicted Vs. target outputs produced by the neural network trained with the sEMG in Figure 4.2.3, is shown in Figure 4.2.5. Follows that, in Figure 4.8, the predicted vs. target outputs produced by the ANN trained

using the sEMG signal in Figure 4.2.4. from the performance and regression results, as well as Figure 4.2.5 and Figure 4.2.6, it is clear that the amount of noise does not have much of an effect on the results which prove the robustness of the proposed approach. The same result holds for all data and ANNs.

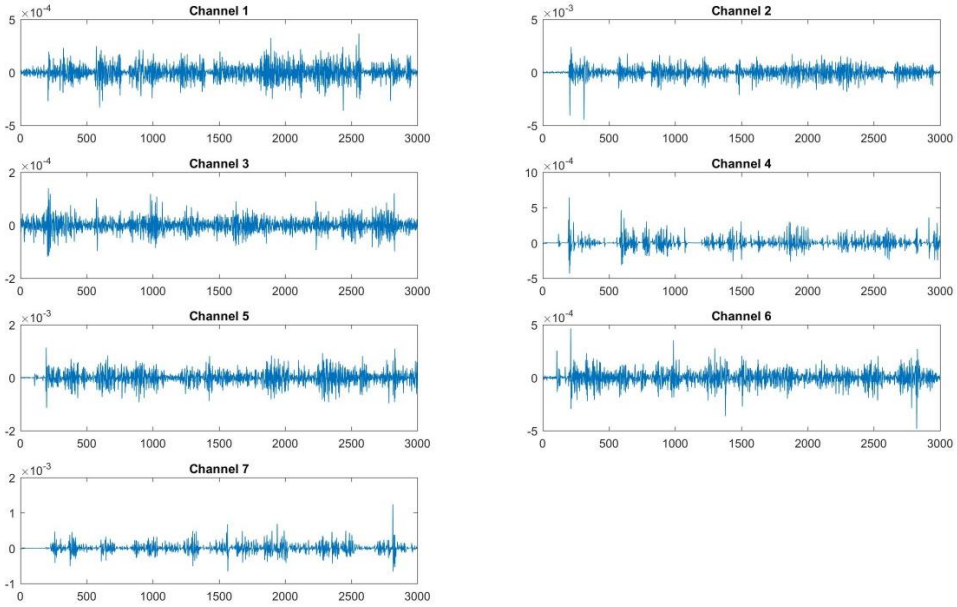


Figure 4.2.3: A noisy sEMG signal of a testing subject

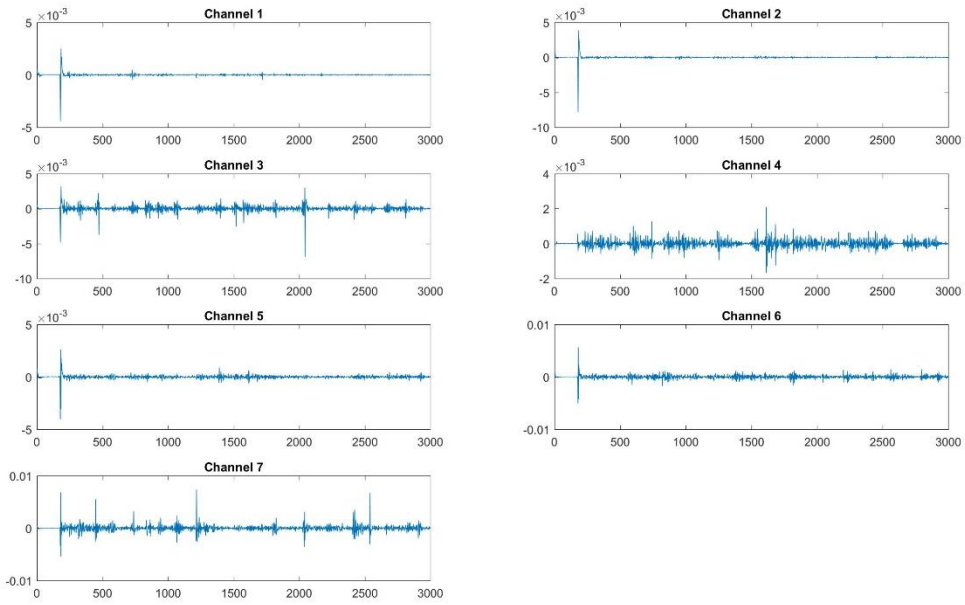


Figure 4.2.4: An sEMG signal of a testing subject that does not contain a large amount of noise

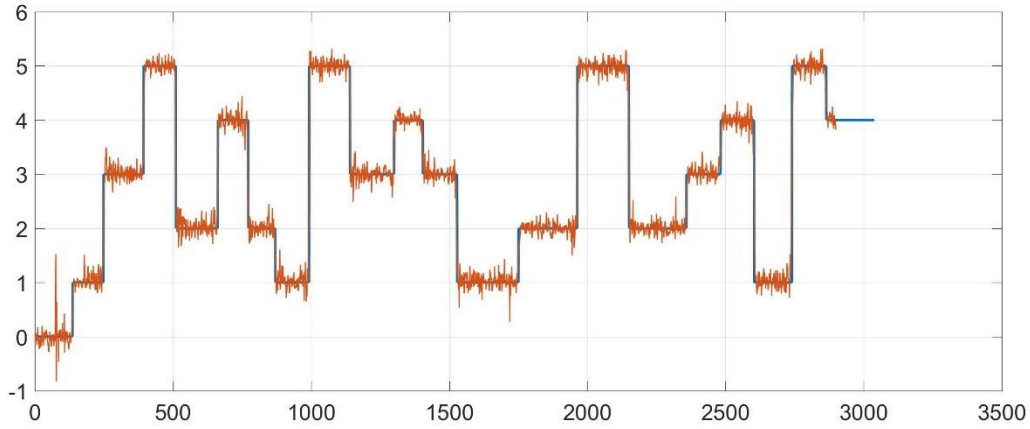


Figure 4.2.5: The predicted Vs. target outputs graph using the sEMG in Figure 4.2.3. The orange line represents the predicted output, while the blue line presents the target output. The training algorithm for the ANN is LM backpropagation.

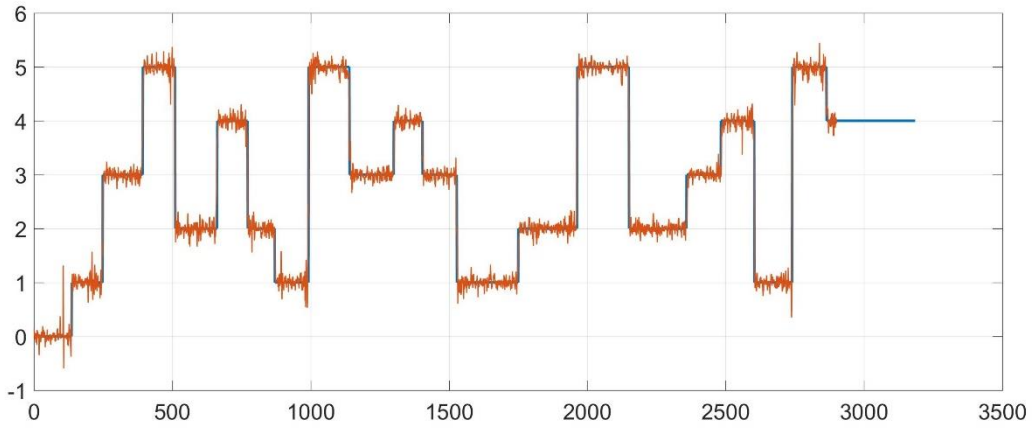


Figure 4.2.6: The predicted Vs. target outputs graph using the sEMG in Figure 4.2.4. The orange line represents the predicted output, while the blue line presents the target output. The training algorithm for the ANN is LM backpropagation.

It is reported in previous research that, in order to record a clear signal, the electrode locations on body parts had to be shaved (refer to the literature review section). In addition, some research placed their electrodes in body locations, where the motor points are more likely to exist (refer to the literature review section). These things were not considered in this research. The electrodes are placed in random locations on the forearm of the participants. In addition, no shaving or cleaning is done. Being able to predict the intended motion with such high accuracy, without the need for such arrangements, is another evidence for the robustness of this approach.

Another aspect of robustness, is the exclusion of kinematical variables. As described in the literature review, most research approached the problem of identifying sEMG signals by including information about the kinematics of the fingers (motional information about the fingers). Including such information helps the learning process for the artificial neural networks. The rule of thumb is that the more information an ANN is provided, the easier for the neural network to relate an input to a target output. In this research, the target outputs provided to the neural networks are discrete values

corresponding to each motion. Figure 4.2.7shows a plot of the target output signal. It is noticeable that the plotted function only takes integer values ranging from zero to five. These values correspond to the five motions explained earlier in the thesis. The results of this work are achieved without providing kinematical information to the neural networks, which is another evidence to the robustness of this work.

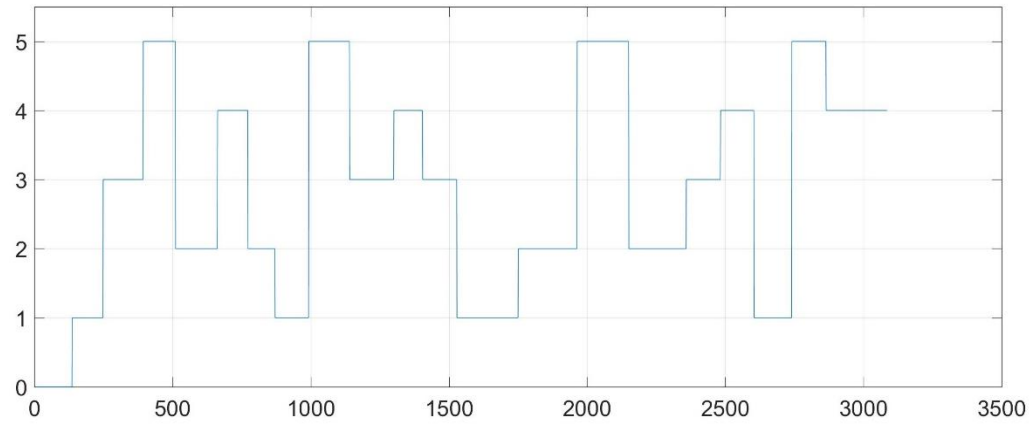


Figure 4.2.7: The target output signal. The signal takes values between zero and five corresponding to the five predefined motions. A value of zero indicates a no motion state.

The final aspect of robustness, is the frequency. Artificial neural networks can be thought as their biological counterparts: they have the ability to learn any type of tasks. Sometimes, they can pick up features that do not belong to the system [7], [4]. One of the features that is very likely to be picked up is the frequency. If some action is repeating in a periodic manner, then the neural network can, very likely, pick up the period and repeat the process in a similar pattern. In this work, this issue is taken into consideration and the experiments are designed so the behavior of the motion is random. Meaning that the time and order of targeted motions is randomly picked. Figure 4.2.7 further illustrates this concept. It is seen that the behavior of the motions function for the total time of the

experiment is random and not periodic. The ability of the ANNs to classify the sEMG signal in such conditions is an evidence of the robustness of the proposed approach.

4.3 Shortcomings

This research, just like any other, has its own shortcomings. Explaining them is beneficial in the process of preparing for the next step. In this section, the shortcomings of this approach are explained. First, the limitations of each training algorithm are highlighted. Next, the problem of generalization is addressed. Finally, complexity of the approach is highlighted.

The LM algorithm, as explained earlier in this thesis, avoids the computation of the Hessian matrix. Instead, the algorithm uses the Jacobian approximation to solve second order problems. The problem with the LM algorithm is that, because of its heavy computation, consumes a large amount of memory. The algorithm also does not use any of the generalization techniques to solve the problem of overfitting. Finally, because this method uses the calculation of Jacobian matrix, it assumes that the performance function is either the sum of squared errors or the mean of squared errors. Therefore, training an ANN with this algorithm requires the use of either of the mentioned performance functions [51].

The Bayesian Regularization (BR) algorithm, on the other hand, is derived under the claim that it generalizes better and tackles the problem of overfitting. The method of regularization, however, does not avoid the use of the Hessian matrix. Therefore, using this method requires a lot of time. During the process of this research, the BR algorithm took the longest. In fact, the desired minimization performance was never attained in any of the learning methods. The training is manually stopped after the performance of the

testing data levels down. In most cases, this happens around the 100th epoch. One of the shortcomings for using this algorithm, is the training was never allowed to continue until it stops.

The SCG algorithm, as discussed earlier, search the conjugate space and still would lead to quadratic termination. This method requires the least memory. During the process of neural networks learning, SCG reported as the fastest training algorithm among all. It is, however, as shown in the results section, the least accurate.

The issue of generalization is looked at in this research as well. When a neural network is trained, it is expected to perform similarly (produce precise outputs) at any other time, at least for the same testing subject. None of the previous research has demonstrated a discussion of this problem. In fact, many studies reported success in classification, using one data set divided into sub-sets (similar to the method of this research). The discussion here, however, takes it a step further. Recall that the experiment is triplicated for each of the participants. This allows, first, the verification of the success of the training methods. More importantly, it allows the testing of a trained ANN with totally different data. When a neural network is trained using a sub-division of a data set (the training subset), it is able to identify the rest of the samples (the validation and testing subsets). Once, however, a totally new data set is introduced (e.g. the data set of the second experiment), the ANN fails to provide a clear identification. A number of tests are developed to further inspect this issue. The findings states that: as long as a certain data set is part of the population used for training, the artificial neural network is able to identify and subset.

This research introduces precise identification, but, with that, it introduces complexity to the solution. The integration of seventeen features into the classification problem introduces a high computational cost. Especially when some of these features requires frequency domain analysis, which, in turn, requires heavy computing.

4.4 Recommendations

Although the findings of this research are of significance, there is still room for further improvement. This research can be both refined and taken to better levels. In this section, the recommendation to such achievements are laid.

Perhaps the most important aspect of this research is generalizing trained neural networks to new datasets taken at any time. One particular reason for which the generalization failed, could be the large number of features. Having many classifiers can lead to the extraction of unnecessary features, which, in turn, can mislead the neural network. The answer could be in a different feature, or may be a different neural network training method. All these are questions that need to be answered.

Many of the parameters in this work are randomly picked. Optimizing them is expected to enhance the results in a dramatic way. These parameters are listed below:

- The window size.
- The combination of features by which the best result is achieved.
- The sliding step.
- The thresholds for some of the features.

It is believed that altering these values can be of an effect to the better of this work. Optimizing the window size can help improve the classification problem and, in case of an optimal value less than a hundred, help keeping more samples. It is discussed that the large number of features can be problematic. Finding the optimal combination of features, by which the best results is achieved, can help this problem in two ways: reduce the complexity of the design and better the results. The sliding step of the window in each feature has the potential, if optimized, to help improve the results. Some of the features discussed in this work has the threshold option as part of their definition. Finding the optimal threshold can be of great benefit to classification. For example, if the zero crossing feature is designed to have a threshold crossing instead of the value zero, the noise in the sEMG signal won't be of a problem.

Chapter 5: Conclusion

The study of human cognition through artificial intelligence has recently surfaced. The rapid improvement of soft computing abilities has allowed the implementation of many concepts including the artificial neural networks. In parallel, the study of sEMG signals and their relation to human cognition has matured to become one of the hot topics in research nowadays.

sEMG signal relation to human motion has been intensively studied. Many researches explored this relation through the application of artificial neural networks. Most of these studies included the use of features to help the learning process of the neural networks. In addition to the inclusion of feature extraction (teacher inputs to ANNs), most literature provided kinematical information at the other end of the process (teacher outputs/target outputs). Usually, a small number of features (ranging from two to four), combined with only one training algorithm for neural networks, is used in the studies.

This research explored the relationship between sEMG and the human finger motion. Seven channels, located at points on the participants' forearm, recorded the data from twenty testing subjects. Unlike previous research, seventeen features are extracted from the sEMG data to provide an intense amount of information. Some of these features are used for the first time in this research. Next, three different neural networks are trained and analyzed against each other. The LM, BR, and SCG algorithms are the corner stones by which the three algorithms are distinguished.

The outcomes of the work have shown significance. The testing results have shown precise ability of the trained neural networks to classify the intended motion. The performance (the lower the better) of the LM, BR, and SCG algorithms are: 0.0461, 0.0310, and 0.1398 respectively. The correlation values (closer to 1 is best) between the predicted and target outputs are: 0.9898, 0.99305, and 0.96753 for the same order of training algorithm.

Finally, this research can be taken to many directions. An optimization procedure is believed to highly positively affect the research and the numbers. The complexity of the research also need to be reduced by finding the combination of features by which the best results are achieved.

Bibliography

[1] S. Sanei and H. Hassani, Singular Spectrum Analysis of Biomedical Signals, Boca Raton: Taylor & Francis Group, 2016.

[2] Peter Konrad , The ABC of EMG: A Practical Introduction to Kinesiological Electromyography, Scottsdale: Noraxon U.S.A, Inc. , 2006.

[3] J. M. W. J. K. S. Lyn D Weiss, Easy EMG: A Guide to Performing Nerve Conduction Studies and Electromyography, Amsterdam: Elsevier Inc., 2016.

[4] M. T. Hagan, H. B. Demuth and M. H. Be, Neural Network Design, Boston: PWS pub, 1996.

[5] H. Jaeger, "A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach," Fraunhofer Institute for Autonomous Intelligent Systems , Bermen, 2002.

[6] D. A. Winter, Biomechanics and Motor Control of Human Movement, New Jersey: Wiely & Sons Inc., 2009.

[7] S. Kumar and A. Mital, Electromyography In Ergonomics, London: Taylor & Francis Inc., 1996.

[8] C. R. Jeffrey and D. Maya, "The History of Muscle Dysfunction and SEMG," *Journal of Electromyography and Kinesiology,* vol. 4, pp. 5-14, 1994.

[9] Noraxon, "Manufacturers of professional electromyography products (emg/semg) and biomechanical sensors," 10 March 2006. [Online]. Available: ww.noraxon.com . [Accessed 31 May 2016].

[10] Mathworks, "MATLAB and Simulink for Technical Computing," 15 September 2005. [Online]. Available: ww.mathworks.com . [Accessed 31 May 2016].

[11] Noraxon, "Compare EMG systems," [Online]. Available: http://www.noraxon.com/products/emg-electromyography/. [Accessed 31 May 2016].

[12] Mathworks, "Signal processing toolbox," [Online]. Available: http://www.mathworks.com/products/signal/. [Accessed 31 May 2016].

[13] T. Z. H. M. N. M.H.Jali, "Pattern Recognition of EMG Signal During Load Lifting Using Artificial Neural Network (ANN)," in *2015 IEEE International Conference on Control System, Computing and Engineering.*, Penang, 2015.

[14] Z. G. H. N. K. G. B. B. L. V. a. H. L. Peng, "Feasibility of NeuCube spiking neural network architecture for EMG pattern recognition," in *International Conference on Advanced Mechatronic Systems (ICAMechS)*, Beijing, 2015.

[15] Z. G. H. a. W. W. L. Peng, "A dynamic EMG-torque model of elbow based on neural networks," in *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Milan, 2015.

[16] G. L. X. Z. J. H. a. G. L. A. Xiong, "Feasibility of EMG-based ANN controller for a real-time virtual reality simulation," in *38th Annual Conference on IEEE Industrial Electronics Society*, Montreal, 2012.

[17] L. W. a. T. S. Buchanan, "Prediction of joint moments using a neural network model of muscle activations from EMG signals," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 10, no. 1, pp. 30-37, 2002.

[18] I. Batzianoulis , S. El-Khoury , S. Micera and A. Billard , "EMG-Based Analysis of the Upper Limb Motion," in *Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts* , New York, 2015.

[19] J.-S. R. Jang, C.-T. Sun and E. Mizutani, Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence, Upper Saddle River: pretice-Hall. Inc., 1997.

[20] Y.-S. Y. F. K. L. Y.-T. Z. a. P. A. P. F. H. Y. Chan, "Fuzzy EMG classification for prosthesis control," *IEEE Transactions on Rehabilitation* , vol. 1, no. 3, pp. 305-311, 2000.

[21] A. M. S. P. D. a. B. R. S. Micera, "A hybrid approach to EMG pattern analysis for classification of arm movements using sta- tistical and fuzzy techniques," *Med. Eng. Phys.* , vol. 21, pp. 303-311, 1999.

[22] A. B. Ajiboye and R. F. Weir , "A Heuristic Fuzzy Logic Approach to EMG Pattern," *IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING*, , vol. 13, no. 3, pp. 280-291, 2005.

[23] A. Phinyomark, C. Limsakul and P. Phukpattaranont , "A Novel Feature Extraction for Robust EMG Pattern Recognition," *JOURNAL OF COMPUTING* , vol. 1, no. 1, 2009.

[24] P. K. Yarlagadda, Real-time sEMG-based finger joint angle control for a smart prosthetic hand, Pocatello: Idaho State University, 2013.

[25] Mathworks, "Measure the Power of a Signal," [Online]. Available: http://www.mathworks.com/help/signal/ug/measure-the-power-of-a-signal.html. [Accessed 26 6 2016].

[26] N. Blaikie, Analyzing Quantative Data, London: SAGE publications Ltd, 2003.

[27] MATHWORKS, "meanfreq," 2016. [Online]. Available: http://www.mathworks.com/help/signal/ref/meanfreq.html. [Accessed 28 6 2016].

[28] W. H., B. P. Flannery, S. A. Teukolsky and W. T. and Vetterling, Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed, Cambridge : Cambridge University Press, 1992.

[29] Oxford University , A Dictionary of Physics, Oxford: Oxford University Press , 2009.

[30] E. L. M. a. R. M. E. A. Clancy, "Sampling, Noisereduction and Amplitude Estimation Issues in Surface Electromyography," *Journal of Electromyography and Kinesiology,*, vol. 12, no. 1, pp. 1-16, 2002.

[31] L. B. W. Frederick J Gravetter, Essentials of Statistics for the Behavioral Sciences 8th ed, Belmont: Wadsworth, 2014.

[32] G. L. X. Z. J. H. a. G. L. A. Xiong, "Feasibility of EMG-based ANN controller for a real-time virtual reality simulation," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Montreal, 2012.

[33] Wikipedia, the free encyclopedia, "Backpropagation," 2014. [Online]. [Accessed 5 7 2016].

[34] Y. D. Deyi Li, Artificial Intelligence with Uncertainty, Boca Raton: Taylor & Francis Group, LLC, 2007.

[35] H. Yu and B. M. Wilamowski, "Levenberg–Marquardt Training," in *Intelligent Systems*, Boca Raton, Taylor & Francis Group, 2011, pp. 12-1 to 12-15.

[36] Wikipedia, "Levenberg–Marquardt algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm. [Accessed 9 7 2016].

[37] K. Levenberg, "A Method for the Solution of Certain Nonlinear Problems in Least Squares," *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164-168 , 1944.

[38] D. W. Marquardt, "An Algorithm for Least Square Estimation of Non-Linear Parameters," *Journal of the Society for Industrial and Applied Mathematics,* vol. 11, no. 2, pp. 431-441, 1963.

[39] L. C. W. a. S. G. P. Dixon, "The global optimization problem: an introduction," in *Towards global optimization 2*, Amsterdam, 1978, pp. 1-16.

[40] S. a. B. D. Surjanovic, "Virtual Library of Simulation Experiments: Test Functions and Datasets," 1 2015. [Online]. Available: http://www.sfu.ca/~ssurjano. . [Accessed 2016 9 7].

[41] Mathworks, "trainlm," [Online]. Available: http://www.mathworks.com/help/nnet/ref/trainlm.html. [Accessed 10 7 2016].

[42] D. MacKay, "Bayesian Interpolation," *Neural Computation*, vol. 4, pp. 415-447, 1992.

[43] M. Hestenes, Conjugate Direction Methods in Optimization, New York: Springer Verlag, 1980.

[44] R. Fletcher, Practical Methods of Optimization, West Sussex: John Wiley & Sons, 1987.

[45] W. M. a. M. H. W. Philip E. Gill, Practical Optimization, Academic Press, 1980.

[46] P. W. a. M. Wright, "Chapters 8 and 9," in *Practical Optimization*, New York, Academia Press, 1981.

[47] L. E. Scales, "Chapters 8, 9, and 12," in *Introduction to Non-Linear Optimization*, New York, Springer-Verlag, 1985.

[48] E. L. Lehmann and G. Casella, Theory of Point Estimation (2nd ed.), New York: Springer, 1998.

[49] F. E. Croxton, D. J. Cowden and S. Klein, Applied General Statistics, Pitman, 1968.

[50] C. Zaiontz, "Real Statistics Using Excel: Basic Concepts of Correlation," [Online]. Available: http://www.real-statistics.com/correlation/basic-concepts-correlation/. [Accessed 20 7 2016].

[51] MATHWORKS, "detrend," [Online]. Available: http://www.mathworks.com/help/matlab/ref/detrend.html. [Accessed 20 7 2016].

[52] MATHWORKS, "subplot," [Online]. Available: http://www.mathworks.com/help/matlab/ref/subplot.html. [Accessed 20 7 2016].

[53] MATHWORKS, "plot," [Online]. Available: http://www.mathworks.com/help/matlab/ref/plot.html?searchHighlight=plot. [Accessed 20 7 2016].

[54] MATHWORKS, "title," [Online]. Available: http://www.mathworks.com/help/matlab/ref/title.html. [Accessed 20 7 2016].

[55] MATHWORKS, "function," [Online]. Available: http://www.mathworks.com/help/matlab/ref/function.html?searchHighlight=function. [Accessed 20 7 2016].

[56] MATHWORKS, "bandpower," [Online]. Available: http://www.mathworks.com/help/signal/ref/bandpower.html. [Accessed 21 7 2016].

[57] MATHWORKS, "sum," [Online]. Available: http://www.mathworks.com/help/matlab/ref/sum.html?searchHighlight=sum. [Accessed 21 7 2016].

[58] MATHWORKS, "mean," [Online]. Available: http://www.mathworks.com/help/matlab/ref/mean.html. [Accessed 21 7 2016].

[60] K. E. a. B. Hudgins, "A robust, real-time control scheme for multifunction myoelectric control," *IEEE Trans. Biomed. Eng.* , vol. 50, no. 7, pp. 848-854, 2003.

[61] Mathworks, "trainscg," [Online]. Available: http://www.mathworks.com/help/nnet/ref/trainscg.html. [Accessed 11 7 2016].

Appendix

MatlabTM Code

For this research, all the data collection, processing, and analysis were done with the help of MATLAB™. In this section, the code, by which the findings of this study are carried, is presented.

Preprocessing

When the data is collected, it is sent to MATLAB™ workspace. The format of these data is "structure with time." Therefore, the first step is to extract the signal values from the structural files. The structural data are defined using the channel number (e.g. Channel5). The output, in the other hand, is given the name "OutputTeacher." The process of extracting the values is done using the following Code:

```
Channel1= [Channel1.signals.values];

Channel2= [Channel2.signals.values];

Channel3= [Channel3.signals.values];

Channel5= [Channel5.signals.values];

Channel6= [Channel6.signals.values];

Channel9= [Channel9.signals.values];

Channel10= [Channel10.signals.values];

OutputTeacher1 = [OutputTeacher.signals.values];
```

For each experiment, the SIMULINK model is started and stopped manually. This takes time and, because of the delay, causes extra data to be collected. As discussed earlier in this thesis, the sampling time for collecting the data is 0.01. This means that, for every second, a 100 samples should be recorded. Every experiment is designed to run for 30 seconds. Therefore, the first 3000 samples are the ones of interest and everything after those samples are to be discarded. The following code is used to eliminate all observations after the 3000th:

```
V1 = 3001;
V2 = length(OutputTeacher.time);
Channel1(V1:V2,:)=[];
Channel2(V1:V2,:)=[];
Channel3(V1:V2,:)=[];
Channel5(V1:V2,:)=[];
Channel6(V1:V2,:)=[];
Channel9(V1:V2,:)=[];
Channel10(V1:V2,:)=[];
OutputTeacher1(V1:V2,:)=[];
```

After extracting the values collected and putting it in a single vector, all vectors are combined in one matrix. This matrix need to be of the format of “double.” The following code is used to achieve this:

```
InputTeacher1 =
[Channel1,Channel2,Channel3,Channel5,Channel6,Channel9,Channel1
0];

InputTeacher1 = im2double(InputTeacher1);

OutputTeacher1 = im2double(OutputTeacher1);
```

Next, the offset is removed using the command '*detrend'* from MATLAB™. This command "removes the mean value or linear trend from a vector or matrix" [52]. For a matrix, '*detrend'* removes the trend from every column, which is how the matrices are prepared using the previous code. The offset is removed from the "InputTeacher" matrix using the following code:

```
InputTeacher1 = detrend(InputTeacher1);
```

After the offset is removed, it is good to look at the raw sEMG signal collected from the seven channels before extracting the features. This step helps making sure everything looks fine and that all the previous processes are made before extracting the signal. The command '*subplot'* divides a figure and creates axes in tiled positions [53]. The command *'plot'* calls a predefined data set and creates a 2-D line figure using the values in the data set [54]. Finally, the command *'title'* labels (add a title) on the top-center of a specified axes [55]. The following code creates seven sub-plots in which the sEMG signal from the seven channels is demonstrated:

```
subplot(4,2,1)

plot(InputTeacher1(:,1))
```

```
title('Channel 1')
subplot(4,2,2)
plot(InputTeacher1(:,2))
title('Channel 2')
subplot(4,2,3)
plot(InputTeacher1(:,3))
title('Channel 3')
subplot(4,2,4)
plot(InputTeacher1(:,4))
title('Channel 4')
subplot(4,2,5)
plot(InputTeacher1(:,5))
title('Channel 5')
subplot(4,2,6)
plot(InputTeacher1(:,6))
title('Channel 6')
subplot(4,2,7)
plot(InputTeacher1(:,7))
title('Channel 7')
```

Feature Extraction

As discussed earlier in the thesis, seventeen features are developed to extract the dynamics from a given sEMG signal. All seventeen are programmed in MATLAB™ as functions and grouped in a toolbox. This makes it easier to call the feature at any time. The MATLAB™ command *'function'* declares (creates) a function, its inputs, and its outputs [56]. The input arguments for all the features are the variable at which the data is stored and the width of the sliding window. The data set could be a vector or a matrix. In case of a matrix, a feature is extracted from every column separately and the output is recorded in columns of a new matrix. The window in each feature, with a specified width (by the user), slides along the columns of the data set and calculates the feature of the sample within its edges. The window then slides one data point and repeats the same process. First, the code for each feature is presented. Later, the code, by which the features are called to be applied on the input teacher, is shown.

The first feature is the Band Power. The MATLAB™ command *'bandpower'* returns the average power in the input signal. If the signal is a matrix, then '*bandpower'* computes the average power in each column independently [57]. The code for this features is presented below:

```
function BandPower = BandPower(Data,WinSize)

[N,nChannels] = size(Data); %pulling out the dimentions

%% computation of the output

BandPower = nan(N-WinSize+1,nChannels); %initializing for faster
computation
```

```
nSteps = (N-WinSize+1);


for i = 1:nSteps
   for j = 1:nChannels
      BandPower(i,j) = bandpower(Data(i:i+WinSize-1,j));
   end
end
```

The feature IEMG is described by (3). the MATLAB™ command *'sum(A)'* "returns the sum of the elements of A along the first array dimension whose size does not equal 1" [58]. If A is a matrix, then the command returns the sum of observations in each column. This command allows the representation of (3). The code for the feature IEMG follows:

```
function IEMG = IEMG(Data,WinSize)
[N,nChannels] = size(Data); %pulling out the dimensions
%% computation of the new matrix
IEMG = zeros(N-WinSize+1,nChannels); %initializing for faster
computation
nSteps = (N-WinSize+1);
for i = 1:nSteps
   for j = 1:nChannels
```

```
        IEMG(i,j) = sum(abs(Data(i:i+WinSize-1,j)));
    end
end
```

Next, is the mean absolute deviation feature. This feature is presented by (7). The same Matlab command, *'sum,'* is used to program this feature as a function. The MATLAB™ code follows:

```
function MAD = MAD(Data,WinSize)
% The Mean Absolute Deviation (MAD) of a set of data
% is the average distance between each data value and the mean.
% This Feature work as a sliding window which calculates MAD as it sweeps
% along each column of a given data set. The formula that calculates MAD is
% given by: MAD = (1/N)*sum(abs(x - xBar), where N, x, and xBar, are the
% size, values, and the mean of a targeted sample in a population.
%% Input Arguments
% Data = Input Matrix to which the function is applied.
% WinSize = the size of the sliding window (integer).
%% Output Arguments
%  MAD = an N-WinSize matrix in which the MAD is computed.
```

```
%%

[nObservations,nChannels] = size(Data);       %pulling out the
dimensions

MAD = nan(nObservations-WinSize+1,nChannels); %initializing

nSteps = (nObservations-WinSize+1);   %Identifying the dimention of

                                      %the new Matrix.

for i = 1:nSteps

   for j = 1:nChannels

      MAD(i,j) = (1/WinSize)*sum(abs(Data(i:i+WinSize-1,j)- ...

               mean(Data(i:i+WinSize-1,j))));

   end

end
```

The mean absolute value of an sEMG is described by (4). The same command used with IEMG and MAD features is used to build the MAV feature. The code is copied below:

```
function MAVout = MAV(Data,WinSize)

% This function computes the mean absolute value (MAV) of the Data
within

% a sliding window of a specified size. Within each iterative window,

% MAV = (1/N)*sum(abs(x(i)), i = 1 .. N); where N = the size of the
window,
```

```
% x(i) = The observations within the window. The input is a matrix of
size

% = observations X variables

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).

%% Output Arguments

% MAV = an N+1-WinSize matrix in which the MAV is computed

[nObservations,nChannels] = size(Data);       %pulling out the
dimensions

MAVout = zeros(nObservations-WinSize+1,nChannels); %initializing

nSteps = (nObservations-WinSize+1);   %Identifying the dimension of

                                      %the new matrix.

for i = 1:nSteps

   for j = 1:nChannels

      MAVout(i,j) = (1/(WinSize))*sum(abs(Data(i:i+WinSize-1,j)));

   end

end
```

The "mean" feature is described in (8). The MATLAB™ command *'mean'* returns the mean of an array whose dimension is not one [59]. If the command is applied to a matrix, it returns the mean of each column. The code that defines this feature as a function is presented below:

```
function Mean = Mean(Data,WinSize)



% mean and expected value are used synonymously to refer to one
measure of

% the central tendency either of a probability distribution or of the

% random variable characterized by that distribution. This function is

% window that slides along the columns of a specified data set and
computes

% the mean.

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).

%% Output Arguments

%  Mean = an N-WinSize matrix in which the mean is computed.



%%

[nObservations,nChannels] = size(Data);       %pulling out the
dimensions

Mean = zeros(nObservations-WinSize+1,nChannels);%initializing for
faster

                              %computation

nSteps = (nObservations-WinSize+1);   %Identifying the dimensions of
```

```
                                %the new matrix.
for i = 1:nSteps
   for j = 1:nChannels
      Mean(i,j) = mean(Data(i:i+WinSize-1,j));
   end
end
```

The Mean frequency feature is programmed with the help of the MATLAB™ command, '*meanfreq.'* This command "estimates the mean normalized frequency of the power spectrum of a time-domain signal." [60]. The code to this feature follows:

```
function MeanFreq = MeanFreq(Data,WinSize)
[N,nChannels] = size(Data); %pulling out the dimentions
%% computation of the new matrix
MeanFreq = zeros(N-WinSize+1,nChannels); %initializing for faster
computation
nSteps = (N-WinSize+1);


for i = 1:nSteps
   for j = 1:nChannels
      MeanFreq(i,j) = meanfreq(Data(i:i+WinSize-1,j));
   end
```

```
end
```

Similar to the mean frequency, the MATLAB™ command, '*medfreq,'* "estimates the median normalized frequency of the power spectrum of a time-domain signal" [61]. The code by which this function is defined in MATLAB™ is below:

```
function MedFreq = MedFreq(Data,WinSize)

[N,nChannels] = size(Data); %pulling out the dimentions

%% computation of the new matrix

MedFreq = zeros(N-WinSize,nChannels); %initializing for faster
computation

nSteps = (N-WinSize+1);



for i = 1:nSteps

   for j = 1:nChannels

      MedFreq(i,j) = medfreq(Data(i:i+WinSize-1,j));

   end

end
```

The first modification of the mean absolute value, MMAV1, is described by (5). The second modification, MMAV2, is described by (6). Both of these features requires the use of the MATLAB™ command, *'sum,'* which is described in earlier features in this

section. The code by which these two features are integrated into MATLAB™ is presented below starting with the code for MMAV1 followed by MMAV2:

```
function MMAV1out = MMAV1(Data,WinSize)

% This function computes the modified mean absolute value 1(MMAV1) of the

% Data within a sliding window of a specified size. Within each iterative

% window, MMAV1 = (1/N)*sum(Wn*abs(x(i)), i = 1 .. N); where N = the size

% of the window, x(i) = The observations within the window. The input is a

% matrix of size = observations X variables. Wn is a weighting window

% function such as Wn = 1 if 0.25N <= n <= 0.75N, and Wn = 0.5 otherwise.

%% InputArguments

% Data = Input Matrix of the size Observations X variables to which the

% function is applied.

% WinSize = the size of the sliding window (integer).

%% Output Arguments

% MMAV1 = an N+1-WinSize matrix in which the MMAV1 is computed

%%
```

```
[nObservations,nChannels] = size(Data);       %pulling out the
dimentions

MMAV1out = zeros(nObservations-WinSize+1,nChannels);
%initializing

nSteps = (nObservations-WinSize+1);          %Identifying the dimention
of

w = zeros(WinSize,1);

for n =1:WinSize

   if (0.25*WinSize <= n && n <= 0.75*WinSize)

        w(n) = 1;

   else

        w(n) =  0.5;

   end

end

for i = 1:nSteps

   for j = 1:nChannels

      MMAV1out(i,j) = (1/(WinSize))*sum(w.*abs(Data(i:i+WinSize-
1,j)));

   end

end
```

```
function MMAV2out = MMAV2(Data,WinSize)
```

```
% This function computes the modified mean absolute value
2(MMAV2) of the

% Data within a sliding window of a specified size. Within each
iterative

% window, MMAV1 = (1/N)*sum(Wn*abs(x(i)), i = 1 .. N); where N =
the size

% of the window, x(i) = The observations within the window. The
input is a

% matrix of size = observations X variables. Wn is a weighting
window

% function such as Wn = 1 if 0.25N <= n <= 0.75N, Wn = 4n/N if n <
0.25N,

% and Wn = 4(n-N)/N if n > 0.75N

%% InputArguments

% Data = Input Matrix of the size Observations X variables to which
the

% function is applied.

% WinSize = the size of the sliding window (integer).

%% Output Arguments

% MMAV2 = an N+1-WinSize matrix in which the MMAV2 is
computed

%%

[nObservations,nChannels] = size(Data);       %pulling out the
dimentions
```

```
MMAV2out = zeros(nObservations-WinSize+1,nChannels);
%initializing

nSteps = (nObservations-WinSize+1);          %Identifying the dimention
of

w = zeros(WinSize,1);

for n =1:WinSize

   if (0.25*WinSize <= n && n <= 0.75*WinSize)

        w(n) = 1;

   elseif n < 0.25*WinSize

        w(n) =  4*n/WinSize;

   elseif n > 0.75*WinSize

        w(n) = (4*(n-WinSize))/WinSize;

   end

end

for i = 1:nSteps

   for j = 1:nChannels



      MMAV2out(i,j) = (1/(WinSize))*sum(w.*abs(Data(i:i+WinSize-
1,j)));

   end

end
```

The RMS feature is presented by (9). With the help of the MATLAB™ command, *'rms,'* this feature is built into the toolbox. The command, *'rms,'* computes the RMS value of a specified data set [62]. If the it is given a matrix, it operates along the columns of the matrix providing a row vector in which the calculation of the RMS value is presented as an element [62]. The code for this feature follows:

```
function RMSout = RMS(Data,WinSize)


% Root Mean Square (RMS) is modeled as amplitude modulated
Gaussian random

% process whose RMS is related to the constant force and non-
fatiguing

% contraction. It relates to standard deviation, which can be expressed
as:

% RMS = sqrt((1/nObservations)*sum(x(n)^2))

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).

% SlideStep = an integer representing the slide of the window at each
step.

%% Output Arguments

% MAV = an N-WinSize matrix in which the IAV is computed
```

```
%%

[nObservations,nChannels] = size(Data);       %pulling out the
dimentions

RMSout = zeros(nObservations-WinSize+1,nChannels); %initializing
the for faster

                                %computation

nSteps = (nObservations-WinSize+1);   %Identifying the dimention of

                                %the new matrix.

for i = 1:nSteps

   for j = 1:nChannels

      RMSout(i,j) = rms(Data(i:i+WinSize-1,j));

   end

end
```

The slope sign change feature (SSC), is computed by (10). To be able to integrate this function, another function, SC, is defined. After that, SC is used to program SSC. SC outputs one when the slop changes signs. The code for SC is copied first, followed by SSC:

```
function SCout = SC(Data)

% This function outputs a value of one, if the slope of a given data set

% changed signs (i.e. positive to negative and vice versa). The function
is
```

```
% used in the SSC feature extraction


%%

dx = diff(Data);

[nObservations,nChannels] = size(dx);       %pulling out the dimentions

SCout = zeros(nObservations+1,nChannels);

for i = 1:nObservations-1

   for j = 1:nChannels

      if  -(dx(i,j)*dx(i+1,j)) > 0

         SCout(i,j) = 1;

      else

         SCout(i,j) = 0;

      end

   end

end
```

```
function SSCout = SSC(Data,WinSize)

% Slope Sign Change (SSC) is another method to represent the
frequency

% information of sEMG signal. The number of changes between
positive and
```

```
% negative slope among three consecutive segments are performed
with the

% threshold function for avoiding the interference in sEMG signal. The

% calculation is defined as: SSC = sum(F((x(n)-x(n-1))*(x(n)-
x(n+1)))),

% where F(x) = 1, if x > threshold and = 0, if x < threshold.

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).

%% Output Arguments

%  SSC = an N-WinSize matrix in which the slope sign change is
computed.



%%

[N,nChannels] = size(Data); %pulling out the dimentions

%% computation of the new matrix

SSCout = zeros(N-WinSize+1,nChannels); %initializing for faster
computation

nSteps = (N-WinSize+1);

SignChange = SC(Data);



for i = 1:nSteps

   for j = 1:nChannels
```

```
        SSCout(i,j) = sum(SignChange(i:i+WinSize-1,j));
    end
end
```

The math of the SSI feature is provided by (11). This feature, similar to SSC, uses the command *'sum'* from MATLAB™, which is described earlier in this section. The code by which SSI is made into a function is:

```
function SSIout = SSI(Data,WinSize)


%Simple Square Integral (SSI) uses the energy of the sEMG signal as a
% feature. It can be expressed as: sum(abs(x(n))^2).
%% InputArguments
% Data = the previously computed MAV matrix
% WinSize = the size of the sliding window (integer).
% SlideStep = an integer representing the slide of the window at each
step.
%% Output Arguments
% SSI = an N-WinSize matrix in which the SSI is computed
[nObservations,nChannels] = size(Data);       %pulling out the
dimentions
SSIout = zeros(nObservations-WinSize+1,nChannels);%initializing for
faster
```

```
                                %computation
nSteps = (nObservations-WinSize+1);   %Identifying the dimention of
                                %the new matrix.


for i = 1:nSteps
   for j = 1:nChannels
      SSIout(i,j) = sum(power(abs(Data(i:i+WinSize-1,j)),2));
   end
end
```

The standard deviation of an array can be computed using the MATLAB™ function, *'std.'* This function is used to build the STD feature presented by (12). The code for this feature follows:

```
function Std = Std(Data,WinSize)

% The Standard Deviation is a measure of how spread out the
observations

% are. This function is window that slides along the columns of a
specified

% data set and computes the standard deviation.

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).
```

```
%% Output Arguments

%  Std = an N-WinSize matrix in which the standard deviation is
computed.



%%

[nObservations,nChannels] = size(Data);       %pulling out the
dimentions

Std = zeros(nObservations-WinSize+1,nChannels); %initializing for
faster

                                %computation

nSteps = (nObservations-WinSize+1);   %Identifying the dimention of

                                %the new matrix.

for i = 1:nSteps

   for j = 1:nChannels

      Std(i,j) = std(Data(i:i+WinSize-1,j));

   end

end
```

The variance of sEMG has a different computation formula than the regular variance. Therefore, the MATLAB™ command, *'var,'* cannot be used to build this feature [63]. VAR (the feature) is described by (13). the code for this feature is:

```
function Varout = Var(Data,WinSize)
```

```
% Variance of EMG (VAR) uses the power of the sEMG signal as a
feature.

% Generally, the variance is the mean value of the square of the
deviation

% of that variable. However, mean of EMG signal is close to zero. In

% consequence, variance of EMG can be calculated by:

% Var = (1/nObservations-1)*sum(x(n)^2)

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).

% SlideStep = an integer representing the slide of the window at each
step.

%% Output Arguments

%  Var = an N-WinSize matrix in which the variance is computed.



%%

[nObservations,nChannels] = size(Data);       %pulling out the
dimentions

Varout = zeros(nObservations-WinSize+1,nChannels); %initializing
for faster

                                %computation

nSteps = (nObservations-WinSize+1);   %Identifying the dimention of

                                %the new matrix.
```

```
for i = 1:nSteps

   for j = 1:nChannels

      Varout(i,j) = (1/(WinSize-1))*sum(power(Data(i:i+WinSize-
1,j),2));

   end

end
```

Next, is the Wilson amplitude feature. This feature is presented by (14). Once again, the command *'sum,'* which is explained earlier, is used integrate this feature into the toolbox. The code used for that follows:

```
function WAMPout = WAMP(Data,WinSize)

% Willison amplitude (WAMP) is the number of times that the
difference

% between sEMG signal amplitude among two adjacent segments
exceeds a

% predefined threshold. It is defined as: WAMP = sum(F(abs(x(n)-
x(n+1)))),

% where F(x) = 1 if x > threshold and = 0 otherwise.

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).
```

```
% SlideStep = an integer representing the slide of the window at each
step.

%% Output Arguments

%  WAMP = an N-WinSize matrix in which the Willison amplitude is
computed.



%%

[nObservations,nChannels] = size(Data);       %pulling out the
dimentions

WAMPout = nan(nObservations-WinSize+1,nChannels);
%initializing for faster

                                %computation

nSteps = (nObservations-WinSize);             %Identifying the dimention
of

                                %the new matrix

for i = 1:nSteps

   for j = 1:nChannels

      WAMPout(i,j) = sum(abs(Data(i:WinSize+i-1,j)-
Data(i+1:WinSize+i,j)));

   end

end
```

The waveform length feature is, as well, coded using *'sum.'* A mathematical describtion of the feature is provided by (15). The code for WL is provided below:

```
function WLout = WL(Data,WinSize)

% Waveform length (WL) is the cumulative length of the waveform over the

% time segment. WL is related to the waveform amplitude, frequency and time.

% It is given by: WL = sum(abs(x(n+1)-x(n)))

%% InputArguments

% Data = Input Matrix to which the function is applied.

% WinSize = the size of the sliding window (integer).

% SlideStep = an integer representing the slide of the window at each step.

%% Output Arguments

%  WL = an N-WinSize matrix in which the waveform length is computed.



%%


[nObservations,nChannels] = size(Data);       %pulling out the dimentions

WLout = zeros(nObservations-WinSize+1,nChannels);    %initializing for faster

                                %computation

nSteps = (nObservations-WinSize);            %Identifying the dimention of
```

```
                                %the new matrix
for i = 1:nSteps
   for j = 1:nChannels
      WLout(i,j) = sum(abs(Data(i+1:WinSize+i,j)-Data(i:WinSize+i-
1,j)));
   end
end
```

Finally, the zero crossing feature for which a mathematical description is provided by (16). The feature is built into two layers. The first layer is defining the signum function (also described in (16). The second layer is to use this function in defining the ZC feature. The code for the SGN function is provided first followed by the code for ZC:

```
function SGN = SGN(Data)
%% Signom function

[nObservations,nChannels] = size(Data);       %pulling out the
dimentions
SGN = zeros(nObservations,nChannels);  %initializing for faster
                                %computation
for i = 1:nObservations-1
   for j = 1:nChannels
      if  -(Data(i,j)*Data(i+1,j)) > 0
```

```
        SGN(i,j) = 1;
      else
        SGN(i,j) = 0;
      end
    end
end
```

```
function ZCout = ZC(Data,WinSize)
% This function produce an indication of when an EMG signal crosses
zero.
% it is window that slides along the columns of a specified data set and
% outputs one whenever the set of observations crosses zero.
%% InputArguments
% Data = Input Matrix to which the function is applied.
% WinSize = the size of the sliding window (integer).
%% Output Arguments
%  ZC = an N-WinSize matrix in which the zero crossing is computed.


%%
[N,nChannels] = size(Data); %pulling out the dimentions
```

```
%% computation of the new matrix

ZCout = zeros(N-WinSize+1,nChannels); %initializing for faster
computation

nSteps = (N-WinSize+1);

Crossing = SGN(Data);


for i = 1:nSteps

   for j = 1:nChannels

      ZCout(i,j) = sum(Crossing(i:i+WinSize-1,j));

   end

end
```

After all the features are defined, they can be simply called as long as MATLAB™ is on the same directory, where the files are located. After applying all the features, the resultant variables (output of each feature) are combined in one big matrix of the size: $nObservations \times 119$. After that, the output teacher vector size is adjusted to match the length of the input teacher matrix. The code for this process is provided below:

```
WinSize = 100;

BandPower1 = BandPower(InputTeacher1,WinSize);

IEMG1 = IEMG(InputTeacher1,WinSize);

MAD1 = MAD(InputTeacher1,WinSize);
```

```
MAV1 = MAV(InputTeacher1,WinSize);

Mean1 = Mean(InputTeacher1,WinSize);

MedFreq1 = MedFreq(InputTeacher1,WinSize);

MeanFreq1 = MeanFreq(InputTeacher1,WinSize);

MMAV11 = MMAV1(InputTeacher1,WinSize);

MMAV21 = MMAV2(InputTeacher1,WinSize);

RMS1 = RMS(InputTeacher1,WinSize);

SSC1 = SSC(InputTeacher1,WinSize);

SSI1 = SSI(InputTeacher1,WinSize);

Std1 = Std(InputTeacher1,WinSize);

Var1 = Var(InputTeacher1,WinSize);

WAMP1 = WAMP(InputTeacher1,WinSize);

WL1 = WL(InputTeacher1,WinSize);

ZC1 = ZC(InputTeacher1,WinSize);

N1 =
[BandPower1,IEMG1,MAD1,MAV1,Mean1,MedFreq1,MeanFreq1,M
MAV11,MMAV21,RMS1,SSC1,SSI1,Std1,Var1,WAMP1,WL1,ZC1];

O1 = OutputTeacher1(1:length(OutputTeacher1)-WinSize+1,:);
```

Neural Network training

Once the features are extracted and combined in a matrix, N1, of the same length as the output teacher matrix, O1, the data is ready to be fed to the neural network for training. For the learning task, the neural networks toolbox from MATLAB™ is used.

The first step of generating the network is to make sure that the input teacher and output teacher (targets) are defined in the work space. This should be the case, if the previous steps are done properly. The training starts by assigning the input and output to the variables ‘x’ and ‘t’ (for notation purposes) The matrices are also transposed to match the way the toolbox is coded (the observations are the columns and the variables are the rows). The code for this procedure is:

```
% This script assumes these variables are defined:
%   N1 - input data.
%   O1 - target data.
x = N1';
t = O1';
```

Next, the training algorithm is picked for the task. All three algorithms discussed in this work are programed into the neural networks toolbox of MATLAB™. The functions are built into the software and can be called at any time. The LM-algorithm is used for the sample code below, but the instructions on how to use a different algorithm is specified in the comments above.

```
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
```

```
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.
```

After choosing the learning algorithm, the network structure is built (number of neurons, hidden layers…etc.). The neural network structure is built using the function *'fitnet(hiddensize,trainFun).'* The function "returns a function fitting neural network with a hidden layer size of hiddenSizes and training function, specified by trainFcn" [64]. The code for this process is:

```
% Create a Fitting Network
hiddenLayerSize = 20;
net = fitnet(hiddenLayerSize,trainFcn);
```

It is always recommended for the input and output to go through certain adjustments [5], [65], [66]. In the neural network toolbox, this is called "processing functions." Two of the most important processing functions are used in this work: *'removeconstantrows'* and *'mapminmax.'* The first function, *'removeconstantrows,'* as the name suggests, removes the rows that contains constant (unchanging) values (e.g. a row of ones or zeros) [65]. The reason is that constant rows do not provide information and often are problematic in the training process [65]. The second processing function, *'mapminmax,'* scales the input and output to fit within the range of -1 to +1 [66]. The scaling of inputs and outputs is reported to be beneficial in many studies [4], [5], [66]. The code for setting up these functions is presented below:

```
% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};
```

Next, the input teacher and the output teacher data sets are divided into the three main data set: training, validation, and testing. The function *'dividerand,'* as the name suggests, divides a specified matrix to three sets using random indices [67]. For all the learnt ANNs, 70% of the data is assigned to the training data set and the rest of the data (30%) is shared equally by the testing and validation data sets. The code to this is provided below:

```
% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
```

As discussed earlier, the performance function used for the learning process is the mean squared error (MSE). This performance is called by the command *'mse.'* this performance function is built in the neural network toolbox [68]. It measures the

performance of the neural network based on the mean squared errors [68]. Choosing this performance function goes as follows:

```
% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error
```

Tracking the progress of the neural network is important. The neural networks toolbox provides these options. Three quantities are chosen to be traced throughout the learning process: performance, regression, and fit. To make this happen the functions *'plotperform,' 'plotregression,'* and *'plotfit'* are used. All of these functions provide a visual trace of the learning process in terms of performance, regression, and function fit, respectively [69], [70], [71]. The code for these specifications is:

```
% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plotregression', 'plotfit'};
```

Next, is training the neural network. The MATLAB™ command *'train'* trains a neural network as long as the training function, inputs, and outputs are specified by the user [72]. The code for this is quite simple:

```
% Train the Network
[net,tr] = train(net,x,t);
```

After the learning is complete, the network's performance is tested. The function *'perform'* "returns network performance calculated according to the net.performFcn and net.performParam property values" [73]. First, the error is calculated and them is used as an input argument to the function, *'perform.'* After that, the performance function is recalculated for the three data sets: training, validation, and testing. The code follows:

```
\\\\\\\% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)
```