Sudan University of Science and Technology

College of Graduate Studies

AUTOMATIC TUNING OF PARAMETERS CONTROLLER USING ARTIFICIAL NEURAL NETWORKS

التوليف التلقائي لمؤشرات المتحكم باستخدام الشبكات العصبية الاصطناعية

A Thesis Submitted in Partial Fulfilment to the Requirements for the Degree of M.Sc. in Electrical Engineering (Microprocessor and Control)

Prepared by: Hajer Khadir Ahmed Almubark

Supervised by: Dr. Awadalla Taifour Ali Ismail

February 2018

قَالَ تَعَالَى:

﴿ يَرْفَعِ ٱللَّهُ ٱلَّذِينَ ءَامَنُواْ مِنكُمْ وَٱلَّذِينَ أُوتُواْ ٱلْعِلْمَ دَرَجَٰتٍ ۚ وَٱللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ ﴾

المجادلة: ١١

DEDICATIONS

This thesis is dedicated to:

The sake of Allah, my Creator and my Master,

My great parents, who never stop giving of themselves in countless ways,

My beloved family and those people who have guided and inspired me throughout my journey of education.

ACKNOWLEDGEMENT

Firstly, I praise God who aids me to complete this research in this way, it gives me great pleasure in expressing my sincere gratitude to everyone who have supported and contributed into making this thesis possible.

I would like to acknowledge my direct supervisor Dr. Awadalla Taifour Ali for his enthusiasm, inspiration and huge efforts to explain things clearly and simply. I would like to thank the Sudan University of Science and Technology for accepting me in its graduate program and motivated me to do this work. I can't end without thanking my all friends.

I would like to thank all the people that supported me through my academic way.

ABSTRACT

proportional, integral and derivative (PID) controllers have become the most popular control strategy in industrial processes due to the versatility and tunning capabilities. The incorporation of auto-tunning tools have increased the use of this kind of controllers. The PID-controllers are often badly tuned, since it is too time consuming to calculate good PID-parameters at the time of deployment.

This work investigates the applicability of artificial neural networks to control systems. The main properties of neural networks are identified as of major interest to this field: their ability to implement nonlinear mappings, their massively parallel structure and their capacity to adapt.

This study suggests a certain technique to apply neural networks for the tuning of the PID controller's gains in a way human tune the gains depending on the environmental and systems requirements. Error Back-Propagation (BP) method is used as the tuning method for the controller which is also known as BP method and this method works on the local minima algorithm.

مستخلص

أصبحت أجهزة التحكم التناسبية التفاضلية التكاملية أكثر استراتيجية التحكم شيوعاً في العمليات الصناعية بسبب تعدد القدرات وإمكانية توليف المعاملات . وقد أدى دمج أدوات التوليف الى زيادة إستخدام هذا النوع من وحدات التحكم بصورة عامة فان توليف معاملات المتحكم التناسبى التكاملى التفاضلى يتم بصورة سيئة خاصة اذا كان التوليف يدويا نظرا لأنه يستهلك وقتا طويلا لحساب هذه المعاملات.

هذا العمل يبحث في امكانية تطبيق الشبكات العصبية الاصطناعية فى أنظمة التحكم . الخاصية الأساسية فى الشبكات العصبية الاصطناعية هي قدرتها على تنفيذ عمليات غير خطية والهيكلة المتوازية على نطاق واسع وكذلك قدرتها على التكيف.

تقترح هذه الدراسة تقنية معينة لتطبيق الشبكات العصبية لضبط وتوليف معاملات المتحكم التناسبى التفاضلى التكاملى إعتمادا على متطلبات النظام وتم إستخدام طريقة خطأ الإنتشار العكسي وتعمل هذه الطريقة على خوارزمية الحد الأدنى المحلي.

TABLE OF CONTENTES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
BL	Boltzmann learning
BPNN	Back Propagation Neural Network
CL	Competitive Learning
CVP	Cross Validation Pruning
DC	Direct Current
ECL	Error Correlation Learning
HL	Hibbing Learning
ICP	Information Criterion Pruning
MLP	Multi-Layer Perceptron
PID	Proportional- Integral-Derivative
RBFN	Radial Basis Function Network
RNN	Recurrent Neural Networks

LIST OF NOMENCLATURES

K_P	Proportional Part gain of PID controller
K_i	Integral Part gain of PID controller
K_d	Derivative Part gain of PID controller
J	Moment of inertia of the rotor
b	Motor viscous friction constant
Ke	Electromotive force constant
Kt	Motor torque constant
R	Electric resistance
L	Electric inductance
θ	Angular velocity
$\varphi(.)$	Activation function.
ε	Error value
$\hat{L}$	Network output.
η	The learning rate.
α	Momentum factor
t_s	Settling time
τ	Time constant

CHAPTER ONE

INTRODUCTION

CHAPTER ONE

INTRODUCTION

1.1 General Review

PID controllers are widely used in industrial processes, and can be implemented in different ways: as a stand-alone regulator or as a distributed component of a control system. Systems with slow dynamics and few performance requirements, as most industrial processes, can be easily controlled using a PID strategy. The incorporation of microprocessors in control systems has modified the meaning of controller's operational characteristics as well as its algorithms. This fact has made possible self-diagnosis and auto tuning.

PID controllers are often implemented with poor tuning, that deteriorates the system performance, in order to let the controlled system work under different conditions [1]. As known from the literature, the auto tuning procedure is performed on the demand of the user, or colloquially after a 'button push'. Thus, it is not performed continuously in the adaptation loop, but rather when the need for tuning or re-tuning arises. This technique reiterates the design steps which the control engineer performs during the design of the controller. Firstly, a simple experiment is performed which determines some characteristics of the process. After that, using the data obtained, the controller parameters are calculated, and the designed controller is started. Such a feature of modern controllers is particularly useful during commissioning of control systems. Besides, auto tuning can also be used for the build-up of the table of controller parameters for gain scheduling [2].

Several robust and auto tuning techniques have been proposed in order to further improve the control and robust performance of the PID controller, recently, PID Neural Network (PIDNN) controller is one of the popular methods used for control complexes systems. The Ziegler and Nichols [1] methods are the most common PID tuning procedures. These methods are very simple and require few information of the system.

1.2 Problem Statement

PID controller model structure needs to be very precise. But in practical applications,

to different extent, most of the industrial processes exist to be nonlinear, the variability of parameters and the uncertainty of model are very high, thus using conventional PID control the precise control of the process cannot be achieved. The common methods known for tuning require the process model to be of a certain type, for example as in the case of a 'First order plus dead time' model. These methods require the process model to be reduced if it's too complicated originally. The above problems can be well addressed by the application of soft-computing methods for tuning of the PID controller. These are especially useful for solving problems of computationally complicated and mathematically in traceable. This is due to the convenience of combining natural systems with intelligent machines effectively with the help of soft-computing methods. Among these entire soft-computing methods available neural network, fuzzy logic and genetic algorithm are the most important ones.

By the implementation of the knowledge of Artificial neural network in PID controller, the system response of the plant can be improved. The overshoot and the rise time of the response can be decreased and the dynamic performance of the system can also be improved.

1.3 Objectives

The main objective of the research reported in this thesis is to study the effectiveness of knowledge based adaptive control with particular emphasis on servo motor control. ANN is used for expressing the knowledge base-adaptation in the controller. The developed techniques is tested and experimented. These experimental results are compared with traditional control techniques, using software and hardware.

1.4 Methodology

This research we are proposes a neural network controller design to control a DC motor.

- The training algorithm is used in the ANN is the back-propagation method.

- Two feed-forward neural networks are used, the first neural network is called the Model Network; the function of this network will be the same function as the DC motor.
- The second network used is called the PID neural network controller, this network has the same function as a PID tuned controller, but the difference is that this network capable of updating itself in a manner to improve the controller function this is why it is considered to be a smart controller.

1.5 Research Layout

In the following chapter we are going to discuss more about the literature review in **c**hapter Two, the methodology in chapter Three, result and analysis of the system in chapter Four, and final chapter Five is the conclusion and recommendations.

CHAPTER TWO

THEORITICAL BACKGROUND

CHAPTER TWO
THEORETICAL BACKGROUND

2.1 Introduction

From the very beginning, it has been realized by systems theorists that most real world dynamical systems are nonlinear. However, linearization of such systems around the equilibrium states yields linear models, which are mathematically obedient. In particular, based on the superposition principle, the output of the system can be computed for any arbitrary input, and alternately, in control problems, the input, which optimizes the output in some sense, can also be determined with relative ease. In most of the adaptive control problems, where the plant parameters are assumed to be unknown, the fact that the latter occur linearly makes the estimation procedure straightforward. The fact that most nonlinear systems thus far could be approximated satisfactorily by linear models in their normal ranges of operation has made them attractive in practical contexts as well. It is this combined effect of ease of analysis and practical applicability that accounts for the great success of linear models and has made them the subject of intensive study for over four decades. In recent years, a rapidly advancing technology and a competitive market have required systems to operate in many cases in regions in the state space where linear approximations are no longer satisfactory. To cope with such nonlinear problems, research has been underway on their identification and control using artificial neural networks based entirely on measured inputs and outputs.

From the beginning of systematic automatic controller design there has been the problem of finding a proper controller structure and the controller parameters for a given process. The main difficulty that comes into sight is the need of the controller to be very well tuned for the whole range of its operating points rather than for one particular operating point. To overcome these circumstances, adaptive controllers were developed in the nineteen forties. Between nineteen sixties and nineteen seventies many fundamental areas in control theory were developed which later proved to be significant for the design of adaptive control systems, e.g. state space and stability theory.

2.2 DC Motor Modeling

A common actuator in control systems is the Direct Current (DC) motor. It directly provides rotary motion and, coupled with wheels or drums and cables, can provide translational motion. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following Figure 2.1 [3].

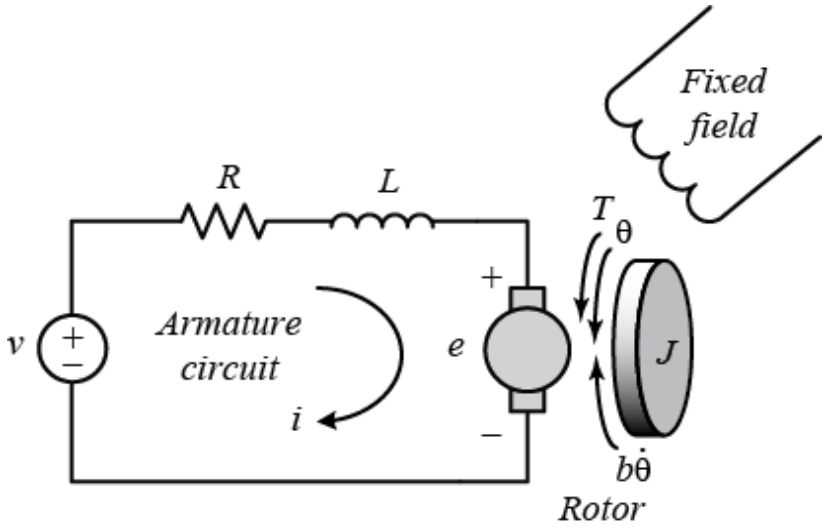


Figure 2.1: Structure of DC motor circuit

For this example, will be assumed that the input of the system is the voltage source (V) applied to the motor's armature, while the output is the rotational speed of the shaft d(theta)/dt. The rotor and shaft are assumed to be rigid. Will be further assumed a viscous friction model, that is, the friction torque is proportional to shaft angular velocity.

The physical parameters for our example are:

(J) Moment of inertia of the rotor 0.01kg.m^2

(b) Motor viscous friction constant 0.1N.m.s

(K_e) electromotive force constant 0.01V/rad/sec

(K_t) motor torque constant 0.01N.m/Amp

(R) Electric resistance 1Ω

(L) Electric inductance 0.5H

In general, the torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field. In this example we will assume that the

magnetic field is constant and, therefore, that the motor torque is proportional to only the armature current i by a constant factor K_t as shown in the equation below. This is referred to as an armature-controlled motor.

$$T = k_t i \tag{2.1}$$

The back emf is proportional to the angular velocity of the shaft by a constant factor K_e.

$$e = k_e \theta' \tag{2.2}$$

In SI units, the motor torque and back emf constants are equal, that is, $K_t = K_e$; therefore, we will use K to represent both the motor torque constant and the back emf constant[3]. From the Figure 2.1, it can derive the following governing equations based on Newton's 2nd law and Kirchhoff's voltage law, we have

$$J\ddot{\theta} + b\dot{\theta} = k\, i \tag{2.3}$$

$$L\frac{di}{dt} + R\, i = V - k\, \dot{\theta} \tag{2.4}$$

2.2.1. S-domain model

Applying the Laplace transform, the above modeling equations can be expressed in terms of the Laplace variable s, as follows:

$$s(Js + b)\theta(s) = K\, I(s) \tag{2.5}$$

$$(Ls + R)I(s) = K\, I(s) \tag{2.6}$$

We arrive at the following open-loop transfer function by eliminating I(s) between the two above equations, where the rotational speed is considered the output and the armature voltage is considered the input.

$$P(s) = \frac{\dot{\theta}(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2} \tag{2.7}$$

2.2.2 State-space model

In state-space form, the governing equations above can be expressed by choosing the rotational speed and electric current as the state variables. Again the armature voltage is treated as the input and the rotational speed is chosen as the output.

$$\frac{dy}{dt}\begin{bmatrix}\dot{\theta} \\ i\end{bmatrix} = \begin{bmatrix}\frac{-b}{J} & \frac{K}{J} \\ \frac{-K}{L} & \frac{-R}{L}\end{bmatrix}\begin{bmatrix}\dot{\theta} \\ i\end{bmatrix} + \begin{bmatrix}0 \\ \frac{1}{L}\end{bmatrix}V \tag{2.8}$$

$$y = [1 \quad 0]\begin{bmatrix}\dot{\theta} \\ i\end{bmatrix} \tag{2.9}$$

2.3 Proportional Integral Derivative Controllers

Other than the Artificial Neural Network (ANN) controller two types of conventional feedback controllers are used in the present study. One is a Proportional-Integral (PI) controller and the other is PID controller. Both these servo controllers are used for comparison purposes with the ANN based controller. At implementation the controllers were built using a host-target prototyping environment with a compatible data acquisition board. In this study a Permanent Magnet (PM) DC motor is adopted as the plant. The idealized equation of a proportional-integral (PI) controller is

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(t)dt\right] \tag{2.10}$$

in which K is the gain, T_i is the integral time and e(t) is the feedback error; i.e., e(t) =r(t) – y(t). Where r (t) and y (t) are reference input and the plant output respectively The equivalent transfer function in the s-domain is given by

$$U(s) = \left[K\left(1 + \frac{1}{sT_i}\right)\right]E(s) \tag{2.11}$$

For digital control, Equation (2.11), is transformed into its discrete-time (z-domain) equivalent, as given by

$$U(z) = \left[K_p\left(1 + \frac{K_i}{1 - Z^{-1}}\right)\right]E(s) \tag{2.12}$$

Or, in velocity form,

$$U(z) = -K_p Y(z) + K_i \frac{E(z)}{1 - Z^{-1}} \tag{2.13}$$

where

$$K_p = K - \frac{KT_s}{2T_i} \tag{2.14}$$

$$K_i = \frac{KT_s}{T_i} \tag{2.15}$$

where T_s is the sampling interval.

The Proportional, Integral, Derivative controller (or the PID controller) is the most popular type of controller used in different engineering applications. The PID controller is a form of control loop that has a feedback mechanism. The PID controller works by calculating the error signal between an output measured value

and a reference value, the controller works to minimize the error signal or the difference between the output signal and the reference signal to a minimum value; such that the output measured value will be as close as possible to the input reference signal [4].

The mathematical representation of the PID controller is:

$$U(t) = Kp.e(t) + Ki\int_0^t e(\tau)d\tau + Kd\frac{d}{dt}e(t) \quad (2.16)$$

Where U(t) is the controller output signal, e(t) is the error signal, K_p is the proportional gain, K_i is the integral gain and K_d is the derivative gain.

As shown in Equation (2.16), the PID controller has three parameters, P or Proportional term, I or Integral term and D or Derivative term, each one of these terms has a gain value related to it, and it makes the controller system to react in a different way from the others. The proportional term depends on the present error value, the proportional gain have a direct relationship to the controller sensitivity, the higher P gain value leads to faster change for the systems' output, which makes the controller to be more sensitive.[4]

2.4 Tuning of the PID controller

An auto-tuner is a device that automatically computes the parameters of a controller. The goal is to achieve the best control possible given the tuning objectives. The goal is not to replace a human control engineer. The auto-tuner should rather be seen as an aid to improvement [5]. Many single-input single-output industrial control loops are poorly tuned [6]. Tuning of a PID controller refers to the tuning of its various parameters (P, I and D) to achieve an optimized value of the desired response. The basic requirements of the output will be the stability, desired rise time, peak time and overshoot. Different processes have different requirements of these parameters which can be achieved by meaningful tuning of the PID parameters. If the system can be taken offline, the tuning method involves analysis of the step input response of the system to obtain different PID parameters. But in most of the industrial applications, the system must be online and tuning is achieved manually which requires very experienced personnel and there is always uncertainty due to human error. Another

method of tuning can be Ziegler-Nichols method [4]. While this method is good for online calculations, it involves some trial-and-error which is not very desirable.

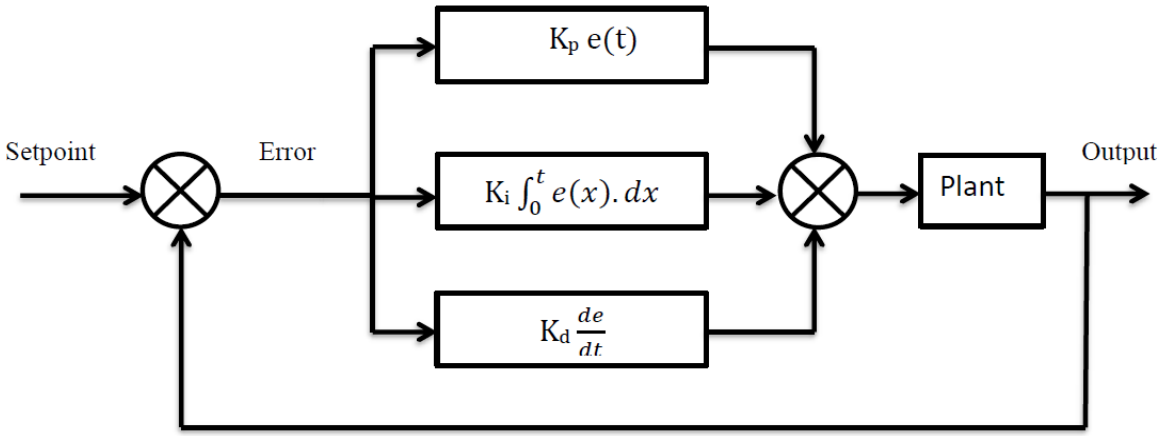


Figure 2.2: Basic block diagram of a conventional PID controller

2.4.1 overview of existing methods of PID

The PID controller, introduced in the last section, is a standard building block form industrial automation. The popularity of this regulator comes from its robust performance in a wide range of operating conditions, and also from its functional simplicity, which makes it suitable for manual tuning. To account for process changes and ageing, regular retuning is normally required. Accurate tuning is an operation which, to be done properly, takes considerable time. Since large plants can have hundreds of PID regulators, methods which automate the tuning of the PID compensators are of great practical importance. A large number of methods for PID auto-tuning have been proposed. In this section some of them will be described. To ease the description these methods will be loosely classified into five classes: frequency response based, step response based, on-line parameter estimation based, expert and fuzzy systems based, and neural networks based [7].

2.4.2 Methods based on expert and fuzzy logic systems

Expert systems and systems fuzzy logic systems have also been proposed for PID auto-tuning. An example of the first class can be found in Anderson et al. Here they propose an iterative rule-based method which analyses the response of the closed-loop system to a step change in the reference. Based on previous

experience and knowledge, a new set of PID values is then chosen. For each set-point disturbance, features of the output like overshoot, settling time, rise time, etc. are computed and compared with desired values. For each criterion not met, a rule is fired. Each rule computes the percentage of the change for each PID parameter, using a measure of the degree to which the criterion has not been satisfied, and a weight associated with that criterion. These weights, one for each parameter and for each criterion, are obtained from an expert's experience.

These percentages of change are accumulated over the criteria not met, to compute the total adjustment for each PID parameter. The PID parameters are then modified accordingly [7]. Lemke and De-zhao introduced a fuzzy PID supervisor to adjust the settings of a PID controller. The error and its derivative, scaled by the value of the reference, are the input variables of the fuzzy supervisor. Different fuzzy regions are specified, in which the inputs are distinguished. For each fuzzy region, fuzzy rules and conditional statements are formulated, according to expert experience. After implementation of these rules the resulting fuzzy outputs are transformed into deterministic values, using a defuzzification rule. These values are the changes to be applied to the PID values.

2.4.3 Methods based on artificial neural networks

To the best of our knowledge, apart from our own approach to PID auto-tuning, to be introduced in the following section, only two other PID auto-tuning techniques involving artificial neural networks have been proposed. The first, in a chronological order, was introduced by Swiniarski. In this approach the open loop step response of a plant is discredited, its samples being used as inputs to a multilayer perception. The role of the neural network is, based on these inputs, to determine the corresponding PID parameters. The Multi-layer perception (MLP) has therefore three outputs, corresponding to the three PID parameters. The open loop Ziegler-Nichols tuning rule is employed to obtain the PID parameters to be used as targets for training. The training set is derived by varying the parameters L, T and K_p, as shown in Figure. 2.7, within a specified range around nominal values. These ranges are discredited to obtain suitable examples for training. In the example proposed, the nominal values were K_p=1, L=0.2 and T=1, the range of change for the three parameters being.

Unfortunately, no results of this technique are presented in the paper. Some observations can be made concerning this method of PID auto tuning:

i) It is an open loop technique; it cannot be applied in closed loop.

ii) Because samples of the open loop step response are used as inputs to the MLPs, obtaining good results might require a high number of samples. This leads to a large number of parameters in the MLP, resulting in large training times.

iii) This technique is likely to be heavily dependent on the sampling time chosen. Specifically, if we consider an example where the sampling time T is used, it is questionable whether the MLP will produce the same PID values if sampling times of T/2 or 2T are used.

iv) It is known that, in most cases, the responses obtained with Ziegler-Nichols tuning rule are not well damped. It seems sensible to obtain the target PID values using a better criterion.

v) If the Ziegler-Nichols technique is used to derive the target PID values then there is no need to consider three outputs for the MLP, since the integral and derivative time constants are linearly interdependent.

The second approach is due to Light body and Irwin [8]. Their example actually considered a PD controller, but their technique can be extended to PID regulators. In this approach the closed loop step response is discredited, the samples being used as inputs to one MLP. They consider a nominal plant, under PD control. The gains of the compensator are varied over some acceptable range, and a family of step responses obtained. The MLP is then trained to map these responses to the actual PD values which originated them.

2.5 Artificial Neural Networks

Work on artificial neural networks, commonly referred to as “neural networks “has been motivated right from its inception by the recognition that human brain computes in an entirely different way from the conventional digital computer. An artificial neural network, which is the formal name for the term neural networks used here, is one of many attempts to build an intelligent machine or to create artificial intelligence. It is based on biological neural networks. The basic idea to model this is to make a very simplified model of biological neurons and their synapses [9]. The

novelty of ANN theory could undoubtedly be attributed to the experiment performed by “McCulloh” and “Pitts” in modeling bio-systems using nets of simple logical operations back in 1943. The idea was to find a simple parametric nonlinear model for a real neuron. Ever since this innovation, there has been a great interest from various researchers and scientists, thus several ANN-based models in different fields were discovered. The technology though had lost its momentum in the late 1969 till 1986 when the back-propagation of error was discovered. To date, ANN-based models have been successfully implemented in a number of industries ranging from: Aerospace, automotive, defense, electronics, entertainment, financial and so on. Artificial neural Networks have been also successfully applied in medical fields [9].

ANN is a part and parcel of intelligent based systems, designed distinctively to improve the performance of conventional computing techniques. The biggest drawback associated with the so called conventional methods is the inability to learn and identify patterns in dynamic systems. Thus the need to eliminate this shortcoming through learning is proven essential[10]. ANN is an information processing paradigm inspired by the way biological nervous systems, such as the brain, process information. The human brain has 100 billion biological neurons with about 100 000 connections per neuron. A simplified biological neuron is illustrated in Figure 2.3

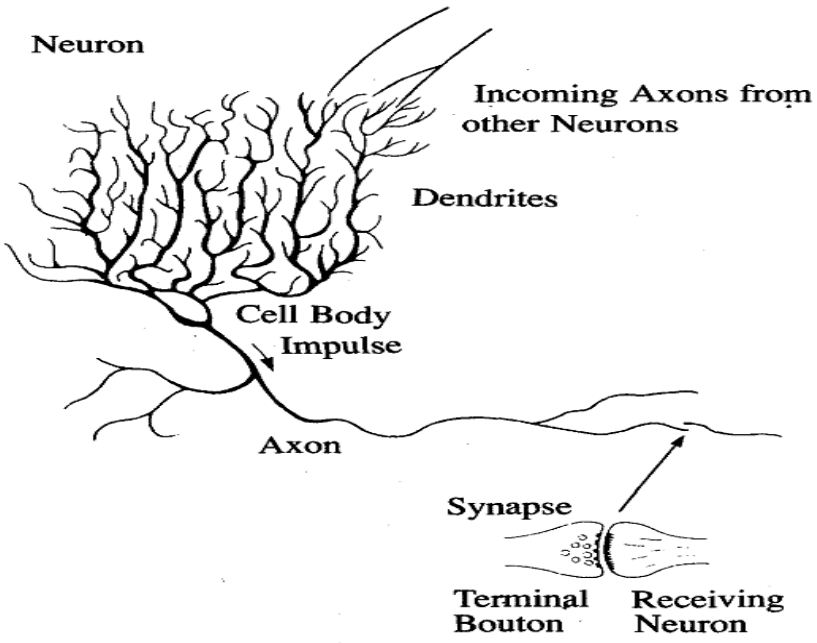


Figure 2.3: A biological neuron

Biological neurons receive spikes through synapses located on the dendrites of the neuron. When the spikes received are strong enough and exceed a definite threshold, the neuron is activated and fires a signal though the axon. This signal travels from the body, down the axon, to the next neuron(s). Learning arises by adjusting the effectiveness of the input (synapses) so that it influences one neuron on other changes. Humans and highly trained animals use the same configuration and summing up to extremely complex networks [9]. Similarly, an artificial network is made up of simple interconnected processing elements called neurons. The neurons are arranged in a layered structure to complete a network competent of executing parallel and distributed computations. Architecture of a simple ANN is shown in Figure 2.8. The attraction of ANN-based models comes with the network's ability to learn, recognize data patterns, and adapt to a changing environment like the human brain. This adaptive characteristic is often called "the human-like reasoning". The architecture illustrated in Figure 2.8, presents a three layered feed-forward network. ANN has a remarkable capability to develop sense from convoluted or imprecise data, extract patterns and detect trends that are too complex often only noticeable by either humans or other computer techniques. In broad terms, ANN-based models offer a variety of benefits namely: adaptive learning, self-organization, real time operation, fault tolerance via redundant information coding. Thus neural network processes information in the similar way the human brain does. The neurons are organized in a way that defines the network structure. The most concerned structure is the MLP type, in which the neurons are organized in layers. The neurons in each layer may share the same inputs, but are not connected to each other. If the architecture is feed-forward, the outputs of one layer are used as the inputs to the following layer. The layers between the input neurons and the output layer are called the hidden layer [10].

2.5.1 Basic architecture of a feed-forward network

The feed-forward network topology illustrated in Figure 2.4 permits signals to travel one way only, from the input through the hidden layer to the output layer. These types of networks are somehow straight forward and associate inputs with outputs.

They are extensively used in pattern recognition. This kind of organization is also referred to as bottom-up or top-down and commonly used in pattern recognition.

Figure 2.4 also shows the commonest type of artificial neural network which consists of two layers. The hidden layer neurons are connected to the output layer neurons. The functions of each layer in the network are defined below:

i) The input layer neurons represent the pre-processed data fed into the network.

ii) The input of each hidden layer neuron is defined by the sum of the input vector set and the connection weights between the input layer and hidden layer.

iii) The input of the output neuron is determined by the weighted sum of outputs of the hidden layer neurons.

iv) The output of a neuron is defined by the type of the transfer function used in that specific layer. This type of network is attractive because the hidden neurons are free to develop their individual representations from the input set.

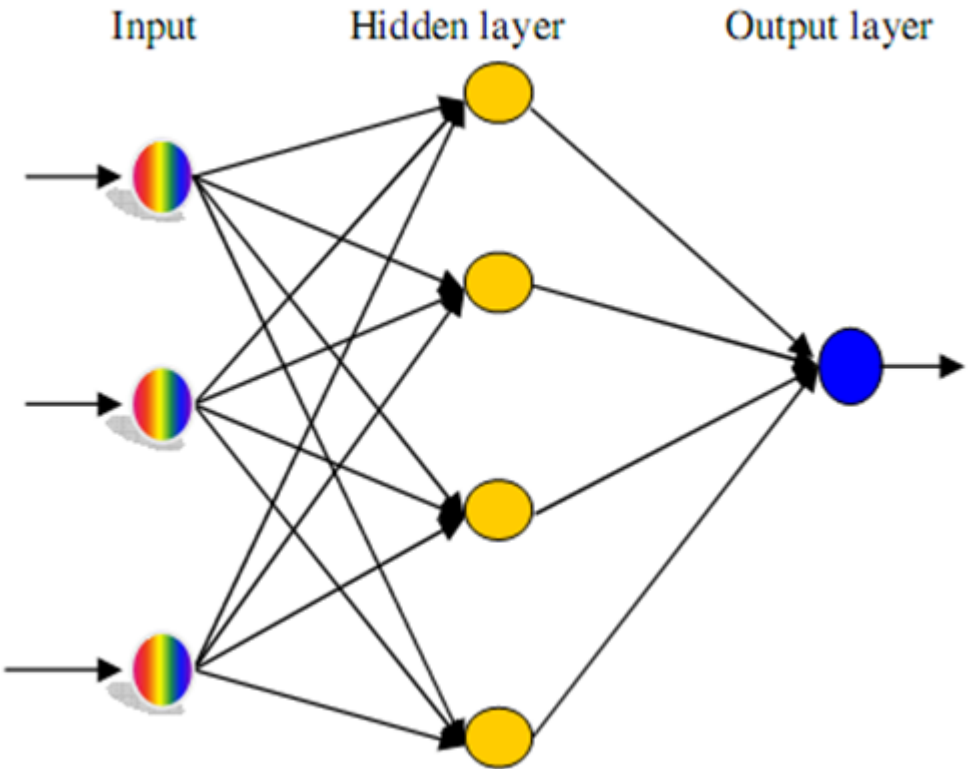


Figure 2.4: Architecture structure of a feed forward neural network

2.5.2 The perceptron – a network for decision making

The perceptron, a basic neuron, invented by Rosenblatt in 1957 at the Cornell Aeronautical Laboratory in an attempt to understand human memory, learning, and

cognitive processes prior to his demonstration on the first machine that could "learn" to recognize and identify optical patterns in the early 1960. The mathematical model of the perceptron or artificial neuron is modeled in the similar manner of the biological architectural set-up. Again, the three major components are considered:
Axons and synapses of the neuron are modeled as inputs and weights respectively.
The strength of the connection between an input and a neuron is denoted by the value of the weight. The mathematical model of this topology is illustrated in Figure 2.5. The weighted inputs are added together and passed through a nonlinear activation transfer function. Finally, the activation function controls the amplitude of the output of the neuron. The suitable scale of output is usually between 0 and 1, or -1 and 1.

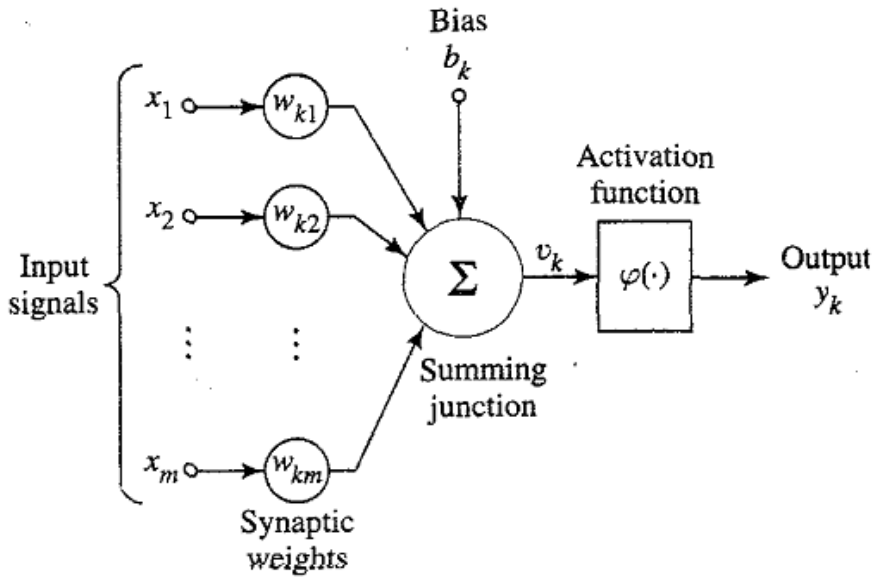


Figure 2.5: A perceptron model

In mathematical terms we may describe a neuron k by writing the following pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.17}$$

and

$$y_k = \varphi(u_k + b_k) \tag{2.18}$$

Where

$x_1, x_2, \ldots, x_m$ are the input signals.

$w_{k1}, w_{k2}, \ldots, w_{km}$ are the synaptic weights of the neuron k.

u_k is the linear combiner output due to input signals.

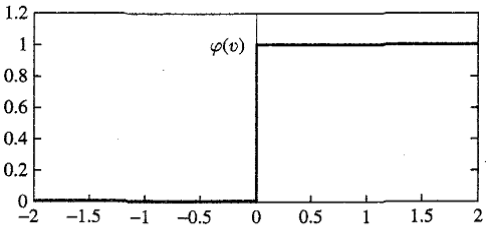
u_k is the bias.

$\varphi(.)$ is the activation function.

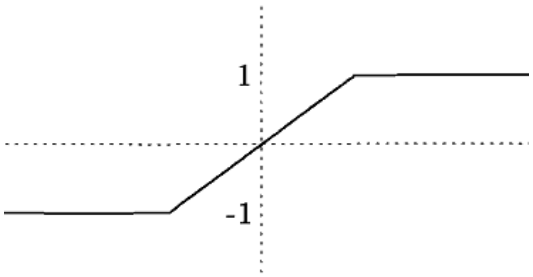
y_k is the output signal of the neuron.

2.5.3 Activation functions

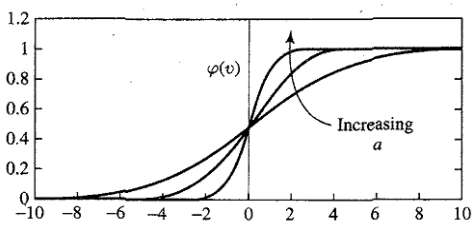
The activation function acts as a squashing function. In simple terms, the network activation rule is denoted as activation function. A number of transfer functions are available for selection. These activation functions include (and not limited to): step function, logistic function, tangent-hyperbolic function, linear function, cosine function and so on. Some of these functions are illustrated in Figure 2.6.



(a) Step function



(b) Function layer



(c) Sigmoid function with varying slop

Figure 2.6

However there is no engineering rule or "rule of thumb" for the selection of transfer function thus the choice is frequently performed arbitrarily. In fact, the effect of different transfer functions on network performance has not been explicitly reviewed in a number of literature papers [11].

2.5.4 Soft computing using neural network topologies

Generally, there are three common types of neural networks: Radial Basis Function Network (RBFN), MLP, and Recurrent Neural Networks (RNNs). The distinction in

different network topologies can perhaps be attributed to the arrangement of neurons and the connection patterns of the layers. A Multi-Layer Perceptron network is the most popular neural network type and the most of the reported neural network in STLF. The inner structure of the processing element (neuron) in each network is interconnected differently, and the configuration set-up is often referred to as network topology. The behavior of the network relies greatly on the network topology [9].

2.5.5 Feed forward Network

In a MLP feed-forward network, connections are unidirectional and no loops are introduced in the network, thus each neuron is linked only to neurons in the next layer, a feed-forward network is referred to as a directed cyclic graph. This implies no backward links either. The importance of loop less networks is that computation can proceed uniformly from input neurons to output neurons. And since there are no backward links, activation functions from the preceding time step play no role in the computation. Figure 2.7 shows a simple multi-layered feed-forward network topology. In Figure 2.5, only one output layer neuron is considered and. In some approximation problems, the number of output neurons equals to the number of outputs. Each layer has a specified number of nodes; the interconnections are only between neurons of adjacent layers, and each neuron belonging to a layer is connected to all the neurons of adjacent layers. Note that ANN may contain more than one hidden layer; the number of neuron in each layer should be carefully selected depending on the application requirements.

2.5.6 Artificial Neural Network Training and Generalization

There are various processes involved in developing a supervised ANN-based model. Amongst others, training process and validation process are some of the vital steps. The input-output patterns are repeatedly presented to the network during the training process. Through this process, the network learns the subjected environment or patterns, and eventually yields the desired output. The desired output is attained as a result of adjusting weights of all interconnections between neurons to establish the correct set of input-output response. However, during the training process, optimal training time is required to avoid overtraining. Overtraining of the network can be

prevented by employing complex stopping criterion. Early stopping (the most common), regularization, pruning, Information Criterion Pruning (ICP), Cross-Validation Pruning (CVP) are some of the stopping methods.

Training the network at infinitum normally results in a reduced error function for a given set of inputs. Though, this does not guarantee better accuracy and robustness of the network because the error could be extremely big if the network is presented with the data it has not seen before. This characteristic is explored during the validation process. Moreover, the validation process also improves the network reliability and generalization.

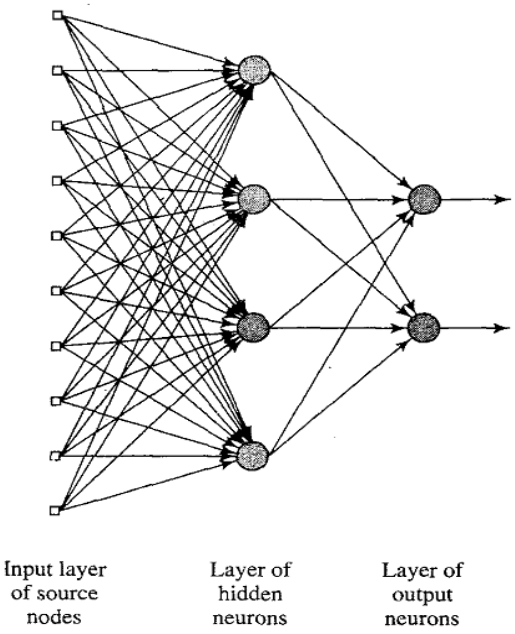


Figure 2.7: Feed-forward network with one hidden layer and two output layer.

2.5.7 Network coupled errors

Broadly speaking, there exist two types of errors during the training process: training error and generalization error. The general error presentation can be seen in Figure 2.8. At the beginning of the training, the training error is at maximum, and gradually decreases as time elapses. Optimum training time should be within the circled area in order to maintain network generalization tendency. In essence, the desired goal should not be an error-free output, as the network will tend to memorize the input-output patterns if trained at infinitum.

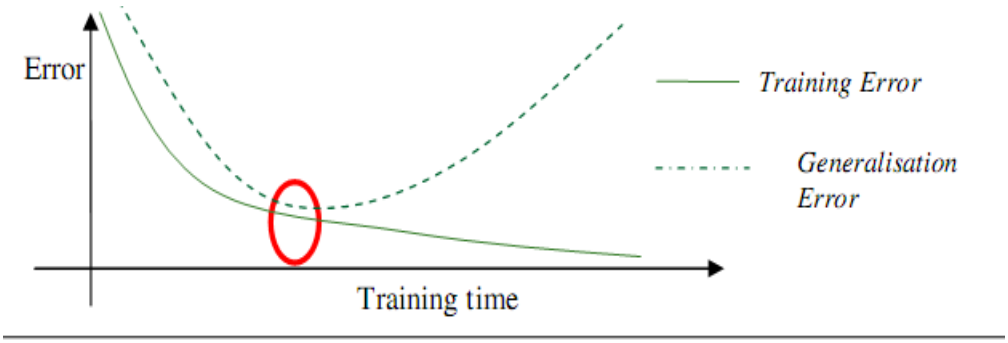


Figure 2.8: Errors versus. optimal network training time

2.6 Artificial Neural Network Learning Paradigms

Like in any other intelligent based systems, a desired output of the network does not come by chance. The network rather adapts itself to a stimulus, and ultimately yields the desired output. The main difference here perhaps could be featured to the type of learning a particular network is subjected to. Broadly speaking, there are different learning paradigms that can be used to train neural networks. Supervised and unsupervised learning are the most common, with reinforced and competitive learning techniques also gaining considerable popularities. The learning process is essential to adjust network weights in order to reduce the error. During the process of adapting and adjusting the synaptic weights, the network acquires knowledge similar to human-like reasoning. However, the learning process requires sets of mathematical algorithms describing how synoptic network weights and biases are attuned.

2.6.1 Supervised learning

This is a type of training approach where the input and the desired output are clearly specified. Suppose, the input vector is represented by [$x_1, \ldots, x_m$] and the corresponding output vector is denoted by $[y_1, \ldots, y_m]$, an optimal rule is to be determined such that:

$$[y_1, \ldots, y_m] = f[x_1, \ldots, x_m] + \varepsilon \qquad (2.19)$$

Where ε represents the approximation error. In this example, the error could be a vector.

The idea is that the network adjusts its weights as an attempt to minimize the approximation error preferably to the smallest value possible, and thereafter the network returns a trial result. This result is then compared to the desired output. Figure 2.9 illustrates a basic network structure for a supervised ANN model. Here the error function is feedback into the system as an attempt to find a correct set of input vectors. If the desired threshold is not attained, another update of the network weight vectors could be initiated. In dynamic systems especially, the need for a network to map the nonlinear relationship between inputs and outputs is enormous, thus this approach is commonly employed in ANN based models. Some of the key measures to evaluate a supervised learning session are: time required per iteration, number of iterations per unit pattern, convergence points i.e. local or global minima(s) …etc [9].

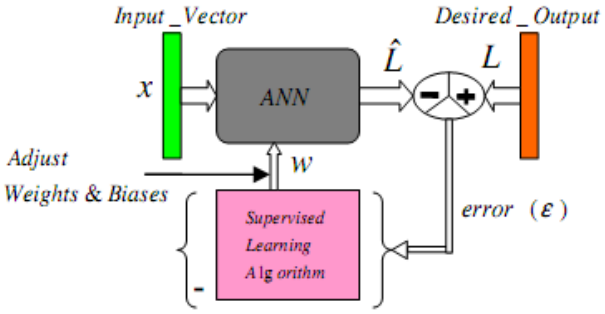


Figure 2.9: Supervised learning paradigm using error correction technique

2.6.2 Competent learning process for ANN

In addition to the ANN learning paradigms discussed earlier in this chapter, here other main issues compulsory for a successful ANN training are exclusively introduced. They are: learning rule, and learning theory. Unlike learning paradigm, learning rule classifies how network weights should be adjusted in the learning trials. There are four basic types of learning rule: Error-Correlation Learning (ECL), Boltzmann Learning (BL), Hibbing Learning (HL) and Competitive Learning (CL) [12].

Most of the supervised learning algorithms use these learning rules during training.

Error-correction learning is the method of comparing the network output to the desired output value, and using that error to guide the training. In the most guiding phases, the error values can be used to directly adjust the tap weights, using an

algorithm such as the back-propagation algorithm. Mathematically, the error used by this method can be formulated as:

$$\varepsilon = L - \hat{L} \tag{2.20}$$

Where, ε is the error value, L is the target output, and $\hat{L}$ denotes the network output.

At each training iteration, error correction learning algorithms attempt to minimize this error function. This method is mostly used in back propagation learning algorithms. The originality of the Hebbian learning rule could perhaps be attributed to a hypothesis proposed by D. Hebb back in the year 1949. The concept is currently adopted as Hebb's law, and is one of the key ideas in biological learning. The Hebb' Law can be presented in the form of two sub-rules:

1. If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.
2. If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

In simple terms, the Hebbian learning rule specifies how much the weight between two neurons should be increased or decreased in proportion to their activation .

Suppose α_i and α_j are the activations of the neurons i and j, and w_{ij} is the synoptic weight between them, then the basic form of the Hebbian rule can be formulated as:

$$\Delta w_{ij} = \eta \alpha_i \alpha_j \tag{2.21}$$

Where, Δw_{ij} is the change is in the synoptic weight, and η is the learning rate.

Another powerful learning rule is the delta rule. This rule utilizes the discrepancy between the desired and actual output of each output unit to change the weights between neurons. Another vital component for competent learning process of neural networks is the learning theory. This concept covers issues related to data quality, data quantity, and best convergence etc. i.e. training data in general. In most of the viewed papers, researchers tend to omit explanations with regards to criterion used for selecting certain variables, training samples, data quality etc. It's obvious there is no general rule in selecting historical data of the phenomena of interest. Thus the input variables are purely identified based on judgements and experiences. In most cases, variables with strong correlation with the target output(s) are suggested. The selection of training data is extremely crucial as it could improve adaptability, reliability, and robustness of an ANN. Training data with an extended phenomena

space margin and free from outrange data points are normally preferred. The larger training data can increase the accuracy of network generalization but it also increases the computation time of the learning process [10].

2.7 Error Back Propagation Learning Algorithm

Error back propagation is a common technique of teaching ANNs how to perform a given task. The term back-propagation refers to the manner in which the gradient is computed for nonlinear multilayer networks. In simple terms, the gradient is the derivatives of the error with respect to the weight ($g_k = \frac{\partial \varepsilon}{\partial w_k}$) . This method was first described by *Paul Werbos* in the mid 1970s, and it only gained recognition about many years later. Figure 2.10 shows a simplified supervised ANN training design chart using back-propagation method.

First of all, the input(s) and desired target (s) are presented to the network prior to computing the resulting output. The difference between the desired and resulting outputs is known as the error. If the error is greater than the acceptable threshold, the network will continue to adjust the weights and biases for all neurons using the error. Though, this process (adjusting of weights and biases) does not continue at infinitum. The network repeats the process until the error reaches an acceptable value (typically error ≤ goal), which means that the NN was trained successfully. If a maximum count of iterations is reached without attaining the goal, this implies that the NN training was not successful [9].

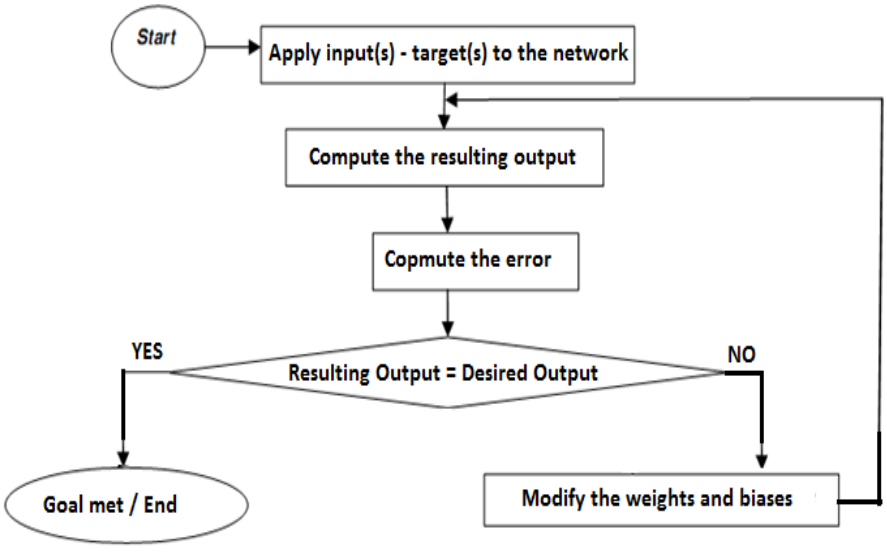


Figure 2.10: Chart of a supervised BP training method

CHAPTER THREE

SYSTEM CONTROL DESIGN

CHAPTER THREE

SYSTEM CONTROL DESIGN

3.1 Introduction

In the previous chapter, the current control technologies that have been widely used for speed control to improve the system performance were reviewed. Then, the shortcomings and problems of current control strategies are summarized. These problems limit the use of control strategies for speed control systems. In order to address these issues, future perspectives and approaches to improve the control performance control are discussed and concluded. In this study, neuro controller is designed for different purposes .The proposed controller design is mainly based on the following considerations:

New control approaches may merge both the conventional and advanced control methods. There is a possible way: a method by which individual merits are combined. PID control is usually utilized due to its practicality. But its control performance is mainly based on its parameters and improper selection of them will lead to poor control performance. Hence intelligent controller is required to regulate the PID parameters automatically to ensure the optimized control output. The control process that does not require intervention of the users with specific skills or knowledge. Closed-loop, real-time and on-line learning ability. The performance evaluation system could be included to make a control process have the self-learning and modifying ability.

3.2 Artificial Neural Network-based PID Controller

As investigation of all types of speed monitoring and control is a complicated matter, a neural network PID controller with back-propagation based weight updating algorithm is proposed in this study. The performance of the ANN-PID controller is tested by computer simulation using MATLAB code as discussed in chapter four.

3.2.1 Structure of the BPNN-PID controller

Figure 3.1 presents the structure of the proposed intelligent PID controller based on

BPNN learning algorithm. It consists two parts: a classic PID controller and ANN. The PID controller is used to control speed of the DC motor. The control performance depends on the setting of PID control parameters K_P, K_i and k_d which can be auto tuned by BPNN. BPNN uses an on-line training algorithm based on a gradient descent approach to update network weights and ensures that the designed neural network is able to calculate the desired PID parameters. Therefore, in this control approach, by combining classic PID control and intelligent BPNN the targeted system output can be tracked with a guaranteed stability.

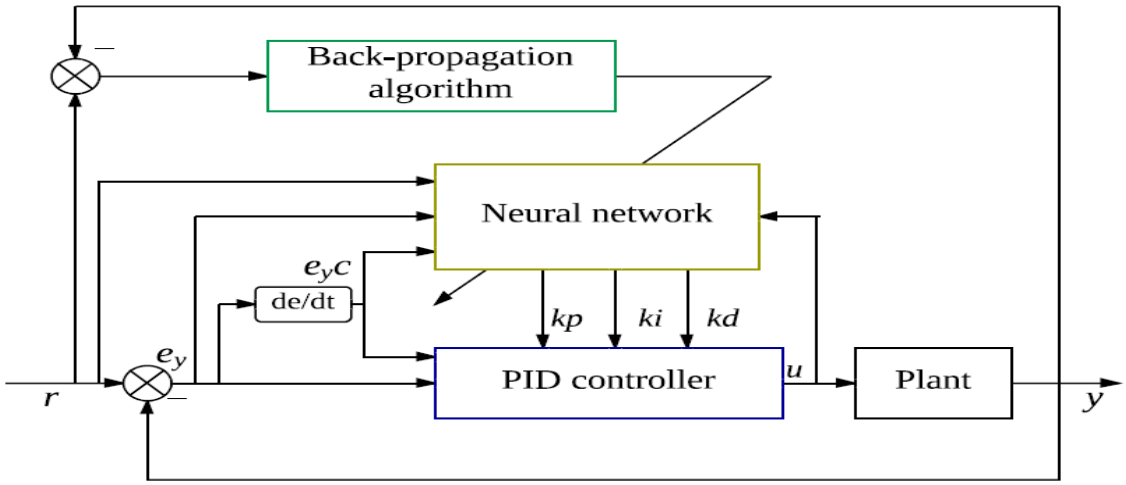


Figure 3.1: BPNN based PID control scheme

where u is the output of the PID controller, K_P is the proportional term, K_i is the integral term, K_d is the derivative term and e_y is the system error that can be expressed as follows:

e_y (k) = y(k) - r(k) (3.2)

Where y is the system actual output and r is the system targeted output.

3.2.2 PID control algorithm

The incremental digital PID control algorithm can be expressed as follows: [11]

$$u(k) = u(k-1) + k_p\{errro(k) - error(k-1)\} + k_i(k) + k_d\{error(k) - 2error(k-1) + error(k-2)\} \quad (3.1)$$

3.2.3 Back propagation neural network algorithm

If the neural network has sufficient amount of neurons, it is able to approximate any continuous function with only one hidden layer. Therefore, a neural network with only one hidden layer is designed. As shown in Figure 3.2, the proposed design has four-input-three-output BPNN with three layers: input layer, one single hidden layer and output layer. In this section, the forward feed algorithm and the back-propagation weights adjustment rule is discussed in detail [12].

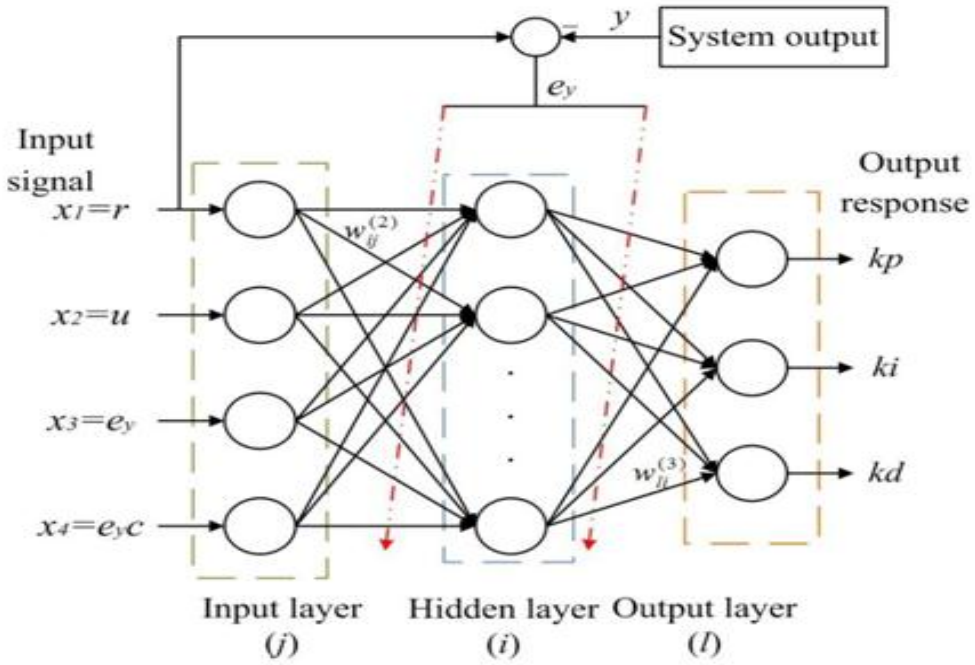


Figure 3.2: BPNN algorithm scheme

The designed neural network has four inputs as shown in both Figure 3.1 and Figure 3.2 and they are:

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} r \\ u \\ e_y \\ e_y c \end{Bmatrix} \tag{3.3}$$

Where r, *u* and *ey* is defined in equations; and *eyc* is the changing rate of system error *ey* that can be expressed as follows:

$$e_y c(k) = e_y(k) - e_y(k-1) \tag{3.4}$$

Output of each neuron in the input layer is expressed as:

$$O_j^{(1)} = x(j) \qquad \{j = 1,2,3,4\} \tag{3.5}$$

In the designed algorithm, superscript (1) stands for input layer.

Input of each neuron in the neural network hidden layer can be calculated based on the input layer output and is expressed as follows:

$$in_i^{(2)}(k) = \sum_{j=1}^{4} w_{ij}^{(2)} o_{-}^{(1)} \qquad \{i=1,2,\ldots\ldots N\} \tag{3.6}$$

where superscript (2) stands for hidden layer; w_{li} (2) is the weight connecting the input layer neurons to the hidden layer neurons and *N* is the number of neurons in the hidden layer.

Then, output of each neuron in the neural network hidden layer can be expressed as follows:

$$o_i^{(2)}(k) = f(in_i^{(2)}(k)) \tag{3.7}$$

where *f*(*x*) is the activation function in the hidden layer that presents the relation between the input and output of each neuron. Symmetrical sigmoid function is used as the activation function and can be expressed as follows:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.8}$$

Once the output of each neuron in hidden layer is calculated, the input of each neuron in the output layer can be given as:

$$in_l^{(3)}(k) = \sum_{j=1}^{N} w_{li}^{(3)} o_l^{(2)} \qquad \{=1,2,\ldots\ldots N\} \tag{3.9}$$

Where superscript (3) stands output layer; w_{li}(3) is the weight connecting the hidden layer neurons to the output layer neurons. The number of neurons in output layer is three and the outputs of the neurons are the PID parameters. Output of each neuron in the output layer is given by:

$$o_i^{(3)}(k) = g(in_i^{(3)}(k)) \tag{3.10}$$

$$o_1^{(3)}(k) = k_p \tag{3.11}$$

$$o_2^{(3)}(k) = k_i \tag{3.12}$$

$$o_3^{(3)}(k) = k_d \tag{3.13}$$

where $g(x)$ is the activation function that presents the relation between the input and output of each neuron in the output layer. Outputs of the output layer are the PID parameters K_P, K_i and k_d. Since these values cannot be negative, the non-negative Sigmoid function is used as the activation function in output layer and it is given as:

$$g(x) = \frac{1}{2}(1 + \tanh(x)) = \frac{e^x}{e^x + e^{-x}} \tag{3.14}$$

The proposed neural network can regulate the PID control parameters automatically and it can reduce sufficient time cost for engineers in control system design process. However, modeling errors often exist in model based process control and dramatically increase the difficulty to accurately control the process. Therefore, anon-line training algorithm is applied to adjust network weights for reducing the system error *ey* in the design of the BPNN controller.

3.2.4 Weight update

In this algorithm, the system output error function is defined by the given equation [12].

$$E(k) = \frac{1}{2}\big(r(k) - y(k)\big)^2 = \frac{1}{2}e \tag{3.15}$$

The training process of the neural network model must be carried out before it can be put into use. This training process is repeated until the mean square error of the training data reaches the desired minimum. In the present work, the training processes based on back propagation. The basic idea of back propagation is to adjust the neuron weights using gradient descent algorithm on the error function in an iteration process. Generally, the adjustment of each weight from hidden-layer to output-layer can be expressed as follows:

$$\Delta w_{li}^{(3)}(k) = -\eta \frac{\partial E(k)}{\partial w_{li}^{(3)}} \tag{3.16}$$

However, in order to avoid the 'local minima' which is the best known problem associated with back-propagation algorithm; a momentum term is added to the weight change in the proposed algorithm. This means that the weight change this iteration depends not just on the current error, but also on previous changes. So the adjustment of each weight from hidden-layer to output-layer is modified as follows based on the system output error function as shown in Figure 3.3:

$$\Delta w_{li}^{(3)}(k) = -\eta \frac{\partial E(k)}{\partial w_{li}^{(3)}} + \alpha \Delta w_{li}^{(3)}(k-1) \tag{3.17}$$

Where is η learning rate, α is momentum factor. Since:

$$\frac{\partial E_y(k)}{\partial w_{li}^{(3)}(k)} = \frac{\partial E_y(k)}{\partial y(k)} \times \frac{\partial y(k)}{\partial u(k)} \times \frac{\partial u(k)}{\partial O_l^{(3)}(k)} \times \frac{\partial O_l^{(3)}(k)}{\partial in_l^{(3)}(k)} \times \frac{\partial in_l^{(3)}(k)}{\partial w_{li}^{(3)}(k)} \tag{3.18}$$

$$\frac{\partial in_l^{(3)}(k)}{\partial w_{li}^{(3)}(k)} = O_i^{(2)}(k) \tag{3.19}$$

And the following equations are calculated:

$$\frac{\partial u(k)}{\partial O_1^{(3)}(k)} = e_y(k) - e_y(k-1) \tag{3.20}$$

$$\frac{\partial u(k)}{\partial O_2^{(3)}(k)} = e_y(k) \tag{3.21}$$

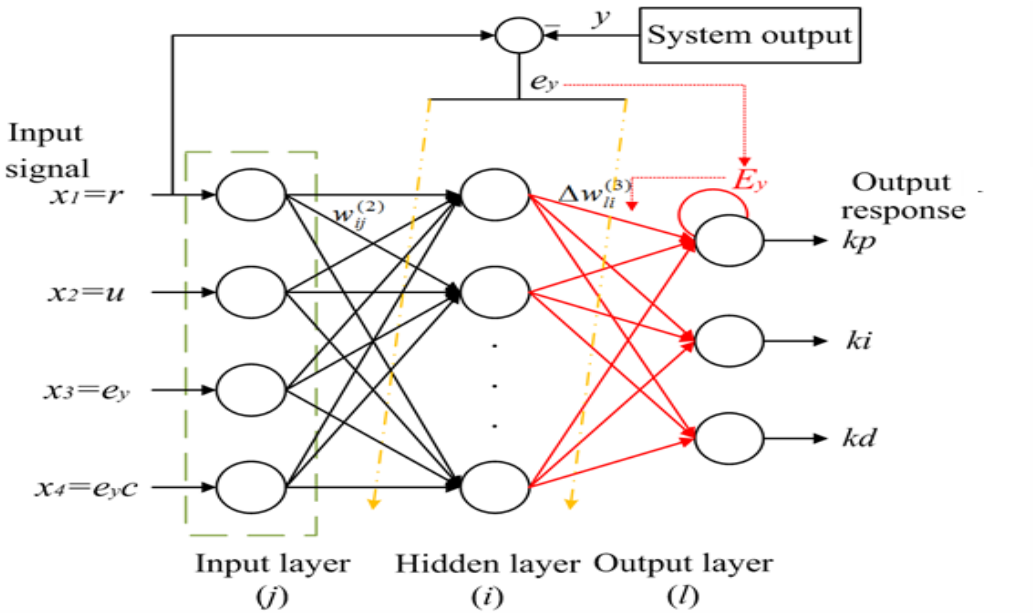


Figure 3.3: Adjustment of weight from hidden layer to output layer

$$\frac{\partial u(k)}{\partial O_3^{(3)}} = e_y(k) - 2e_y(k-1) + e_y(k-2) \tag{3.22}$$

Then, the learning algorithm of the weight update in output layer can be expressed as follows:

$$\Delta w_{li}^{(3)}(k+1) = \Delta w_{li}^{(3)}(k) + \Delta w_{li}^{(3)}(k) \tag{3.23}$$

$$\Delta w_{li}^{(3)}(k) = \alpha \Delta w_{li}^{(3)}(k-1) + \eta \Delta \delta_l^{(3)} O_i^{(2)}(k) \tag{3.24}$$

where $\delta_l^3$is the error function of the network hidden layer that is need for the adjustment of weights from input layer to hidden layer as shown in Figure 3.4.

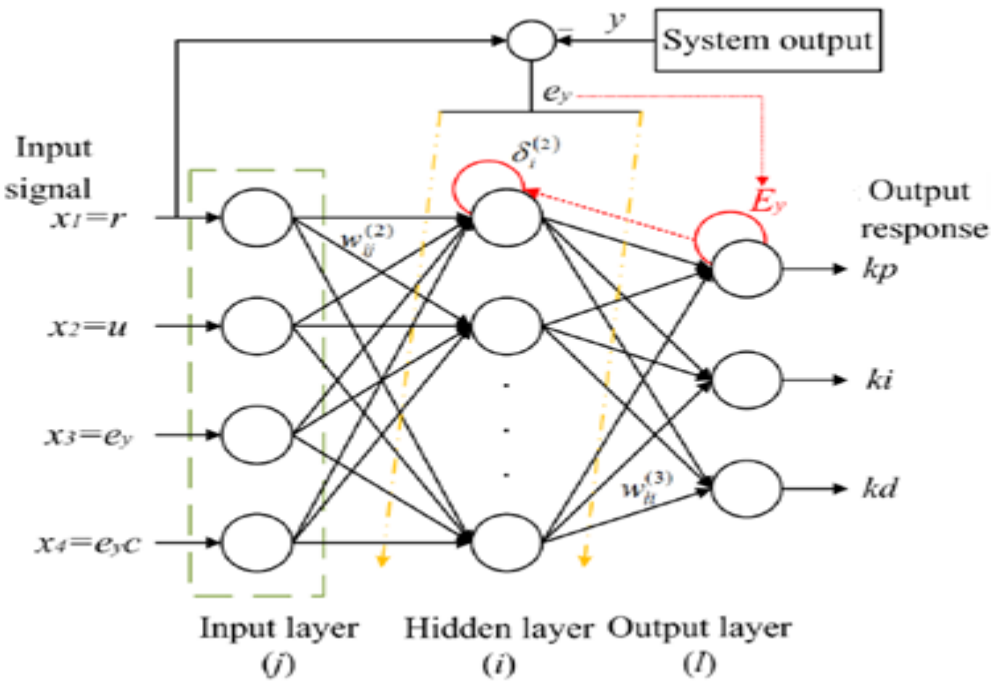


Figure 3.4: Error function of the network hidden layer

δ_l^3can be expressed as follows: [12]

$$\delta_l^{(3)} = e_y(k) \times \frac{\partial y(k)}{\partial u(k)} \times \frac{\partial u(k)}{\partial O_i^{(3)}(k)} \times g'(in_i^{(3)}(k)) \quad (3.25)$$

3.3 Summary

In the control process, the weights in the neural network are trained by the back-propagation weights adjustment rule in order to obtain the best PID parameters K_p, K_i and K_d. For the PID controller. Therefore, an acceptable indoor air quality cans be provided by the control of designed system. The algorithm of the BPNN based PID can be summarized as follows and its flowchart is presented in Figure 3.4:

i) Initialize the each weight in the neural network $w_{ij}(k)$ and $w_{li}(k)$, as well as learning rate and momentum factor while k=1.

ii) Collect data $r(k)$ and $y(k)$ and calculate the system error e_y using Equation (3.2).

iii) Calculate the input and output of each neuron and get the PID parameters r_p, K_i and K_d.

iv) Calculate the output of PID controller using Equation (3.1).

v) On-line training. Adjust the weight of each neuron in the neural network with the back-propagation learning algorithm in order to realize self-adaptive regulation of the PID parametersK_p, K_i and K_d.

vi) Set k=k+1 and go back to rule (i).

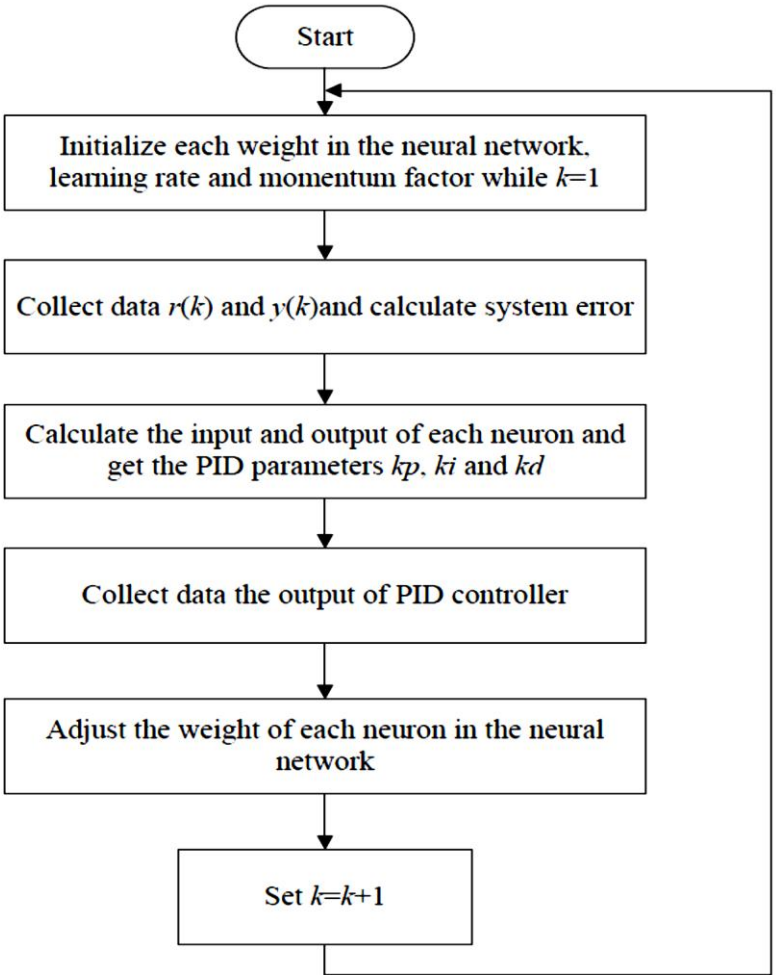


Figure 3.5: Flow chart of BPNN-PID control scheme

CHAPTER FOUR

SYSTEM SIMULATION

RESULTS AND DISCUSSION

CHAPTER FOUR

SYSTEM SIMULATION RESULTS AND DISCUSSION

4.1 Introduction

The detail designs of the proposed controllers were introduced in chapter three. In order to achieve the goal of speed of DC motor to be controlled properly. three controllers: neural network based PID controller and back propagation neural network based PID controller were proposed in chapter three. In this chapter, the performance of the proposed controller is presented. The simulating tests of the control processes are based on the mathematical models of the DC motor is discussed in chapter three. The simulations have been taken on the platform of MATLAB. The controllers' performance including overshoot, response speed, adaptability, robustness and etc. are discussed. Then it analyzed whether the proposed controllers are suitable for their control objects and process.

4.2 Uncontrolled System Response to Step Input

Figure 4.1 shows the system response without controller.

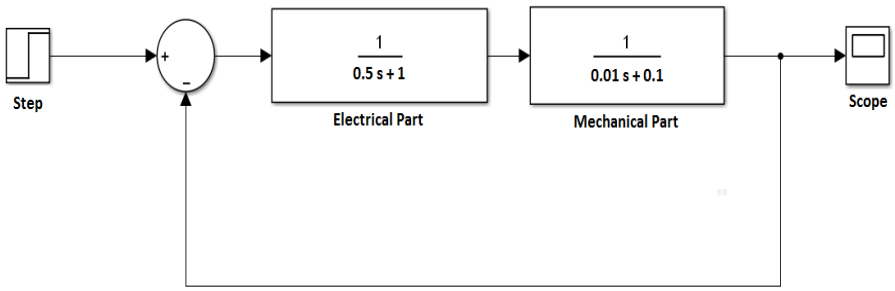


Figure 4.1: Simulink model of uncontrolled system

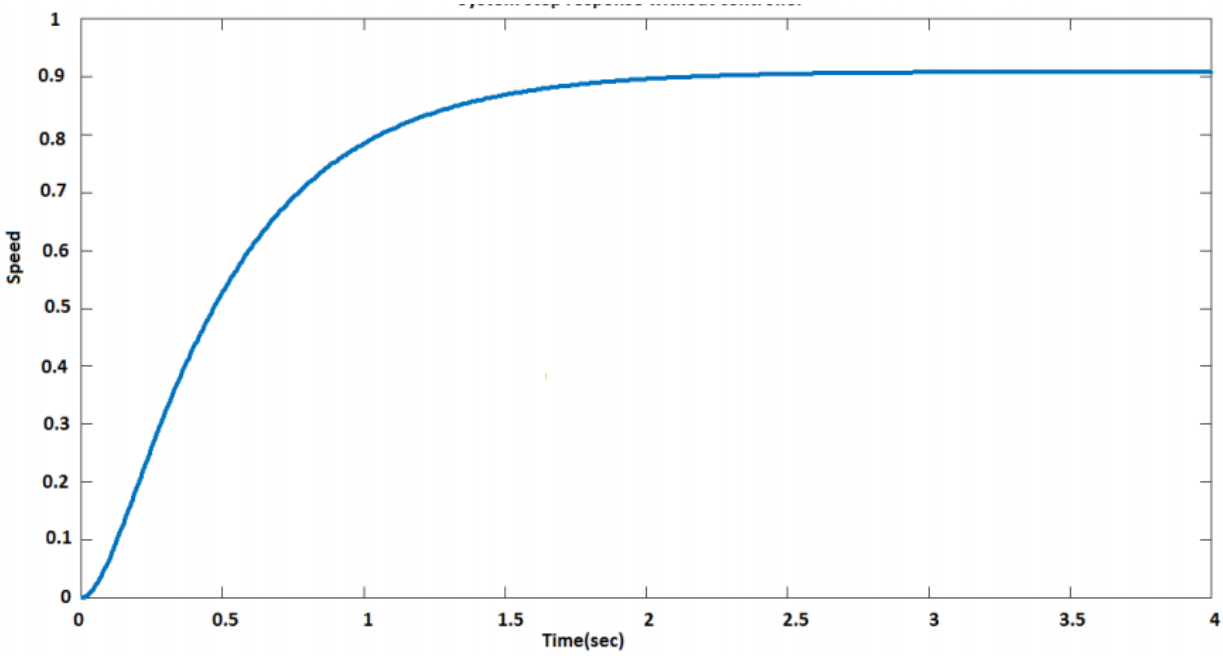


Figure 4.2: speed response of uncontrolled system

4.3 System Response with PID Controller to Step Input

As mentioned earlier in chapter two, PID controller is a popular conventional approach. Figure 4.3 shows the Simulink diagram of the DC motor with PID controller. The motor speed in Figure 4.4.

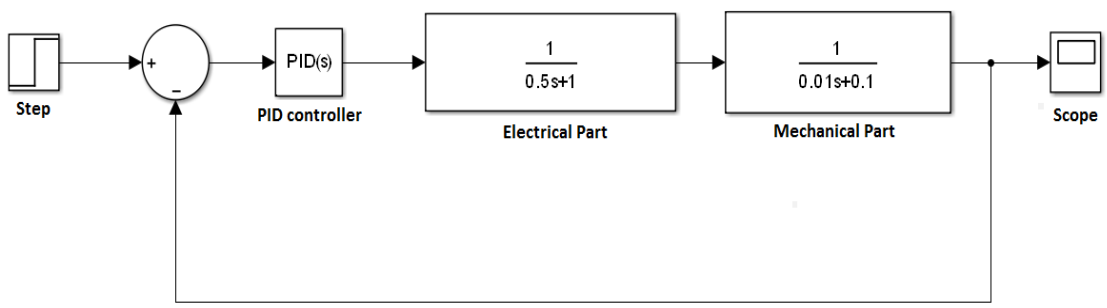


Figure 4.3: Simulink model of DC motor with PID controller

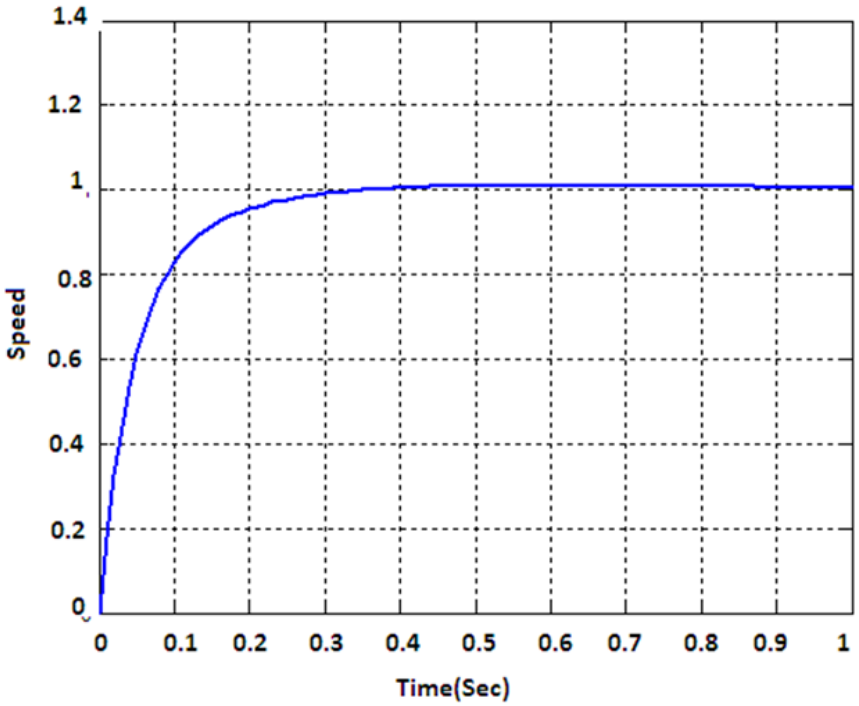


Figure 4.4: speed response of the system with PID controller

4.4 Simulating Tests of BPNN-PID Control

ANN-PID control was designed to solve the difficulties of speed control and to achieve the goal of maintaining good performance. The proposed ANN-PID controller combined a typical PID controller and a neural network which the back-propagation algorithm is applied to. The structure of this design was discussed in Chapter three as it has two main parts: a typical conventional PID control for controlling the speed and a back-propagation neural network to regulating the parameters for the PID controller according to the current conditions. Theoretically analysis according to chapter three showed that the proposed ANN-PID speed controller has the following potentials:

PID controller is suitable for various control objects. Neural networks for optimal PID parameters tuning to ensure stability, back-propagation algorithm for adjustment of weights in neural network to ensure the system quickly response to change. In order to discussed and evaluate the proposed design, the simulating tests have been done by using the MATLAB code shown in Appendix, Then the simulating results

are used to indicate the controller's performance based on several indexes: response speed, overshoot, time constant, stability and adaptability.

Then whether the proposed controller can achieve the targeted control requirement is discussed based on the controller's performance of the listed indexes. Thus simulations are used to conduct and results are analyzed to discuss the performance including of the proposed control strategy. During the simulating tests, the accurate mathematical model of the real control objects that in this case is the speed.

As usual, the step signal is used to for simulation test. After several tests, set the best initial learning rate η=0.28 and momentum factor α=0.04.The weights in the neural network are initialized in the range [-0.5, 0.5] randomly. The randomly preset weights may cause a little unstable to the control process but the back-propagation algorithm is able to quickly response to any uncertain parameters and the weights of neural network are updated in relative short time to ensure the targeted output.

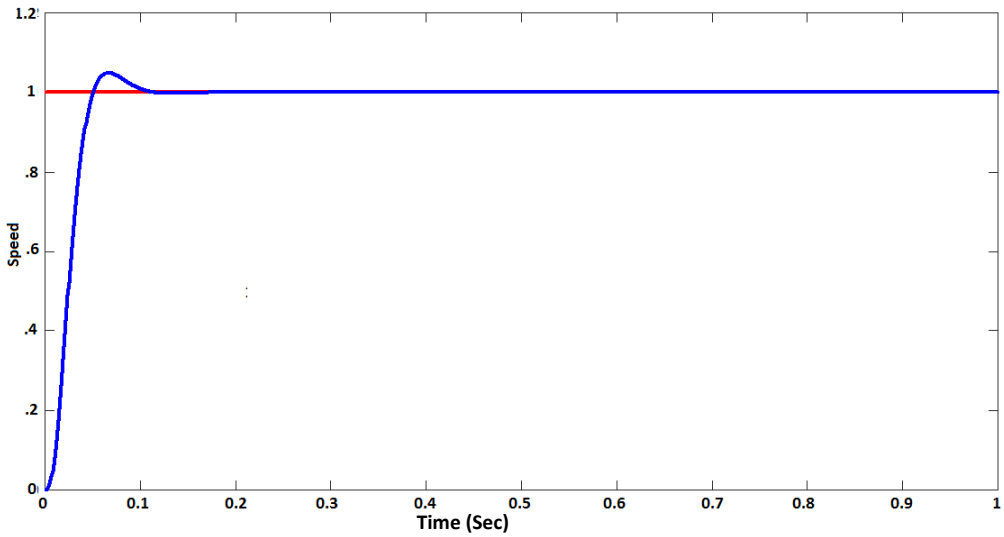


Figure 4.5: System output response to step input with Neural Network

The step signal (r(k)=1) is introduced at time t=0. The simulation result of the Proposed control system output is presented in Figure 4.6. As shown in the Figure 4.6, the system has very fast rising speed as well as very small overshoot. The time constant τ=0.003s, settling time t_s= 0.091s and the maximum percent overshoot 5%. The uncertainness might be caused by randomly preset weight values of neural network have not significant disturbed control process at the beginning of response as the curve rising fast. Then it is brought back to the set-point before the overshot is

still very small, 5% in this simulation work and the control process is brought to the steady state, the steady error of which is zero as shown in profile. It has to be clear that the steady state error is zero because the control object function is calculated based on an ideal model of the system and the system error generally exists in real control process.

In Figure 4.7 PID controller output response to the step input signal is presented. It can be seen that the controller calculated the output (control command) for the plant (command was sent to the equipment and the change the speed). The system output results in Figure 4.6 prove that the controller is able to obtain required output to ensure good performance on fast and accurate control.

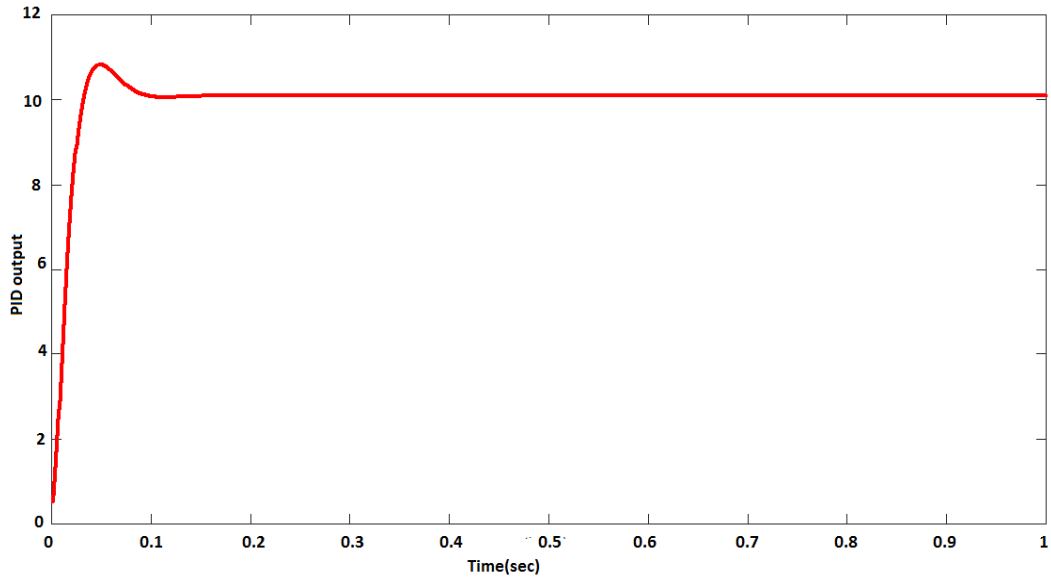


Figure 4.6: PID with neural network control signal

The auto-tuning process of the PID parameters K_P, K_i and K_d can be observed in Figure 4.8. It shows that K_P= 0.25, K_i=0.23 and K_d=0.05 at the beginning of the control process and the PID parameters are tuned by the neural network during the controlprocess. The system output results in Figure 4.6 shows good control performance. This means that the proper PID parameters can be obtained using online training algorithm based neural network control scheme. The value of the PID control parameters are kept modifying to optimize the control performance until the system enters the steady state. The PID parameters are settled at K_P= 0.372, K_i =0.0.462 and K_d=0.162 and kept stable.

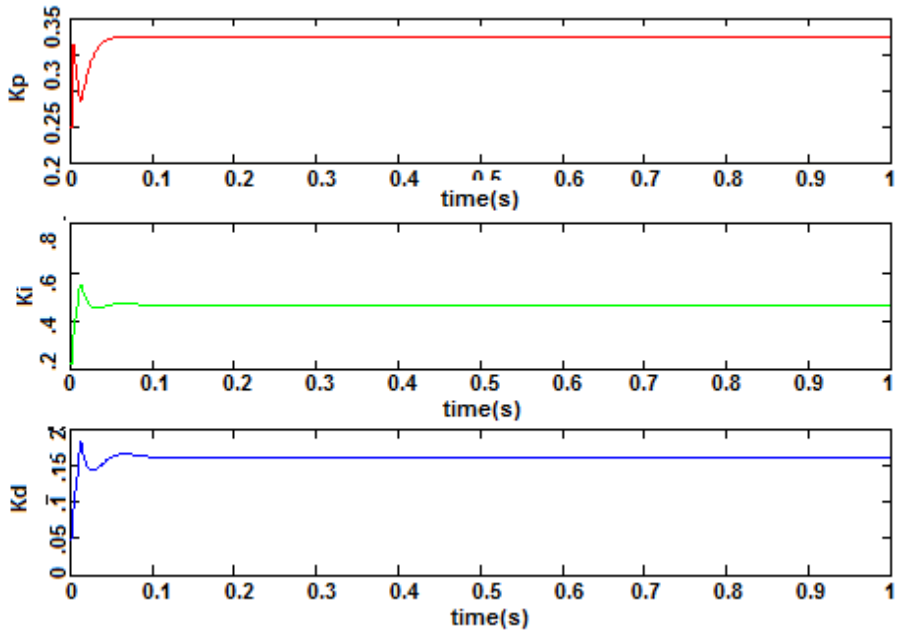


Figure 4.7: PID parameters auto regulating

Equation (3.1) presented the input of the neural network in the proposed control strategy. In other words, the PID parameters are regulated based on these variables. System output error (e), change of system output error (e_c), PID controller output error (u) and system output (y). It can be seen if only one input is applied to the neural network; the incorrect PID parameters may be produced. Therefore, with four input variables the design BPNN algorithm can calculate and then provide the optimized K_P, K_i and K_d for the best control performance.

Table 4.1: Comparison between various parameters for different controllers.

Type of Control	**Settling Time (second)**	**Overshoot (%)**	**Stability**	**Steady State Error**
Uncontrolled	1.65	-	stable	0.008
PID Controller	0.365	0	stable	0
ANN controller	0.091	5	stable	0

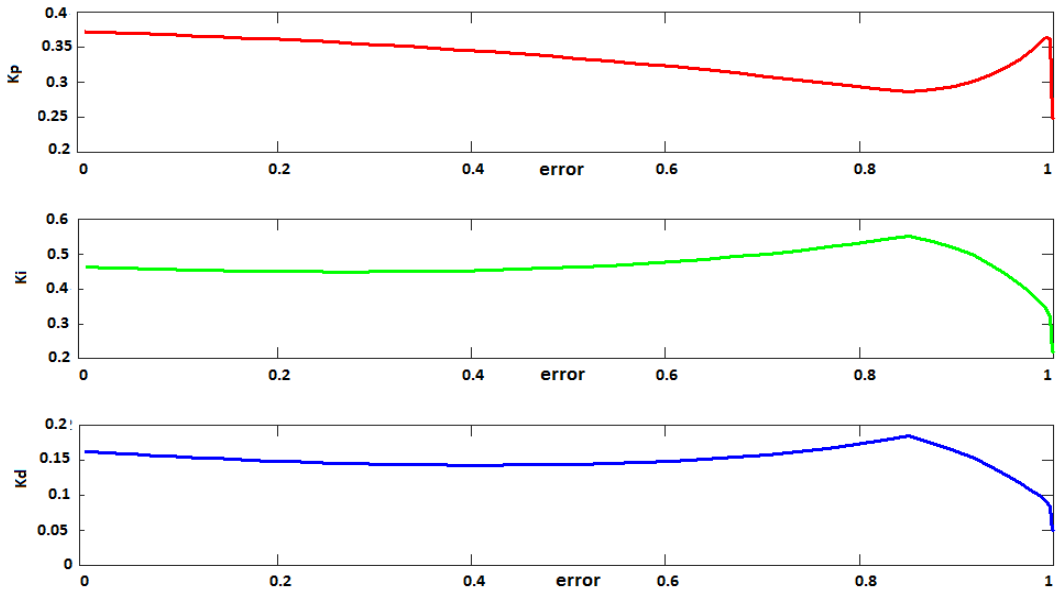


Figure 4.8: Relation between PID parameters and system error

From Table 4.1, it is clear that system without controller is stable without overshoot, but with conventionality, it becomes stable but overshoot and settling time is relatively high for K_p= 200, K_d = 100 and K_i= 10. With help of multilayer neuro PID the system becomes stable and settling time and overshoot becomes too small but rise time of the system increases which is a setback but it can be neglected since the basic requirement is minimum settling time and less overshoot and this is achieved as it becomes 90 milliseconds and 5% respectively.

CHAPTER FIVE

CONLUSION AND RECOMMENDATIONS

CHAPTER FIVE

CONLUSION AND RECOMMENDATIONS

5.1 Conclusion

The current control methods of for speed control systems using PID have been reviewed. The drawbacks of current control methods are summarized based on the literature review. Then, based on discussion of advantages and disadvantages of current control technologies. In this research, ANN based PID controller is developed for speed control of DC motor. Principles and detailed designs are introduced in chapter three. These controllers are tested by computational simulation. The simulation is crried out on the platform of MATLAB. The programming codes have been developed to simulate the control strategies and the indoor environment model for testing the controllers 'performance. The simulation results show clearly that the Neuro PID controller is better compared to conventional PID, although the PID controller has less overshoot but from the point view of settling time, the neuro controller has a very short settling time compared to PID controller. Having very short settling time is an excellent characteristics ant tells that the system will settle to the steady state cast within a very short time and the transient can be ignored.

5.2 Recommendations

The major work of this research is to discuss the potential of improving performance of control system by using ANN-based controllers. The results show that excellent control performance has been achieved. But there are futures works need to be carried out:

- The experimental investigation of ANN-PID controllers will be carried out since they are only tested by computer simulations in this research.
- The performance of control system could be significant improved by using the proposed controllers based on the results.
- According to current studies, there are one model used for tuning the PID controller, other types also can be used including: fuzzy logic controller and radial basic function controller.

REFERERNCES

[1] F. Rivas, et al, "Neural Network-Based Auto-Tuning for PID Controllers", Neural Network World",Vol,11.No3,pp.277-284, 2001.

[2] Nedjeljko Peric, et al, “Modification and Application of Auto_tunning PID Controller” , Proceedings of the 8th IEEE Mediterranean Conference on Control and Automation (MED 2000), Rio Patras, 17-19 July 2000

[3] Makableh,et al, "Efficient Control of DC Servomotor Systems Using Backpropagation Neural Networks" , Electronic Theses and Dissertations, 771, 2011.

[4] Rahul Patel, et al," Multilayer Neuro PID Controller Based on Back Propagation Algorithm”, Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015) , Procedia Computer Science 54,pp.207 – 214, 2015 .

[5] Camilla Anderssonet, et al, “Auto tuning of a PID-controller”, Department of Automatic Control Lund Institute of Technology, October 2004.

[6] M. Grégroire, et al, “Development of an Auto Tuning PID and Application to the Pulp and Paper Industry”, Automatic, Volume 32, pp.71-82, 1996.

[7] António Eduardo de Barros Ruano," Applications of Neural Networks to Control Systems", University of Wales, Bangor, February, 1992.

[8] Sangram Keshari ,et al, “Study of the Design and Tunning Methods of PID Controller Based on Fuzzy Logic and Genetic Algorithm”, National Institute of Technology, Rourkela, May, 2011.

[9] Howard Demuth, et al, “Neural Network Design”, 2nd Edition, Orlando De Jesús Consultant Frisco, Texas, 2006.

[10] Haykin Simon , “ An Introduction to Feed Forward Networks”, 1999

[11] Demuth, et al, “MATLAB Neural Network Tool box Documentation”, 2004

[12] Song, Yang, “Intelligent PID Controller Based on fuzzy Logic Control and Neural Network Technology for Indoor Environment Quality Improvement”, University of Nottingham, 2014.

APPENDIX

MATLAB CODE

```
%PID Control based on BPNN
clear all;
close all;
xite=0.28;
alfa=0.04;
IN=4;H=5;Out=3; %NN Structure
%wi= random [-0.5,0.5];
wi=[-0.4394 -0.2696 -0.3756 -0.4023;
-0.4603 -0.2013 -0.3024 -0.2596;
-0.4749 0.4543 -0.3820 -0.2437;
-0.3625 -0.4724 -0.3463 -0.2859;
0.1425 0.4279 -0.2406 -0.4660];
wi_1=wi;wi_2=wi;wi_3=wi;
wo=[0.3576 0.2616 0.2820 -0.1416 -0.1325;
-0.1146 0.2949 0.1352 0.2205 0.4508;
0.3201 0.4566 0.3672 0.4962 0.3632];
%wo= ramdom[-0.5,0.5];
wo_1=wo;wo_2=wo;wo_3=wo;
x=[0,0,0];
du_1=0;
u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
Oh=zeros(H,1); %Output from NN hidden layer
I=Oh; %Input to NN hidden layer
error_2=0;
error_1=0;
ts=0.001;
for k=1:1:1000
time(k)=k*ts;
```

```
rin(k)=1.0;
%nonlinear model
%a(k)=1.4*(1-0.8*exp(-0.1*k));
%a(k)=1.4*(1-0.8*exp(-0.1*k));
%yout(k)=a(k)*y_1/(1+y_1^2)+u_1;
yout(k)=1.511*y_1-0.5488*y_2+0.002059*u_1+0.001686*u_2;
error(k)=rin(k)-yout(k);
xi=[rin(k),yout(k),error(k),1];
x(1)=error(k)-error_1;
x(2)=error(k);
x(3)=error(k)-2*error_1+error_2;
epid=[x(1);x(2);x(3)];
I=xi*wi';
for j=1:1:H
Oh(j)=(exp(I(j))-exp(-I(j)))/(exp(I(j))+exp(-I(j))); %hidden Layer
end
K=wo*Oh; %Output Layer
for l=1:1:Out
K(l)=exp(K(l))/(exp(K(l))+exp(-K(l))); %Getting kp,ki,kd
end
kp(k)=K(1);ki(k)=K(2);kd(k)=K(3);
Kpid=[kp(k),ki(k),kd(k)];
du(k)=Kpid*epid;
u(k)=u_1+du(k);
dyu(k)=sign((yout(k)-y_1)/(du(k)-du_1+0.0001));
%Output layer
for j=1:1:Out
dK(j)=2/(exp(K(j))+exp(-K(j)))^2;
end
for l=1:1:Out
delta3(l)=error(k)*dyu(k)*epid(l)*dK(l);
end
```

```
for l=1:1:Out
for i=1:1:H
d_wo=xite*delta3(l)*Oh(i)+alfa*(wo_1-wo_2);
end
end
wo=wo_1+d_wo+alfa*(wo_1-wo_2);
%Hidden layer
for i=1:1:H
dO(i)=4/(exp(I(i))+exp(-I(i)))^2;
end
segma=delta3*wo;
for i=1:1:H
delta2(i)=dO(i)*segma(i);
end
d_wi=xite*delta2'*xi;
wi=wi_1+d_wi+alfa*(wi_1-wi_2);
%Parameters Update
du_1=du(k);
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
wo_3=wo_2;
wo_2=wo_1;
wo_1=wo;
wi_3=wi_2;
wi_2=wi_1;
wi_1=wi;
error_2=error_1;
error_1=error(k);
end
ec = gradient(error);
figure(1);
plot(time,rin,'r',time,yout,'b');
```

```
xlabel('time(s)');ylabel('speed');
figure(2);
plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('PID output');
figure(4);
subplot(311);
plot(time,kp,'r');
xlabel('time(s)');ylabel('kp');
subplot(312);
plot(time,ki,'g');
xlabel('time(s)');ylabel('ki');
subplot(313);
plot(time,kd,'b');
xlabel('time(s)');ylabel('kd');
figure(5);
subplot(311);
plot(error,kp,'r');
xlabel('error');ylabel('kp');
subplot(312);
plot(error,ki,'g');
xlabel('error');ylabel('ki');
subplot(313);
plot(error,kd,'b');
xlabel('error');ylabel('kd');
figure(6);
subplot(311);
plot(ec,kp,'r');
xlabel('ec');ylabel('kp');
subplot(312);
```

```
plot(ec,ki,'g');
xlabel('ec');ylabel('ki');
subplot(313);
plot(ec,kd,'b');
xlabel('ec');ylabel('kd');
figure(7);
subplot(311);
plot(u,kp,'r');
xlabel('u');ylabel('kp');
subplot(312);
plot(u,ki,'g');
xlabel('u');ylabel('ki');
subplot(313);
plot(u,kd,'b');
xlabel('u');ylabel('kd');
figure(8);
subplot(311);
plot(yout,kp,'r');
xlabel('yout');ylabel('kp');
subplot(312);
plot(yout,ki,'g');
xlabel('yout');ylabel('ki');
subplot(313);
plot(yout,kd,'b');
xlabel('y');ylabel('kd');
```