TA/SEKJUR/TE/2016/062

FEEDFORWARD CONTROL OF A DC MOTOR USING A NEURAL NETWORK

Diajukan Sebagai Salah Satu Syarat

Untuk Memperoleh Gelar Sarjana Teknik Elektro

Oleh :

Nama	:	**Fahmi Adi Nugraha**
No. Mahasiswa	:	**10524035**

JURUSAN TEKNIK ELEKTRO

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

YOGYAKARTA

2016

FEEDFORWARD CONTROL OF A DC MOTOR USING A NEURAL NETWORK

Diajukan Sebagai Salah Satu Syarat

Untuk Memperoleh Gelar Sarjana Teknik Elektro

Oleh :

Nama	**: Fahmi Adi Nugraha**
No. Mahasiswa	**: 10524035**

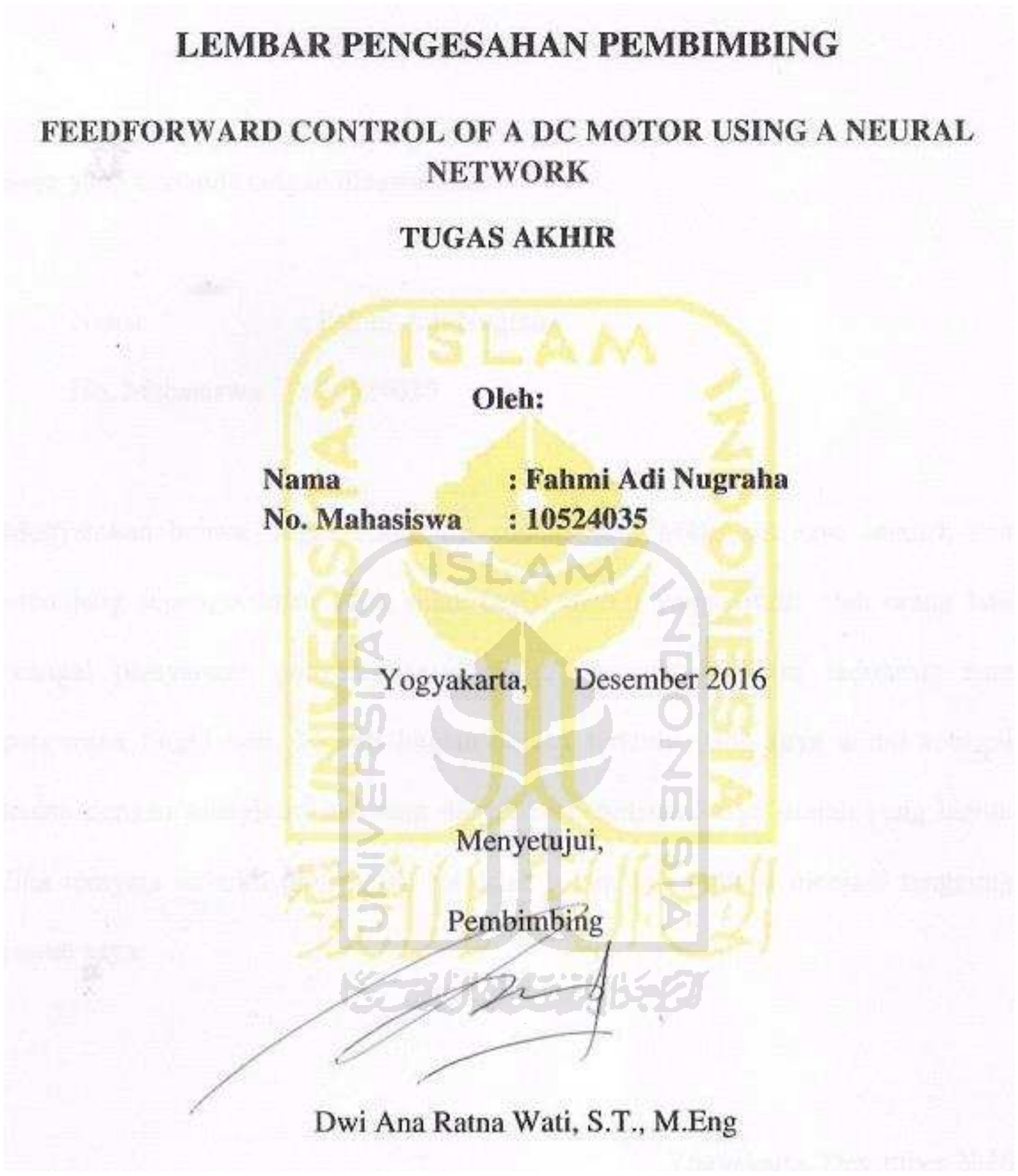
JURUSAN TEKNIK ELEKTRO

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

YOGYAKARTA

2016



LEMBAR PENGESAHAN PEMBIMBING

FEEDFORWARD CONTROL OF A DC MOTOR USING A NEURAL NETWORK

TUGAS AKHIR

Oleh:

Nama	**: Fahmi Adi Nugraha**
No. Mahasiswa	**: 10524035**

Yogyakarta, Desember 2016

Menyetujui,

Pembimbing

[illegible]

Dwi Ana Ratna Wati, S.T., M.Eng

LEMBAR PERNYATAAN KEASLIAN

Saya yang bertanda tangan dibawah ini:

Nama	: Fahmi Adi Nugraha
No. Mahasiswa	: 10524035

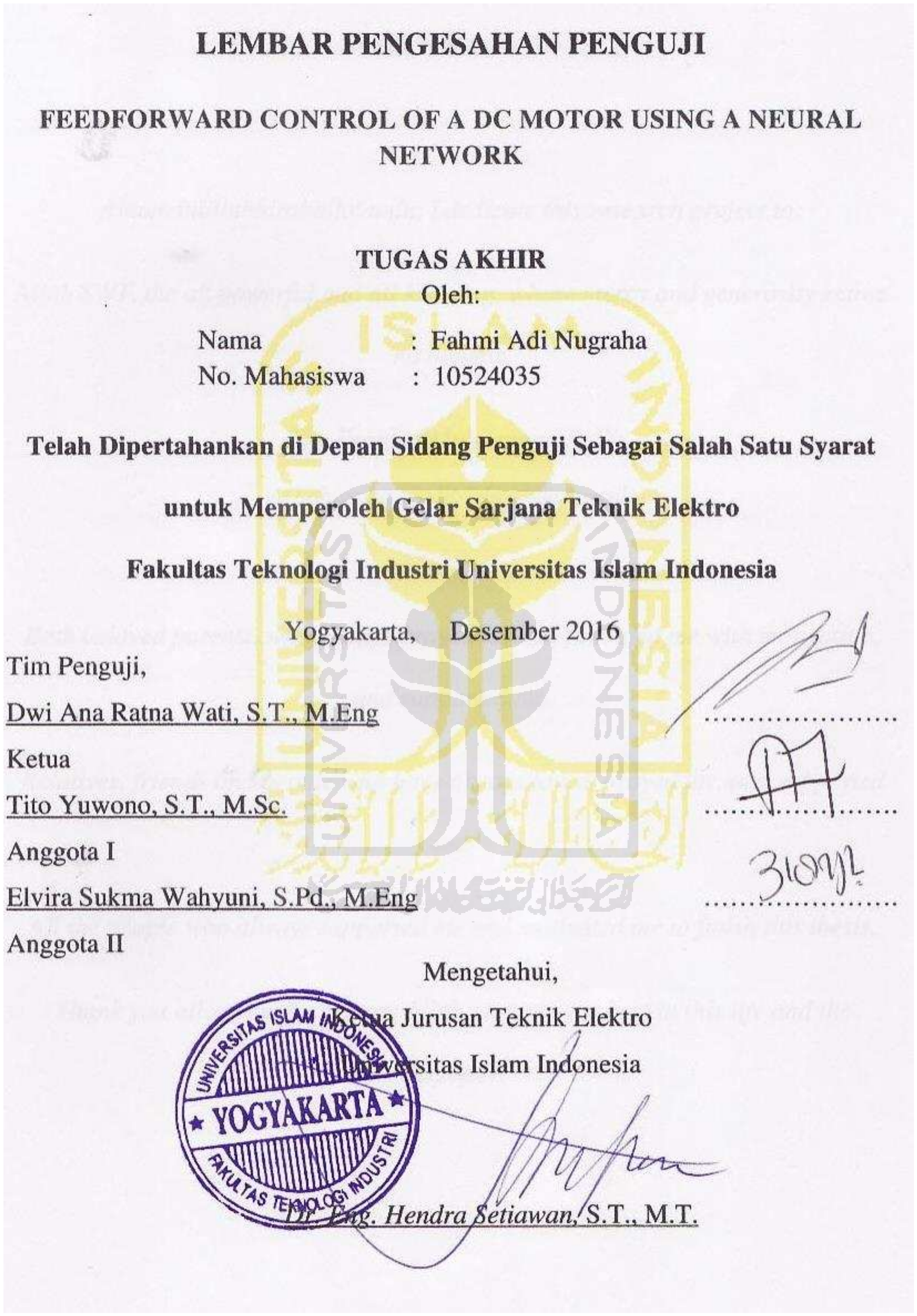
Menyatakan bahwa Tugas Akhir ini adalah hasil pekerjaan saya sendiri, dan sepanjang sepengetahuan saya, tidak berisi materi yang ditulis oleh orang lain sebagai persyaratan penyelesaian studi di Universitas Islam Indonesia atau perguruan tinggi lain, kecuali bagian-bagian tertentu yang saya ambil sebagai acuan dengan mengikuti tata cara dan etika penulisan karya ilmiah yang lazim. Jika ternyata terbukti pernyataan ini tidak benar, sepenuhnya menjadi tanggung jawab saya.

Yogyakarta, Desember 2016

METERAI TEMPEL
TGL 20
46EFDAEF094278712
6000
ENAM RIBU RUPIAH

[illegible]

Fahmi Adi Nugraha



LEMBAR PENGESAHAN PENGUJI

FEEDFORWARD CONTROL OF A DC MOTOR USING A NEURAL NETWORK

TUGAS AKHIR

Oleh:

Nama	: Fahmi Adi Nugraha
No. Mahasiswa	: 10524035

Telah Dipertahankan di Depan Sidang Penguji Sebagai Salah Satu Syarat

untuk Memperoleh Gelar Sarjana Teknik Elektro

Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, Desember 2016

Tim Penguji,

Dwi Ana Ratna Wati, S.T., M.Eng Ketua	[illegible] ....
Tito Yuwono, S.T., M.Sc. Anggota I	[illegible] ....
Elvira Sukma Wahyuni, S.Pd., M.Eng Anggota II	[illegible] ....

Mengetahui,

Ketua Jurusan Teknik Elektro

Universitas Islam Indonesia

UNIVERSITAS ISLAM INDONESIA
YOGYAKARTA
FAKULTAS TEKNOLOGI INDUSTRI

[illegible]

Dr. Eng. Hendra Setiawan, S.T., M.T.

Acknowledgements

Alhamdulillahhirobbilalamin, I dedicate this research project to:

Allah SWT, the all powerful and all knowing, whose mercy and generosity knows no bounds.

The Prophet Muhammad SAW

(Allahumma Sholli 'ala Muhammad)

Both beloved parents, who always prayed for me, provided me with motivation, and supported me.

Relatives, friends and people who have always loved, prayed for, and supported me.

All the people who always supported me and motivated me to finish this thesis.

Thank you all so much and may Allah give you the best in this life and the hereafter.

Mottos

- ***If you can't fly then run, if you can't run then walk, if you can't walk then crawl, but whatever you do you have to keep moving forward.***

  -Martin Luther King, Jr.

- ***Live as if you were to die tomorrow. Learn as if you were to live forever***

  -Mahatma Ghandi

- ***Any fool can know. The point is to understand.***

  -Albert Einstein

Preface

Bismillahirrahmanirrahim,

I am deeply grateful to Allah for the aid he has given me during the course of working on my research project and the writing of this report. He was the one who made everything possible, from providing me with the idea for my thesis topic to ensuring that my work on this project progressed smoothly and without too much trouble.

This report, entitled “Feed Forward Control of a DC Motor using Neural Networks”, is based on the results of developing a neural network controller for feed forward control of a DC motor. It is one of the prerequisites for graduating from the Electrical Engineering Undergraduate Program at Universitas Islam Indonesia.

I realize that conducting research can never be a truly solitary endeavor. At each step of my research project, I received help from a great many people. Because of this, I would like to use this opportunity to thank:

1. My mother and my father for their love and support. Without them, I would have had great difficulty finishing my internship.
2. Dwi Ana Ratnawati for acting as my advisor and helping me complete my thesis.
3. Heri for providing assistance with the equipment at the Control and Industrial Automation Laboratory.
4. My good friend Iqbal, for providing the opportunity to teach the subject of neural networks and, thus, solidify the knowledge in my mind.
5. All of the members of the research project group study sessions headed by Bu Ana.
6. All of the lectures in the Electrical Engineering Department
7. All of my friends at Universitas Islam Indonesia

Despite my efforts, I know this report still suffers from many shortcomings and is far from perfect. My only hope is for it to provide some benefit to the people who take the time to read it.

Yogyakarta, September 2016

Fahmi Adi Nugraha

Table of Contents

Title Page i

Confirmation from the Supervisor ii

Form of Authentication iii

Confirmation from the Examiners iv

Acknowledgments v

Mottos vi

Preface vii

Table of Contents ix

List of Figures... xi

List of Tables xii

Chapter 1 Introduction

1.1 Background 1

1.2 Problem Statement 2

1.3 Purpose of Research 2

1.4 Benefits of Research 2

1.5 Scope of Research 2

1.6 Research Method 2

1.7 Report Structure 3

Chapter 2 Literature Review

2.1 Neural Networks 5

2.2 Neural Network Training 8

2.2.1 The Concept of Learning in Neural Networks …................... 8

2.2.2 Backpropagation ……………….…………..………….......10

2.2.3 The Levenberg-Marquardt Algorithm ..……………….........11

2.3 DC Motor Control Schemes using NNs ...……………………….…13

2.4 Summary of Previous Research in this Field ..………………….…15

Chapter 3 Design of the Feed Forward DC Motor Control System

3.1 Neural Network Controller Architecture ……...…………………… 16

3.2 Neural Network Controller Training ……………………………… 18

3.3 Applying the Neural Network to the DC Motor ...…………….……20

Chapter 4 Result Analysis

4.1 Step Response ……………………....…………………………..23

4.2 Steady State Tracking with no Load ...…...…………………………25

4.3 Steady State Tracking with Full Load …...………………………… 27

4.4 Steady State with Variable Load ......……………………………… 29

Chapter 5 Closing

5.1 Conclusion....……………………………………………………. 33

5.2 Suggestions for Future Research …………………………………... 34

DAFTAR PUSTAKA. …………………………………………………………35

List of Figures

Figure 2.1 A biological neuron……….................…………………………….. 6

Figure 2.2 The perceptron model of a neuron……...………………………6

Figure 2.3 Illustration of an MLP……………..…………………………… 7

Figure 2.4 Pseudocode of the LM algorithm ………………………………. 11

Figure 2.5 DC motor control schemes ……………………………………... 14

Figure 3.1 NN controller architecture…………………..…………………... 16

Figure 3.2 NN activation functions……………………………………….18

Figure 3.3 Scatterplot of NN controller training data................................... 19

Figure 3.4 MATLAB training results..……………….………………..……19

Figure 3.5 DC motor control system……………………………………... 21

Figure 3.6 DC motor control VI…………………………………………… 21

Figure 4.1 Step response graph ..……………………………………….…... 24

Figure 4.2 Graph of steady-state tracking of DC motor with no load ...…… 26

Figure 4.3 Graph of steady-state tracking of DC motor with full load...……28

Figure 4.4 Steady state at different load percentages ……………………… 32

List of Tables

Table 4.1 Step response characteristics errors for each system ……….….… 25

Table 4.2 Load Percentages and their associated steady-state errors ……… 29

Chapter 1
Introduction

1.1. Background

PID controllers dominate the world of industrial automation with their prevalence being such that they have become the de facto standard controller. Similarly, in many industries it is hard to find a process, system, or appliance that does not incorporate a DC motor in some way. The ubiquity of DC motors is almost on par with that of PID controllers, as they are used in everything from cranes, conveyor belts, extruders, mixers, and machine presses. Since both DC motors and PID controllers find such wide usage in many industrial applications, it is very common to see DC motors being controlled by PID controllers.

While PID controllers provide reasonable performance for the control systems to which they have been applied, they suffer from several limitations. The most prominent of which are their linearity and their susceptibility towards uncertainties within the plant. When placed in control of a nonlinear system, such as a DC motor, PID controllers cannot offer optimal performance.

A promising solution is to use a neural network (NN) in place of a PID controller. One of the greatest strengths of the NN is its ability to approximate both linear and nonlinear functions of arbitrary complexity. This makes NNs capable of developing a model for processes that are difficult or even impossible to model through analytical methods. Given the nonlinear nature of a DC motor, a NN is quite capable of functioning as a controller in a DC motor control system.

Despite this advantage, there are some obstacles that must be surmounted before NNs see widespread use. The biggest of these is developing an accurate model of a DC motor such that it captures the behavior of the motor. Due to their being based in statistics, NNs require data in order to build a model of a given process. For a complex process such as a DC motor the NN requires data from the motor under as many different conditions as possible. In many cases, it is very difficult to obtain this data.

Due to this, many researchers in this area use online training methods in order to acquire the data directly. However, in practice this is not always feasible as these methods tend to be very technical and implementing them as part of a practical application requires extensive knowledge in the field of NNs. A simpler approach is to begin with a feed forward system and then build off of it.

1.2. Problem Statement

How do we design and train a neural network to act as a feed forward controller for a DC motor that provides good performance?

1.3. Purpose of Research

The purpose of this research is to design a feed forward neural network controller for a DC motor and analyze its performance.

1.4. Benefits of Research

There are several benefits provided by this research, such as:

1. Providing the configuration of a neural network controller and the optimization algorithm used to train it.
2. Detailing the method used to train the neural network controller for feed forward control.
3. Serving as a stepping stone for future research in this area.

1.5. Scope of Research

This research encompasses the following:

1. The type of DC motor along with its associated model. For this research project the DC motor in my university's Control and Industrial Automation Lab.

2. The number of hidden layers and hidden neurons used in the NN controller as well as the algorithm used to train it. For this project, the NN controller has three layers: two hidden layers and one output layer. The first hidden layer has three hidden neurons, the second has two, and the output layer has one output layer. The activation function of each hidden neuron is the hyperbolic tangent function while the activation function of the output neuron is the linear function.
3. The configuration of the control system.

1.6. Research Method

The methodology used in this research is to first learn the underlying theory of NNs and DC motors. After understanding the underlying theory both the NN architecture and the control scheme will be determined. Once this is complete the next step is to acquire data from a DC motor, train the NN on it, and evaluate the performance of the trained NN. The final step is to insert the NN into the DC motor feed forward control scheme and collect the resulting data.

1.7. Report Structure

This report consists of five chapters, the contents and order of which are as follows:

Chapter 1 Introduction

This chapter consists of the title, background, problem statement, purpose of the research, benefits of the research, scope of the research, research method, and report outline.

Chapter 2 Literature Review

This chapter discusses NNs and their training as well as provides a summary of previous research in the field.

Chapter 3 Design of the Feed Forward DC Motor Control System

This chapter describes the architecture of the NN controller and the method used to train it. It also describes the DC motor control system and how it was implemented.

Chapter 4 Result Analysis

This chapter consists of an analysis and evaluation of the NN controller as part of the feed forward DC motor control system.

Chapter 5 Closing

This chapter consists of the conclusions derived from this research and suggestions for future research.

Chapter 2
Literature Review

Despite having existed as a concept since as early as 1943 the study of NNs for use in control systems is relatively recent. Research in this area with regards to the control of a DC motor mainly focuses on adaptive control or predictive control. Before we can discuss this research, it is imperative to examine the operating principles behind NNs in order to have a better understanding of how they are applied to control systems. We will also briefly look at adaptive and predictive control schemes and how they function.

2.1 Neural Networks

NNs draw inspiration from the human brain, which consists of an extremely large number of neurons connected together to form a massive network. The neuron serves as the primary computing element of the NN. From Figure 2.1, we can see that a neuron consists of three main components: the dendrites, the cell body, and the axon. The dendrites accept input signals from other neurons that have been weighted by the strength of the connection between the neuron receiving the input signals and the neurons transmitting them. The cell body collects the weighted input signals, sums them, and generates an output signal if the sum of the inputs exceeds a certain threshold. The axon sends the output signal from the cell body to other neurons [1].

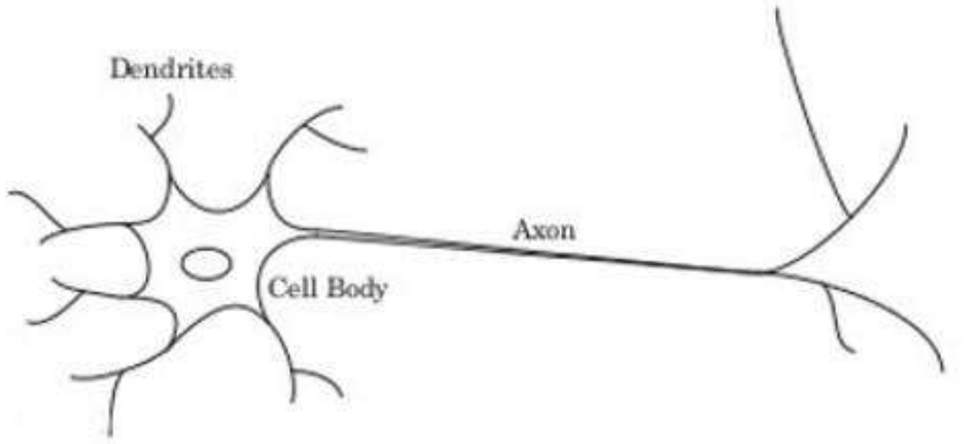


Figure 2.1. A biological neuron

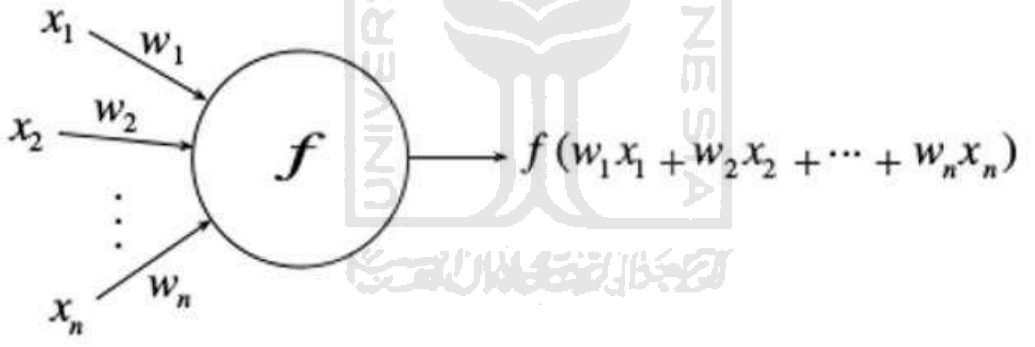
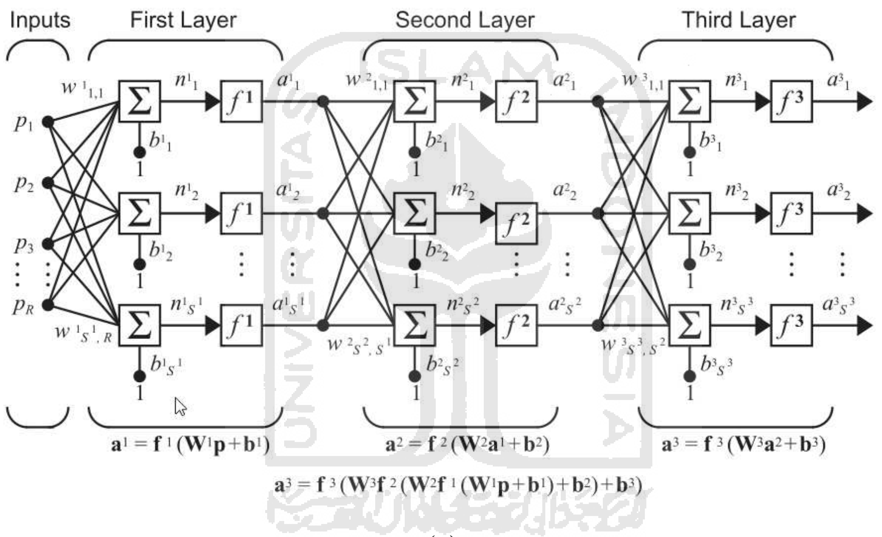


Figure 2.2. The perceptron model of a neuron

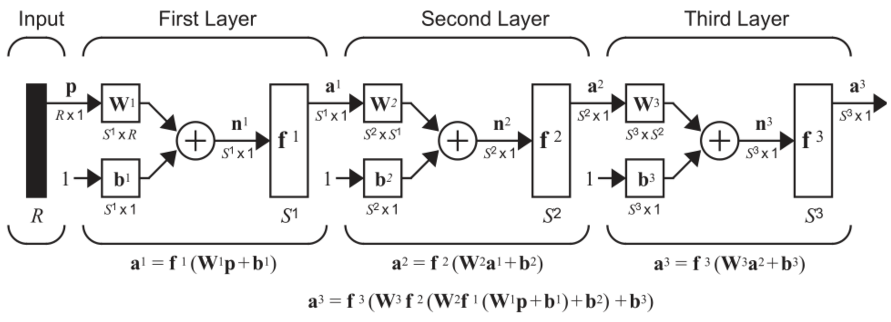
The neuron essentially performs three operations: multiplying the input signals by the connection weights, taking the sum of the input signals, and passing the sum of the input signals through an activation function. With this knowledge, we can create an abstract model of the neurons that encapsulates these operations. Such a model is shown in Figure 2.2 and is known as the perceptron model of the neuron.

Perceptrons by themselves do not possess a great deal of computing ability. However, when combined into a network their computing power increases dramatically and it is through these networks that perceptrons derive their uncanny ability to approximate nearly any function. These perceptron networks, known as multilayer perceptrons (MLP), are comprised of three layers: the input layer, the hidden layer, and the output layer. It should be noted that while there is only one input layer and one output layer, there can be more than one hidden layer. Each layer can contain an arbitrary number of neurons. An illustration of an MLP is provided in Figure 2.3a while an illustration of the same MLP with more compact notation is provided in Figure 2.3b.



Inputs
First Layer
Second Layer
Third Layer
$a^1 = f^1(W^1p + b^1)$
$a^2 = f^2(W^2a^1 + b^2)$
$a^3 = f^3(W^3a^2 + b^3)$
$a^3 = f^3(W^3f^2(W^2f^1(W^1p + b^1) + b^2) + b^3)$

(a)



Input
First Layer
Second Layer
Third Layer
$a^1 = f^1(W^1p + b^1)$
$a^2 = f^2(W^2a^1 + b^2)$
$a^3 = f^3(W^3a^2 + b^3)$
$a^3 = f^3(W^3f^2(W^2f^1(W^1p + b^1) + b^2) + b^3)$

(b)

Figure 2.3. Illustration of an MLP [3]: (a) Expanded notation, (b) compact notation.

We pause here to elaborate on the notation used in Figures 2.3 and 2.4 as they will be used for throughout the rest of this chapter. From these figures, we have:

- **p**: the inputs to the network
- **W**: the connection weights for a given layer, usually referred to as weights
- **b**: the layer biases
- **n**: the net layer input
- **f**: the layer activation function
- **a**: the layer output

2.2 Neural Network Training

Much like the human brain on which they are based, MLPs are capable of learning. Since the concept of learning is key to a proper understanding of NNs, the following sections briefly explain learning as it is understood within the context of NNs. An overview of the learning process is also provided.

2.2.1 The Concept of Learning in Neural Networks

Children are first taught about the world by being shown examples of the various objects that comprise it. They learn about trees, clouds, cats and dogs by looking at them. While the knowledge gained about these objects can, at times, be lacking it is usually sufficient for most purposes. NNs learn in much the same way. Presented with a dataset containing many examples of some set of objects described by a set of features along with a label indicating what the object is a NN, through a process of altering the values of its weights and biases, can learn to correctly identify the objects.

As an illustrative example, suppose we had a pile of assorted fruits and vegetables. We would like to teach a NN to be able to tell us whether an object drawn from this pile is a fruit or a vegetable. To do this we would select features, such as crunchiness and sweetness, that the NN could use to distinguish between fruits and vegetables. After ranking both the crunchiness and sweetness of each object in the pile according to some measurement scale, we then assign each object a label identifying them as either a fruit or vegetable. This set of object features and their associated label is called the training data. The features for each object are then fed to the NN and the resulting output is compared to the actual label for that object. If the label generated by the NN is not the same as the actual label, an error value is calculated and used to update the weights and biases of the NN in order to correct the output of the NN. This process of presenting object features and altering the weights and biases continues until a certain performance measure is reached. Usually this performance measure is having the error of a performance function such as the Mean Squared Error (MSE) fall below a certain threshold. Upon reaching this performance measure, the NN is said to be trained and should be able to accurately predict whether a given object is a fruit or vegetable [2].

The process described above is known as training and is an instance of supervised learning. Supervised learning is a learning paradigm that involves using a set of training examples consisting of both a set of features and the correct network output for those features. This is why supervised learning is often called "learning with a teacher." The majority of NNs used for practical applications are trained using supervised learning. The types of problems that supervised learning is used to solve can be divided into two broad categories: classification and regression. In a classification problem the output value, or target, is a class label. The fruit and vegetable identification example given above is an example of a classification. In a regression problem, the target is a continuous value. Function approximation is an example of regression. It is this latter problem that we will be focusing on for the remainder of this chapter.

Training a NN to solve a regression problem is not much different than training a NN to solve a classification problem. In fact, the main difference is that the output value is no longer a label indicating the class to which a training example belongs. Rather, the output value is a real number which represents the output of the function we are attempting to estimate when presented with the current input. The process of altering the values of the weights and biases remains the same in both cases. In MLPs, this process is known as backpropagation.

2.2.2 Backpropagation

Backpropagation makes up the bulk of the MLP learning process. Despite being a central component in NN training the idea behind it is quite simple. When updating the weights and biases of a NN, it is necessary for us to calculate the error. In the case of a single layer NN, this is as simple as subtracting the output of the NN from the target value associated with the current input. This is because the output of the layer neurons depends only on the immediately preceding set of weights and biases.

For an MLP, where there are multiple layers, the final output depends not only on the weights and biases of the output layer but also the weights and biases of the preceding layers. In complex networks with many hidden layers and neurons, even slight changes in the weight and bias values can cause the output of the NN to vary wildly. In order to determine the degree to which all of the weights in such a network should be altered, we need to know the error at each output of each layer. We do this by calculating the error for the output layer and propagating it back towards the input layer.

Backpropagation can be broken down into two stages: the backward error propagation stage and the weight and bias update stage. During the backward error propagation stage, the total network error is calculated according to some error function, usually the Sum of Squared Error (SSE) or the MSE as mentioned earlier. Once this is done, the error for each output neuron is computed and then passed backwards through each layer of the NN starting with the output layer. After all of the errors have been calculated, the weights and biases for the entire network are updated.

The entire NN training process is essentially an optimization problem. The goal is to find a set of weight and bias values that minimize an error function. Thus, the calculations used in the backpropagation process to compute the errors and update the weights and biases are entirely dependent on the optimization algorithm used to train the network. Many different optimization algorithms are used with some commonly used ones being Gradient Descent, Resilient Backpropagation, Conjugate Gradient, and Levenberg-Marquardt. In developing an NN controller for the DC motor used in this research project, the Levenberg-Marquardt algorithm was used.

2.2.3 The Levenberg-Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm can be seen as a combination of the Gradient Descent and Gauss-Newton algorithms. As a result, the LM algorithm inherits the strengths of both algorithms, namely stability and quick convergence. The LM algorithm is primarily a batch or offline training algorithm, which means that it is designed to be trained on data that has already been collected. This, in conjunction with its previously mentioned strengths, makes the LM algorithm ideal for training an NN controller on the data acquired from a live DC motor.

```
Initialize Weights
while not StopCriterion
do {
    calculate $e^p(w)$ for each pattern
    calculate $e1 = \sum_{p=1}^{P} [e^p(w)]^2$
    calculate $J^p(w)$
    repeat
      calculate $\Delta w$
      $\Delta w = -[\mu I + \sum_{p=1}^{P} J^p(w)^T J^p(w)]^{-1} \nabla E(w)$
      $e2 = \sum_{p=1}^{P} e^p(w + \Delta w)^T e^p(w + \Delta w)$
      if $e1 \leq e2$
        then $\mu = \mu * \beta$
    until $(e2 < e1)$
    $\mu = \mu / \beta$
    $w = w + \Delta w$
```

Figure 2.4. Pseudocode of the LM algorithm

Pseudocode of the LM algorithm is shown in Figure 2.4. From this pseudocode, we see that the LM algorithm consists of three main stages: calculating the error on the current weights, calculating the Jacobian matrix for the current weights, and updating the weights. In order to aid in visualizing the calculations, refer to the illustration of an MLP using compact notation presented in Figure 2.3b

Calculating the error on the current weights involves presenting each pattern from the set of training data and propagating them forward through the network according to the following equations [3]:

$$\boldsymbol{a^0} = \boldsymbol{p}, \tag{2.1}$$

$$\boldsymbol{a^{m+1}} = \boldsymbol{f^{m+1}}(\boldsymbol{W^{m+1}a^m} + \boldsymbol{b^{m+1}}) \text{ for m = 0,1,…, M-1} \tag{2.2}$$

Once $\boldsymbol{a^M}$ is obtained for the current pattern, the error can be obtained by:

$$\boldsymbol{e_q} = \boldsymbol{t_q} - \boldsymbol{a_q^M} \tag{2.3}$$

We then calculate the SSE for all patterns using:

$$\boldsymbol{F(w_k)} = \sum_{q=1}^{Q}(\boldsymbol{t_q} - \boldsymbol{a_q})^T(\boldsymbol{t_q} - \boldsymbol{a_q}) = \sum_{q=1}^{Q}\boldsymbol{e_q^T e_q} = \sum_{q=1}^{Q}\sum_{j=1}^{M}(\boldsymbol{e_{j,q}})^2 = \sum_{i=1}^{N}(\boldsymbol{v_i})^2 \tag{2.4}$$

The Jacobian matrix is calculated by first computing the sensitivities for each layer with:

$$\widetilde{\boldsymbol{S_q^M}} = -\dot{\boldsymbol{F}}^{\boldsymbol{M}}(\boldsymbol{n_q^M}) \tag{2.5}$$

$$\widetilde{\boldsymbol{S_q^m}} = \dot{\boldsymbol{F}}^{\boldsymbol{m}}(\boldsymbol{n_q^m})(\boldsymbol{W^{m+1}})^T\widetilde{\boldsymbol{S_q^{m+1}}} \tag{2.6}$$

where $\dot{\boldsymbol{F}}$ is defined as:

$$\dot{\boldsymbol{F}}^{\boldsymbol{m}}(\boldsymbol{n^m}) = \begin{bmatrix} \dot{\boldsymbol{f}}^{\boldsymbol{m}}(\boldsymbol{n_1^m}) & 0 & \cdots & 0 \\ 0 & \dot{\boldsymbol{f}}^{\boldsymbol{m}}(\boldsymbol{n_2^m}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{\boldsymbol{f}}^{\boldsymbol{m}}(\boldsymbol{n_{S^m}^m}) \end{bmatrix} \tag{2.7}$$

Once the sensitivities have been calculated, they are then combined according to:

$$\widetilde{\boldsymbol{S^m}} = \left[\widetilde{\boldsymbol{S_1^m}} \mid \widetilde{\boldsymbol{S_2^m}} \mid \dots \mid \widetilde{\boldsymbol{S_Q^m}}\right] \tag{2.8}$$

From here, the individual elements of the Jacobian matrix can be calculated using:

$$[\boldsymbol{J}]_{\boldsymbol{h,l}} = \widetilde{\boldsymbol{s_{i,h}^m}}\,\widetilde{\boldsymbol{a_{j,q}^{m-1}}} \tag{2.9}$$

for each weight and

$$[\boldsymbol{J}]_{\boldsymbol{h},\boldsymbol{l}} = \widetilde{\boldsymbol{s}_{\boldsymbol{i},\boldsymbol{h}}^{\boldsymbol{m}}} \tag{2.10}$$

for each bias.

The final stage, updating the weights, requires the calculation of $\Delta \boldsymbol{w_k}$. This is done through the following equation:

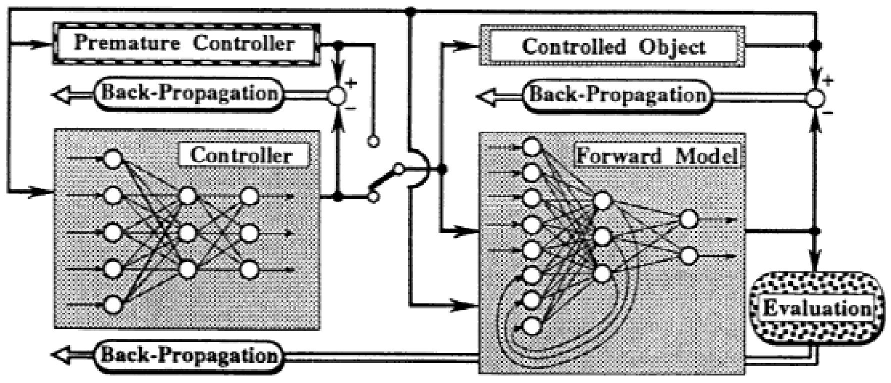
$$\Delta \boldsymbol{w_k} = -[\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{w_k})\boldsymbol{J}(\boldsymbol{w_k}) + \boldsymbol{\mu_k I}]^{-\boldsymbol{1}}\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{w_k})\boldsymbol{v}(\boldsymbol{w_k}) \tag{2.11}$$

Recompute the SSE using $\boldsymbol{w_k} + \Delta \boldsymbol{w_k}$. If this new SSE is smaller than that computed using $\boldsymbol{w_k}$ then divide $\boldsymbol{\mu}$ by some factor **f**, let $\boldsymbol{w_k} = \boldsymbol{w_k} + \Delta \boldsymbol{w_k}$ and go back to the first stage. If the SSE is not reduced, then multiply $\boldsymbol{\mu}$ by **f** and repeat the weight update stage [3,4].

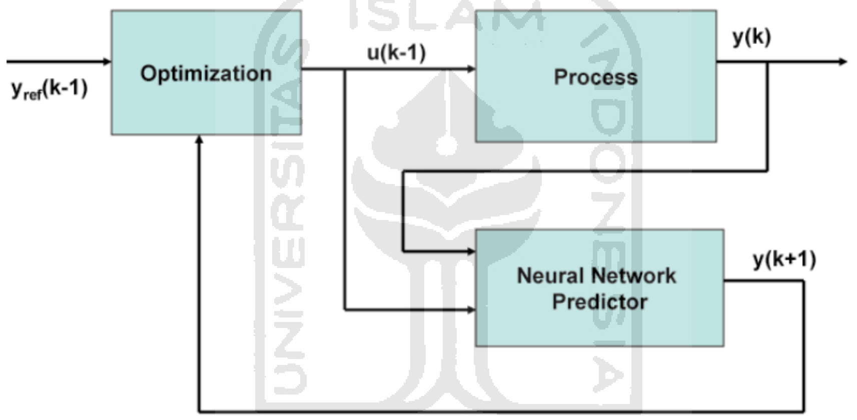
All three stages of the LM algorithm are repeated until a stopping condition is met. Usually there is more than one stopping condition for a given training session. These stopping conditions include exceeding a maximum number of epochs, having the SSE fall below a certain threshold, having $\boldsymbol{\mu}$ exceed a certain maximum or minimum value, or having the gradient exceed a certain maximum or minimum value.

2.3 DC Motor Control Schemes using NNs

NNs are typically used to control DC motors in one of two types of control schemes: adaptive control schemes or predictive control schemes. In an adaptive control scheme, corrections are made to the NN controller in real time based on the current error. The error is calculated as either the difference between the set point and the DC motor output or the difference between the output of a reference model and the DC motor output. The vast majority of adaptive control schemes are closed loop in nature as they very often require the output of the DC motor to be fed back so as to calculate the error. Even those implementations that calculate the error based on a reference model need to have the error fed back to the NNs. Figure 2.5a shows an example of this latter adaptive control scheme.



(a)



(b)

Figure 2.5. DC motor control schemes: (a) an example of an adaptive control scheme, (b) an example of a predictive control scheme

Rather than adjust the NN controller in response to the error in the system, a predictive control scheme anticipates the error at the next time step and provides the control signal value necessary to reduce this error. In order to do this, a model of the plant is first developed using system identification techniques. This model is then used to generate an estimate of the error which the NN controller can then use to prepare the correct control signal. In several implementations of the predictive control scheme, the model and controller are combined into a single NN. Predictive control schemes make use of previous system inputs and outputs far more often than adaptive control schemes. Figure 2.5b shows an example of a predictive control scheme.

2.4 Summary of Previous Research in this Field

The majority of research concerning the control of a DC motor through a NN controller makes use of either adaptive or predictive control schemes. In the case of adaptive control schemes, there seem to be two trends. The first trend is to have NNs serve as both the reference model and the controller [4-7] while the second is to have a separate controller whose parameters are adjusted by a NN [8,9]. A significant shortcoming of feedforward NNs such as MLPs is that they cannot process sequences of inputs. This prevents them from learning more complex relationships within the data of processes like DC motors that would allow them to develop a more accurate model of the process. Due to this, some implementations of the adaptive control scheme make use of Recurrent Neural Networks which overcome this limitation [4,5].

The implementations of predictive control schemes follow a similar set of trends as the implementations of adaptive control schemes. The first noticeable trend is the use of a pair of NNs, one to act as the controller and the other to act as the DC motor model [10-16] where the purpose of the model is to predict the system error in the next time step. In some implementations, the NN model and NN controller are combined into a single unit [14, 16]. The next trend is an extension of the first and involves using the predictions made by the NN to adjust a controller which may or may not be a NN [17-19]. The final trend is to train a single NN to act as both a predictor and a controller [20].

The one thing that both adaptive and predictive control schemes have in common is that they are both closed loop systems. Unfortunately, compared to closed loop systems research about feedforward control of DC motors using NNs is rather scarce. This research project seeks to rectify this problem by providing a basis for further research in this area.

Chapter 3
Design of the Feed Forward DC Motor Control System

3.1 Neural Network Controller Architecture

The NN architecture used for the NN controller is shown in Figure 3.1. From the figure, we see that the NN controller taken as a whole has one input and one output. Looking at the structure of the NN controller reveals that it consists of three layers: two hidden layers and one output layer. The first hidden layer has three hidden neurons, the second hidden layer has two hidden neurons, and the output layer has only one output neuron. Borrowing the terminology of the field of NN research, this NN has a 3-2-1 architecture. The neurons of both hidden layers have as their activation function the hyperbolic tangent function, $\mathbf{tanh}(\boldsymbol{x}) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ (Figure 3.2a), while the activation function of the output neuron is a linear function, $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}$ (Figure 3.2b). All three layers of the NN controller have biases.

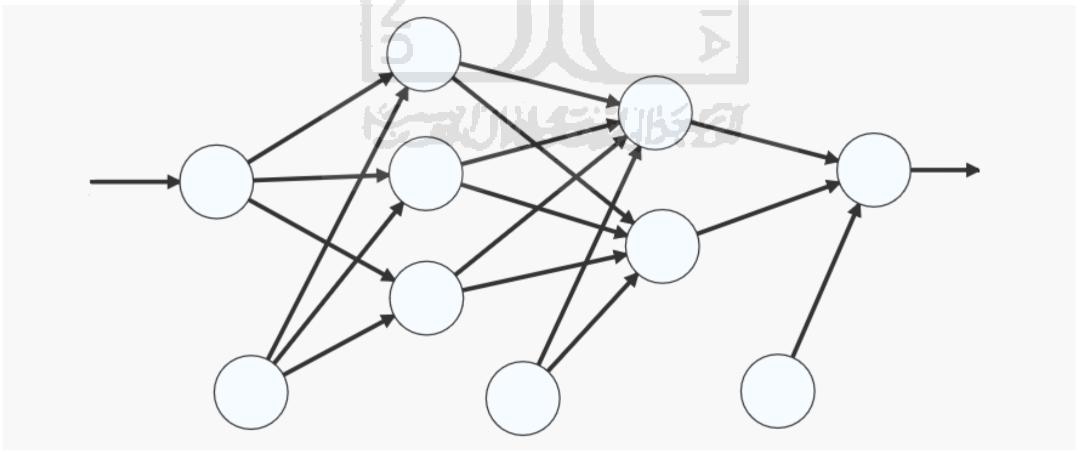
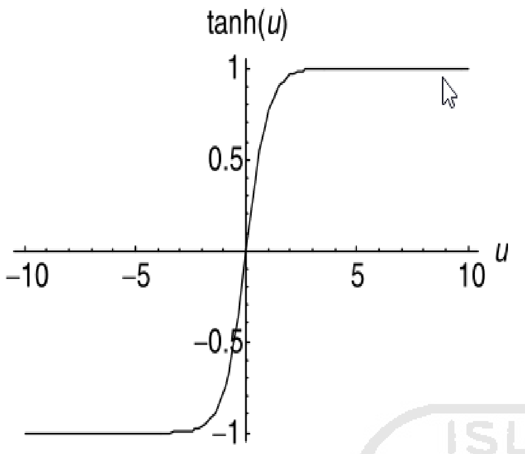
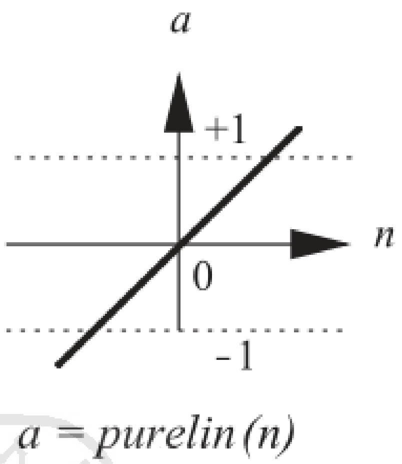


Figure 3.1. NN controller architecture



(a)



(b)

Figure 3.2. NN activation functions: (a) Hyperbolic tangent function, (b) linear function.

Designing a NN for any application can be a rather difficult task. Unlike the mathematical computations that make up a NN and its many training algorithms, there are no strict rules or guidelines for choosing an optimum NN architecture for a given application. Thus a fair amount of time is spent at the beginning of any endeavor involving NNs narrowing down a set of NN architectures through a process of trial-and-error. Each NN architecture in this set of potential NN architectures is trained on the data obtained for the application and its performance measured. The performance of each NN architecture is then compared to identify the one with the best performance as specified for the given application. A similar process was used to select the NN architecture used in this research project. At the start of the project, there were two NN architectures: 2-1 and 3-2-1. After training the two architectures on sample DC motor data, it was determined that the 3-2-1 NN architecture yielded the best performance in terms of the MSE.

3.2 Neural Network Controller Training

Since the goal of this research project is to design a feedforward control system for a DC motor, the NN controller is trained as an inverse model of the DC motor. For this project, the plant consisted of the DC motor along with the PWM generator that controls the speed of the DC motor. This means that the training data gathered from the DC motor contained speed-duty cycle percentage value pairs. Therefore, when training the NN controller the order of these value pairs was swapped so that the input of the NN controller was the motor speed and the output was the duty cycle percentage of the PWM generator.

As mentioned in the previous chapter, the NN controller was trained using the LM algorithm. The data used to train the NN controller was taken from a live DC motor with no load. During this research project, the NN controller was trained using validation-based early stopping in MATLAB.

For the NN controller training, the data acquired from the DC motor was randomly divided into three sets: a training set, a validation set, and a test set. 70 percent of the original DC motor dataset made up the training set while 15 percent made up both the validation and training sets. The training and validation sets were then passed to the NN training algorithm where training proceeded until the error on the validation set increased for six consecutive epochs.

The MATLAB vendor MathWorks provides a NN toolbox which was included in the installation of MATLAB used in this project. Through the use of this toolbox, the splitting of the data as well as the training itself was handled by MATLAB. After importing the data into MATLAB the LM algorithm used in the NN toolbox was able to converge. The results of the NN Controller training are shown in Figure 3.4. The plot of the training set, validation set, and test set errors show that the MSE, while not particularly small, is acceptable for the data used to train the network (shown in Figure 3.3). The plots of the correlation between the network outputs and target values for the three sets indicate good performance from the network as the tests of correlation for the sets all have **r** values of around 0.99.

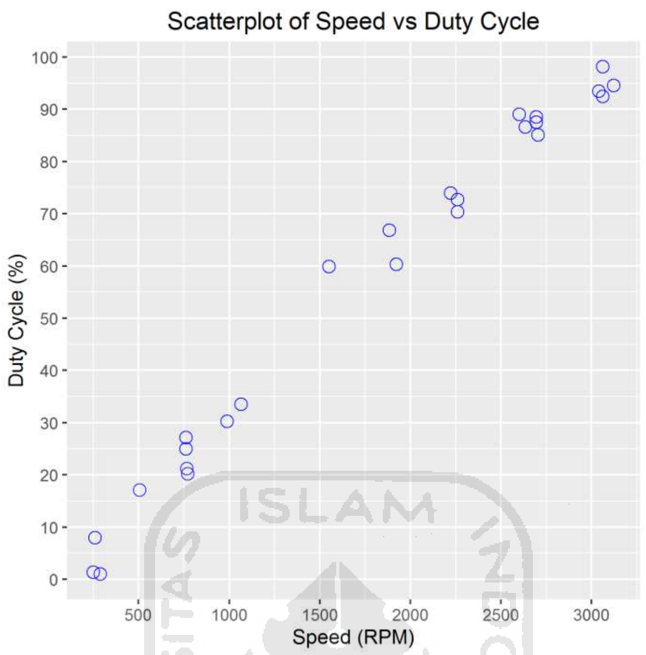
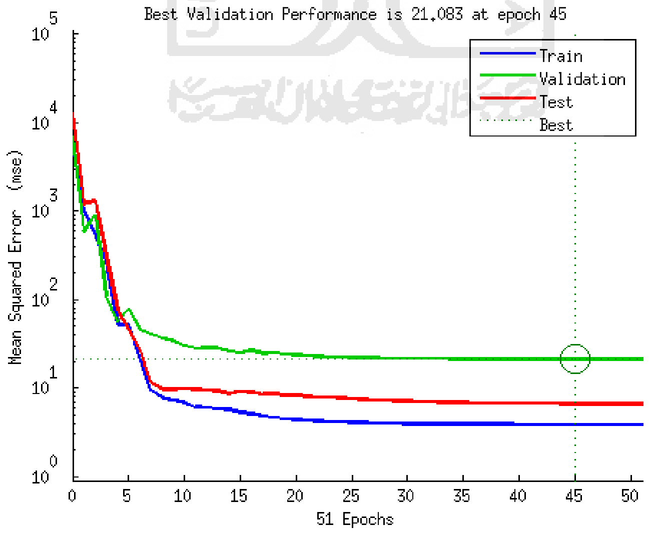
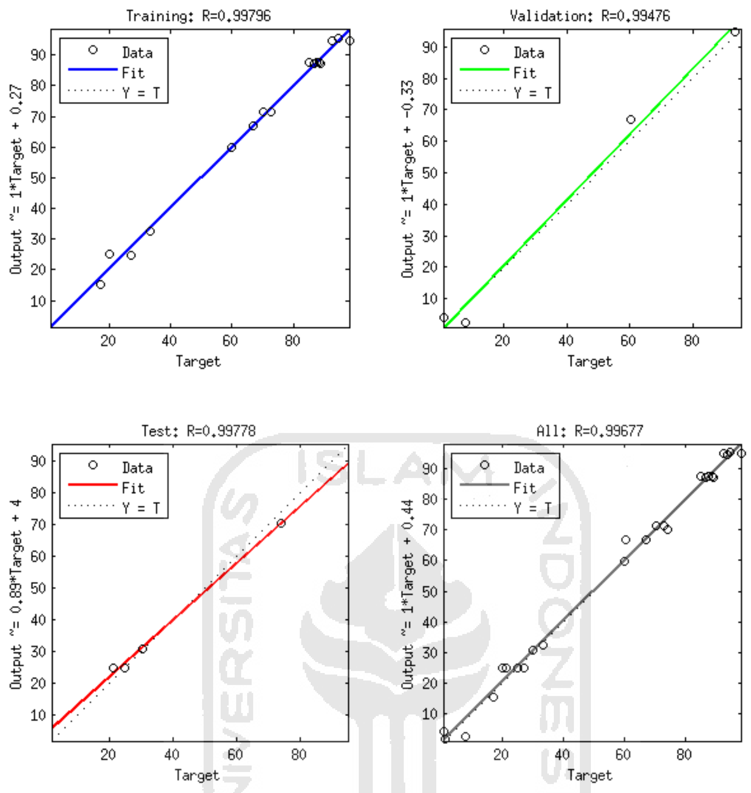


Figure 3.3. Scatterplot of NN controller training data



(a)



(b)

Figure 3.4. MATLAB training results: (a) Error on the training, validation and test sets, (b) correlation between the outputs of each set and the target values.

3.3 Applying the Neural Network Controller to the DC Motor

The DC motor used in the project was part of a digital control system implemented in LabVIEW that made use of a National Instruments USB-6008 DAQ to mediate between the computer with the LabVIEW control VI and the DC motor. Figure 3.5 shows a block diagram of the entire control system.

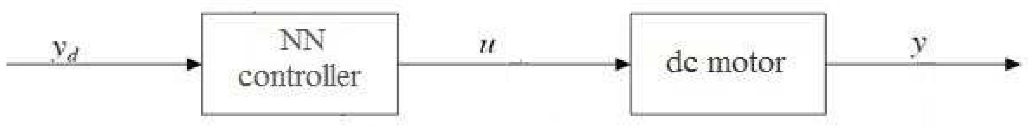
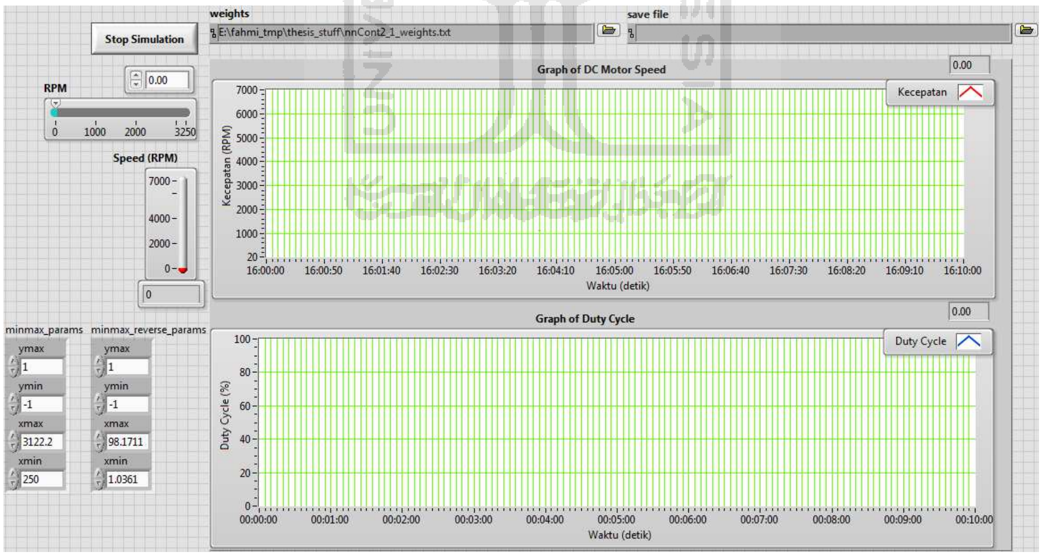


Figure 3.5. DC motor control system

As mentioned earlier in this chapter, the plant for this system consists of the DC motor and a PWM generator. The VI controls the speed of the DC motor by allowing the user to set the speed of the DC motor (y_d), where the speed is measured in RPM. This value is then passed to the trained NN controller which converts the RPM value to a duty cycle percentage (u). The VI then sends a PWM signal to the DC motor. The VI can be seen in Figure 3.6.

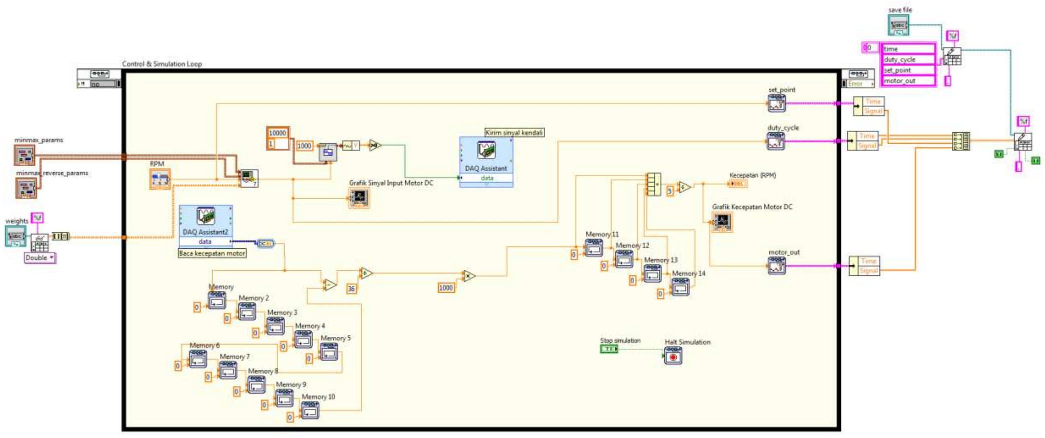


weights: E:\fahmi_tmp\thesis_stuff\nnCont2_1_weights.txt

save file

minmax_params		minmax_reverse_params	
ymax	1	ymax	1
ymin	-1	ymin	-1
xmax	3122.2	xmax	98.1711
xmin	250	xmin	1.0361

(a)



(b)

Figure 3.6. DC motor control VI: (a) Front panel, (b) block diagram

Chapter 4
Result Analysis

In this chapter, we evaluate the performance of the NN controller. We begin by examining the step response of the DC motor without any controller and compare it to that of the DC motor with the NN controller. We then look at the results of subjecting the NN control system to a series of tests. The first of these tests was to set the load on the DC motor to 0 percent while varying the set point of the control system. The next test was identical to the first with the only difference being the load on the DC motor which was set to100 percent. The final test was to select a constant value for the set point of the control system and vary the load. We compare the results of each of these tests to those of an equivalent control system that has a single variable linear regression model (LRM) in place of an NN controller.

4.1 Step Response

The step response was obtained from the DC motor by itself as well as the DC motor with the NN controller. Since an alternative control system using a LRM was used as a point of comparison for the performance tests, the step response of this system was taken as well. Below is the LRM equation showing the values of the coefficients.

$$\boldsymbol{y = -1.47298826 + 0.03264442x}$$

The use of a live DC motor for this project means that the step response characteristics could not be determined analytically as there was no mathematical model from which to derive the necessary calculations. For this reason, the step response characteristics for each of these systems were determined as follows. The value of 2000 RPM was chosen to be the set point for each of these systems and from the resulting motor speed data the step response characteristics were approximated using MATLAB. Figure 4.1 shows the graphs of the step response for each of these systems.

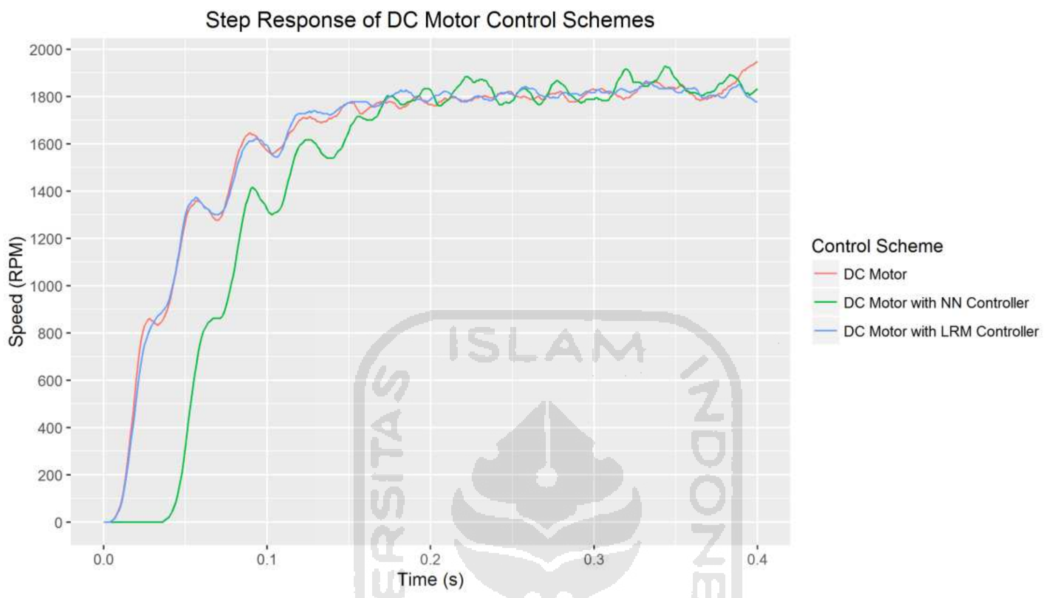


Figure 4.1. Step response graph

As can be seen from the graphs, none of the systems were able to reach the set point. In the case of the DC motor by itself this is due to the fact that, without a controller converting the set point from units of speed to duty cycle percentage, the duty cycle percentage required to reach the set point of 2000 RPM had to be estimated. Table 4.1 shows us the step response characteristics for each system. This table tells us that the overshoot percentage for each system is 0 percent, supporting what we saw in the graphs. The table also shows us that the performance of both the DC motor with the NN controller and the DC motor with the LRM controller are both quite good with small steady-state errors. The NN controller had slightly better performance than the LRM controller.

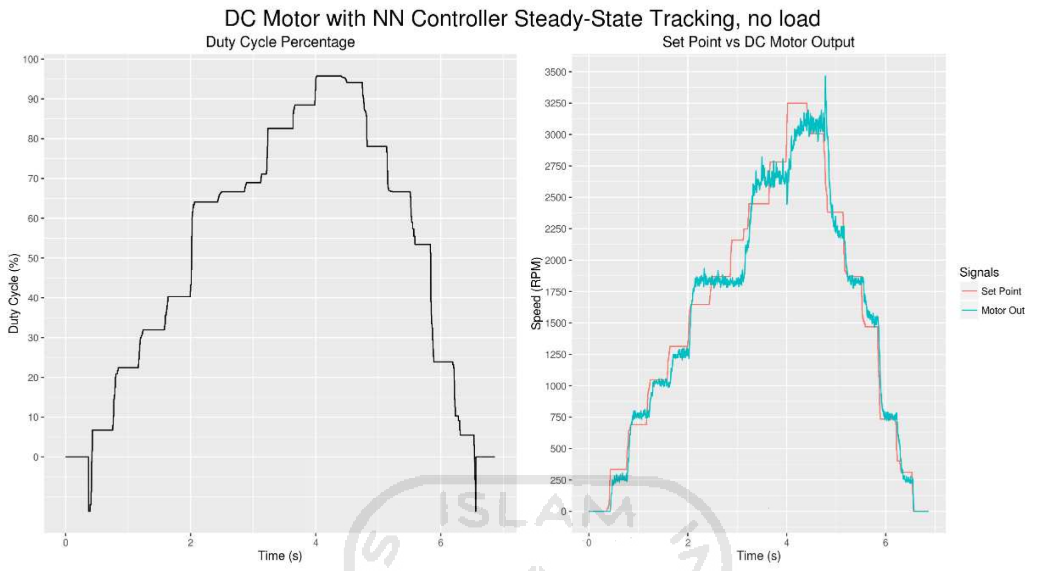
We can see from the table that the rise time for each system was within 0.12-0.16 seconds which tells us that neither the NN controller nor the LRM controller introduced a significantly large delay into the system. However, there is a slight discrepancy between the rise times of the DC motor with the NN controller and the DC motor with the LRM controller. This discrepancy likely stems from the fact that the MATLAB function used to determine the step response characteristics for each system, makes its estimates based on data. Had the data we acquired for the step response of the DC motor with the LRM controller been different it is possible that the rise time of this system would have been closer to that of the other systems. There was also a discrepancy in the settling times of the systems with the NN controller system having a fairly long settling time compared to the plain DC motor and LRM controller systems. The reasoning behind the rise time discrepancies applies here as well.

Table 4.1. Step response characteristics for each system

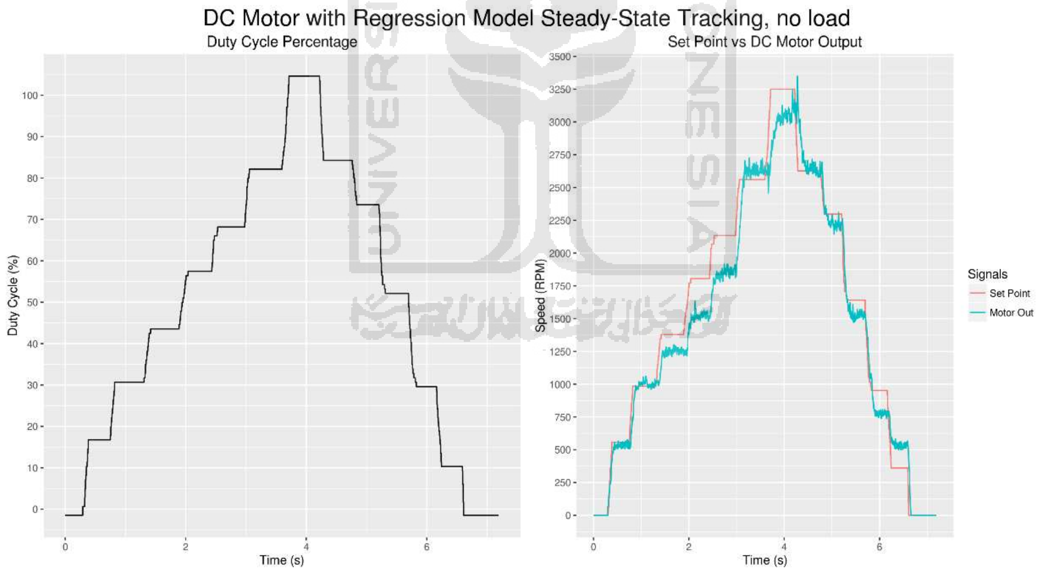
System	**Rise time (s)**	**Overshoot (%)**	**Settling Time (s)**	**Steady-state error (%)**
DC motor	0.1206	0	0.146	9.17
DC motor with NN controller	0.1253	0	0.169	7.74
DC motor with LRM controller	0.1610	0	0.115	9.59

4.2 Steady State Tracking with no Load

During this test, the load on the DC motor was set to 0 percent while the motor was running. With the control system active, the set point of the motor was varied from the control VI and the resulting motor output was captured. Figure 4.2 shows the results of this test.



(a)



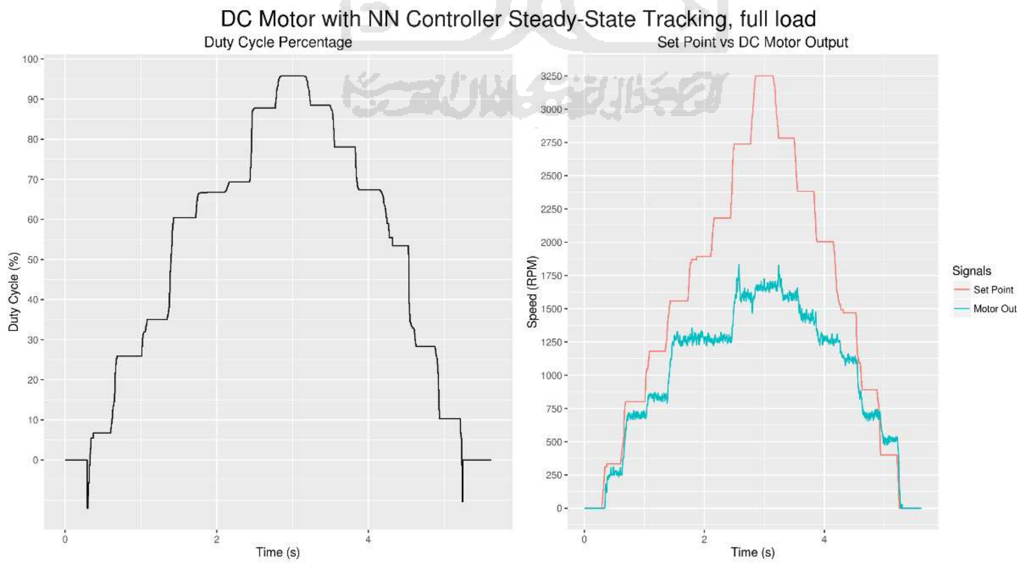
(b)

Figure 4.2. Graph of steady-state tracking of DC motor with no load: (a) DC motor with NN controller, (b) DC motor with LRM controller

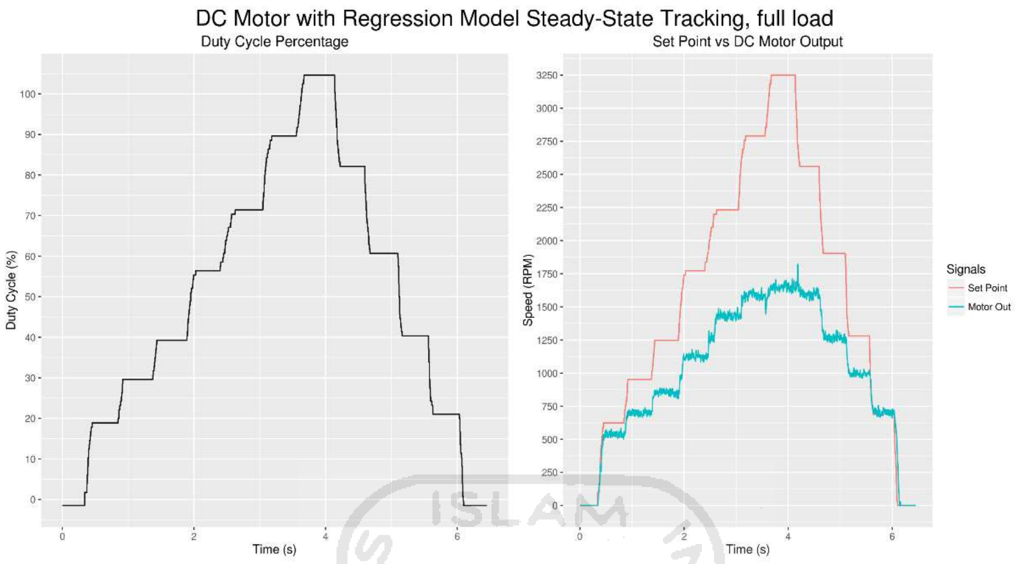
Comparing the graph of the set point to the graph of the control signal for both control systems, we see that the curves mirror each other. Significant changes in the values seem to occur at about the same time. This tells us that there were no major issues regarding delay between the control VI and the DC motor that could affect the performance of either controller. The graph comparing the set point to the motor speed for both control systems shows us that at steady state, the speed of the DC motor stays close to the set point. In other words, the steady state error is small. This graph also seems to show that the NN controller gave slightly better performance than the LRM controller as the DC motor output with the NN controller remained much closer to each set point value compared to the DC motor output with the LRM controller.

4.3 Steady State Tracking with Full Load

During this test, the load on the DC motor was set to 100 percent while the motor was running. With the control system active, the set point of the motor was varied from the control VI and the resulting motor output was captured. Figure 4.3 shows the results of this test.



(a)



(b)

Figure 4.3. Graph of steady-state tracking of DC motor with full load: (a) DC motor with NN controller, (b) DC motor with LRM controller

A comparison of the graph of the set point and the graph of the control signal for both control systems shows that, again, the curves mirror each other. So, as with the previous test, we do not need to worry about problems regarding delay between the control VI and the DC motor that could affect the performance of either controller. However, this time the graph comparing the set point and the DC motor speed for both control systems shows that there is significant steady state error particularly for the set points above 1000 RPM. Since the NN controller is just an inverse model of the DC motor it is not able to adjust the speed of the DC motor in order to account for the load. The reasoning is similar for the degraded performance of the LRM controller. Being a linear model, it cannot account for any of the more complex behavior displayed by the DC motor when the load is set to 100 percent. From this graph, we see that the performance of both controllers is equally terrible.

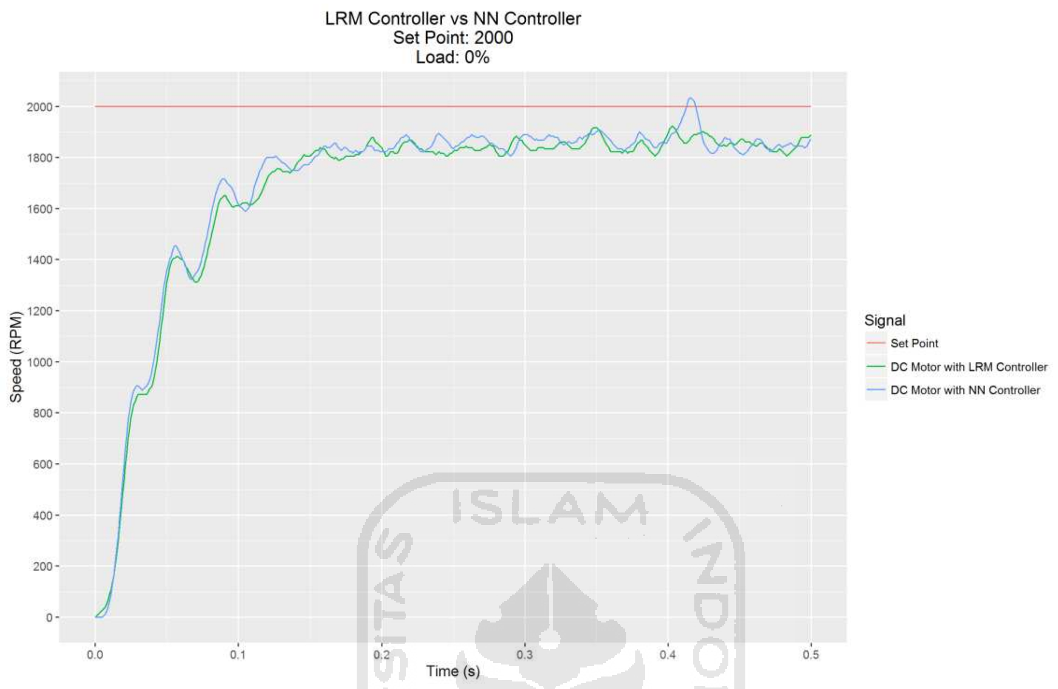
4.4 Steady State with Variable Load

For this test, the set point for the DC motor was set to 2000 RPM and the load was varied between 0 percent, 25 percent, 50 percent, 75 percent, and 100 percent. As mentioned in the previous sections the output of the DC motor fluctuates very rapidly about the steady state value. Due to this, we use the average of the DC motor output values when calculating the steady state error for each load percentage.

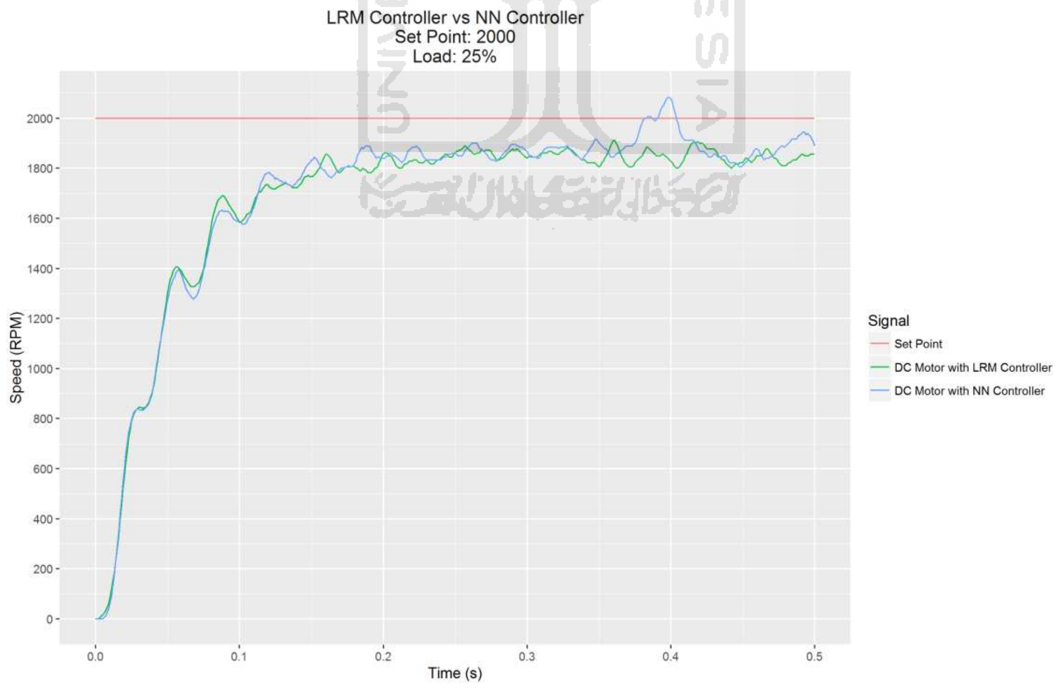
Table 4.2. Load percentages and their associated steady-state errors

Load (%)	**Steady-state Error (%)**	
	NN controller	**LRM controller**
0	7.41	7.48
25	7.07	7.54
50	12.79	12.61
75	24.36	25.26
100	35.43	35.72

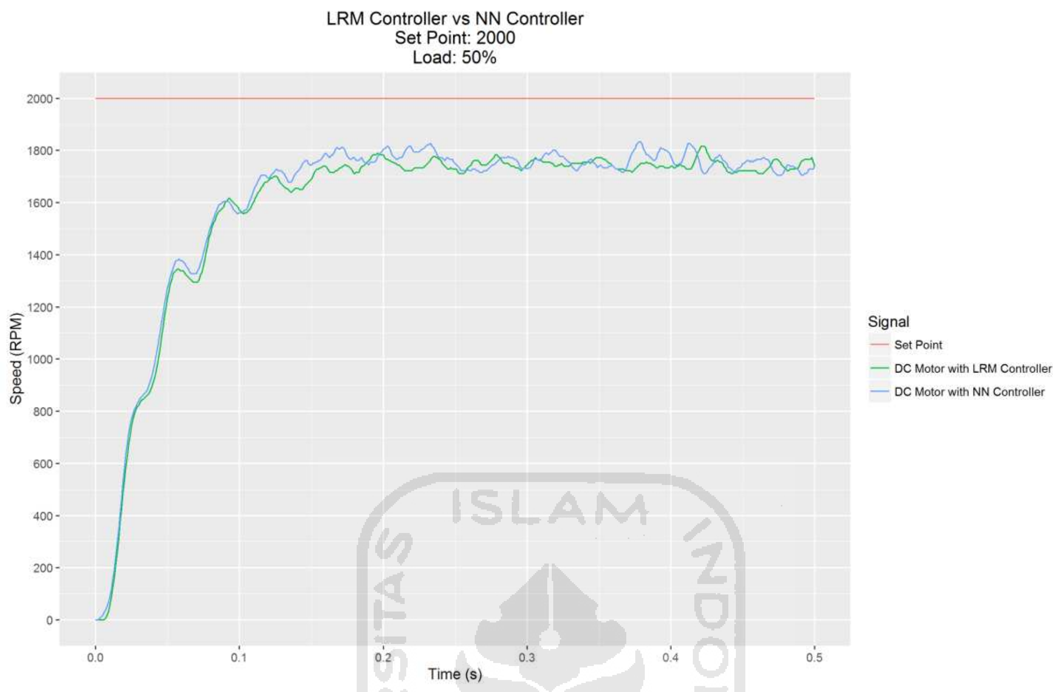
Table 4.3 shows us each load percentage along with the steady state error at that load percentage and Figure 4.4a shows us the graphs of the DC motor output and the set point at each load percentage. From both the table and the graphs it is apparent that the DC motor speed and the load are inversely proportional regardless of the controller meaning the greater the load on the DC motor, the greater the steady state error. This is to be expected as any amount of load on a DC motor reduces its speed. Also, as mentioned previously, due to the nature of both controllers neither of them can compensate for the additional load. From Table 4.3, we see that the NN controller offers marginally better performance than the LRM controller for a greater range of load percentages. This can be attributed to the fact that NNs are far better at detecting patterns and structure within data than LRMs. Thus, the NN controller was able to create a slightly better model of the DC motor than the LRM.


LRM Controller vs NN Controller
Set Point: 2000
Load: 0%
Speed (RPM)
Time (s)
Signal
Set Point
DC Motor with LRM Controller
DC Motor with NN Controller

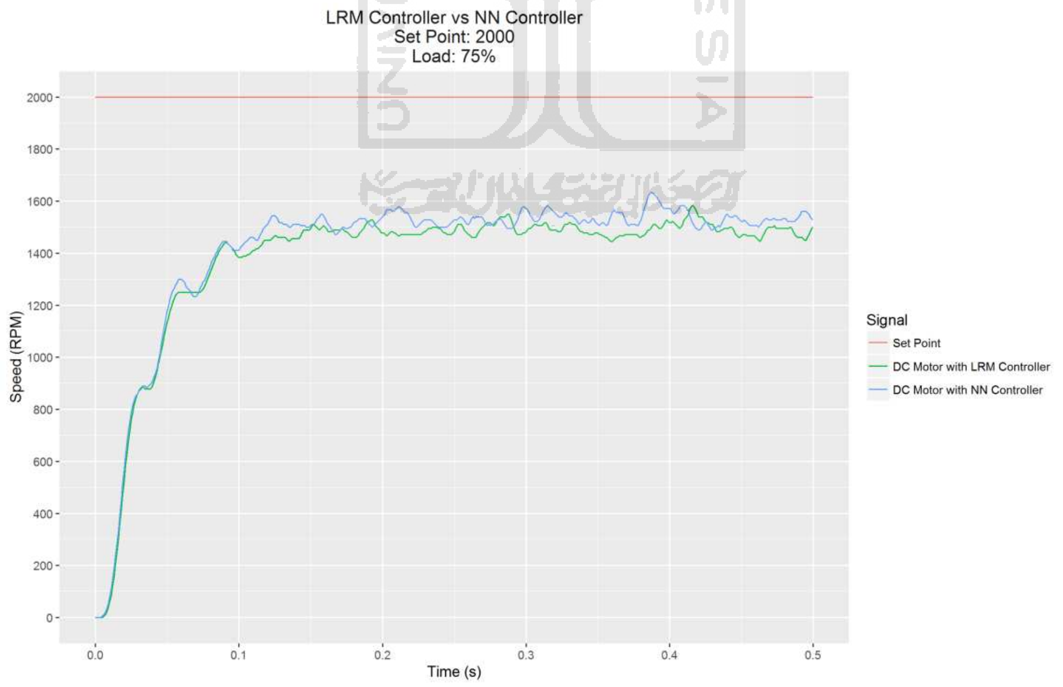
(a)


LRM Controller vs NN Controller
Set Point: 2000
Load: 25%
Speed (RPM)
Time (s)
Signal
Set Point
DC Motor with LRM Controller
DC Motor with NN Controller

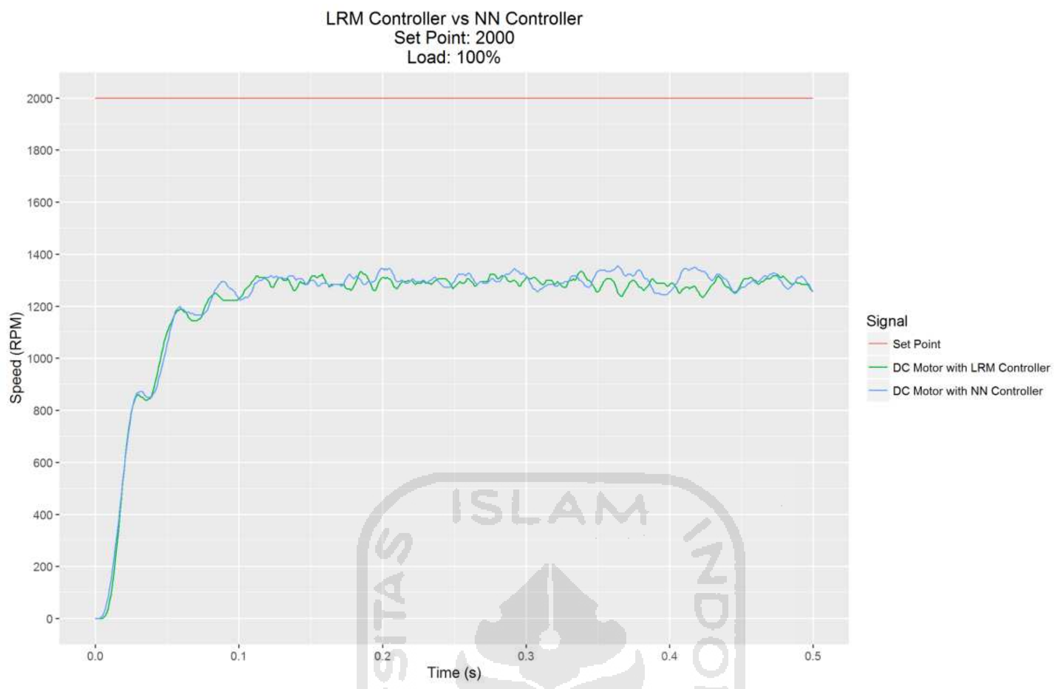
(b)


LRM Controller vs NN Controller
Set Point: 2000
Load: 50%
Speed (RPM)
Time (s)
Signal
Set Point
DC Motor with LRM Controller
DC Motor with NN Controller

(c)


LRM Controller vs NN Controller
Set Point: 2000
Load: 75%
Speed (RPM)
Time (s)
Signal
Set Point
DC Motor with LRM Controller
DC Motor with NN Controller

(d)



(e)

Figure 4.4. Steady-state at different load percentages: (a) 0% load, (b) 25% load, (c) 50% load, (d) 75% load, (e) 100% load.

Chapter 5
Closing

5.1 Conclusion

Several conclusions can be drawn from this research:

1. A NN trained as an inverse model of a DC motor is capable of acting as a controller for said DC motor in a feed forward control system. With no load, the NN gives good performance with minimal steady state error. Even when compared to the performance of a LRM controller, the performance of the NN controller was slightly better.
2. When the load of the DC motor is set to 100 percent, the performance of the NN drops drastically. The same is true of the LRM controller, whose performance was about the same as the NN controller in this regard. For set point values at approximately 1000 RPM and above the steady state error is enormous. Since a DC motor controller should allow the motor to maintain a consistent speed regardless of the load on the motor, an NN trained as an inverse DC motor model is not suitable for controlling a DC motor at full load.
3. The performance of the NN controller becomes progressively worse the larger the load on the DC motor is. At 0 percent load, the steady state error was calculated to be 8.14%. At 25 percent load, the steady state error was 7.35%. At 50 percent load, the steady state error was 13.05%. At 75 percent, it was 24.80% and at 100 percent it was 35.65%. Thus, the NN controller developed for this research project is not suitable for controlling DC motors with large loads although it is quite able to control DC motors with relatively small loads.
4. For each of the tests performed in this research project, the NN displayed better performance than the LRM albeit by a small amount. This demonstrates the effectiveness of NNs as controllers.

5.2 Suggestions for Future Research

The research done here has shown the viability of NNs as DC motor controllers in a feed forward system. From here the logical next step is to apply NNs to DC motor closed loop control. It is also possible to continue this research by improving either the NN controller, the control scheme, or both in order to yield a feed forward NN control system with better performance than the one developed for this project.

BIBLIOGRAPHY

[1] Samarasinghe, Sandhya. *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*. Boca Raton, FL: Auerbach, 2007

[2] Lantz, Brett. *Machine Learning With R*. Birmingham: Packt Publishing, 2015.

[3] Hagan, Martin T., Howard B. Demuth, Mark H. Beale, and Orlando D. Jesus. *Neural Network Design*. 2nd ed. Boston: PWS Pub., 2014.

[4] Baruch, Ieroham et al. "AN ADAPTIVE NEURAL CONTROL SYSTEM OF A DC MOTOR DRIVE.". *IFAC Proceedings Volumes* 35.1 (2002): 277-282.

[5] Zouari, Farouk. "Adaptive Internal Model Control Of A DC Motor Drive System Using Dynamic Neural Network". *JSEA* 05.03 (2012): 168-189.

[6] Karadeniz, Mehmet, Ires Inskender, and Selma Yuncu. "Adaptive Neural Network Control Of A DC Motor".

[7] Mishra, Manish. "Speed Control Of DC Motor Using Novel Neural Network Configuration". Undergraduate. National Institute of Technology, 2008.

[8] Horng, J. "Neural Adaptive Tracking Control Of A DC Motor". *Information Sciences* 118.1-4 (1999): 1-13.

[9] Fallahi, Mohsen and Sasan Azadi. "Adaptive Control Of A DC Motor Using Neural Network Sliding Mode Control". *International Multiconference Of Engineers And Computer Scientists*. 2009.

[10] Hussein, Ahmed, Kotaro Hirasawa, and Jinglu Hu. "Online Identification And Control Of A PV-Supplied DC Motor Using Universal Learning Networks". *European Symposium On Artificial Neural Networks*. 2003. Print.

[11] Raviprasad, Widanalge. "Artificial Neural Network Based Adaptive Controller For DC Motors". Graduate. National University of Singapore, 2003.

[12] Hedjar, Ramadane. "Adaptive Neural Network Model Predictive Control". *International Journal of Innovative Computing, Information and Control* 9.3 (2013): 1245-1257.

[13] Dzung, Phan and Le Phuong. "ANN - Control System DC Motor".

[14] George, Moleykutty. "Speed Control Of Separately Excited DC Motor". *American Journal of Applied Sciences* 5.3 (2008): 227-233.

[15] A. Bature, Amir et al. "SENSORLESS POSITION CONTROL OF DC MOTOR USING MODEL PREDICTIVE CONTROLLER". *Jurnal Teknologi* 77.12 (2015): n. pag.

[16] Rashad, Lina. "Speed Control Of Permanent Magnet DC Motor Using Neural Network Control". *Engineering and Technology Journal* 28.19 (2010): 5844-5856.

[17] SONG, Ying, Zengqiang CHEN, and Zhuzhi YUAN. "Neural Network Nonlinear Predictive Control Based On Tent-Map Chaos Optimization". *Chinese Journal of Chemical Engineering* 15.4 (2007): 539-544.

[18] Yen, Chen-Wen and Mark Nagurka. "Design Of Predictive Controllers By Dynamic Programming And Neural Networks". *American Control Conference*. 2003.

[19] Mendes, Jerome, Nuno Sousa, and Rui Araujo. "Adaptive Predictive Control With Recurrent Fuzzy Neural Network For Industrial Processes".

[20] Brandstetter, Pavel. "Sensorless Control Of DC Drive Using Artificial Neural Network". *Acta Polytechnica Hungarica* 11.10 (2014): n. pag.