SPEED CONTROL OF DC MOTOR USING NOVEL NEURAL NETWORK CONFIGURATION

A PROJECT THESIS SUBMITTED IN THE FUFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

ELECTRICAL ENGINEERING

BY

MANISH MISHRA

ROLL NO. 10502014

DEPARTMENT OF ELECTRICAL ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

ROURKELA

2008-2009

SPEED CONTROL OF DC MOTOR USING NOVEL NEURAL NETWORK CONFIGURATION

A PROJECT THESIS SUBMITTED IN THE FUFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

ELECTRICAL ENGINEERING

BY

MANISH MISHRA

ROLL NO. 10502014

UNDER THE GUIDANCE OF

PROF. J.K SATHPATHY

DEPARTMENT OF ELECTRICAL ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

ROURKELA

2008-2009

ACKNOWLEDGEMENT

I am deeply indebted to Professor J.K SATHPATHY for his able guidance, expert advice and showing me the right way of carrying out the research project. I am also thankful to him for giving me the special attention and cooperation in between his very busy hours.

I am also grateful to our entire electrical faculty for their efforts to strengthen the fundamental concepts so that one can take up any assignment of serious gravity with confidence. I would like to take this opportunity to express my heart-felt indebtedness towards Prof. P. C. Panda and Prof. B. D. Subudhi for their support and enlightenment.

At this moment I would also like to express my gratitude for the technical staff of our laboratories. They have always helped me in every-way they can during my project work.

I also duly acknowledge the work of the people listed in references.

Date: MANISH MISHRA

(ROLL NO.10502014)

NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

CERTIFICATE

This is to certify that the progress report of the thesis entitled, "SPEED CONTROL OF DC MOTOR USING NOVEL NEURAL NETWORK CONFIGURATION**"** submitted by **Shri Manish Mishra** in partial fulfilment of the requirements for the award of Bachelor of Technology degree in Electrical Engineering at the National Institute of Technology Rourkela (Deemed University), is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any degree or diploma.

Prof. J.K SATHPATHY

Date: Department of Electrical Engineering

Place: National Institute of Technology Rourkela

CONTENTS

LIST OF FIGURES

ABSTRACT

This paper uses Artificial Neural Networks (ANNs) in estimating speed and controlling it for a separately excited DC motor. The rotor speed of the dc motor can be made to follow an arbitrarily selected trajectory. The purpose is to achieve accurate trajectory control of the speed, especially when the motor and load parameters are unknown.

Such a neural control scheme consists of two parts. One is the neural identifier which is used to estimate the motor speed. The other is the neural controller which is used to generate a control signal for a converter. These two neural networks are trained by Levenberg-Marquardt back-propagation algorithm. ANNs used in this are the standard three layers feed-forward neural network with sigmoid activation functions in the input and hidden layers while linear activation function is employed for the output layer.

The conventional constant gain feedback controller fails to maintain the performance of the system at acceptable levels under unknown dynamics in load torque. On the other hand, ANNs act as an effective tool to implement the model and adaptive control in a complicated non-linear system having expansive allocations.

The adaptive learning algorithm is formed in such a way that the learning rate is as large as possible while maintaining the stability of the learning process. This simplifies the learning process in terms of computation time, which is of special importance in real-time implementation.

Chapter 1

INTRODUCTION

The development of high performance motor drives is very important in industrial applications. Generally, a high performance motor drive system must have good dynamic speed command tracking and load regulating response.

D.C motors have long been the primary means of electric traction. D.C motor is considered a SISO system having torque/speed characteristics compatible with most mechanical loads. This makes a D.C motor controllable over a wide range of speeds by proper adjustment of its terminal voltage. Recently, brushless D.C motors, induction motors, and synchronous motors have gained widespread use in electric traction. However, there is a persistent effort towards making them behave like dc motors through innovative design and control strategies. Hence dc motors are always a good proving ground for advanced control algorithm because the theory is extendable to other types of motors.

Many practical control issues (motor control problems):

- Variable and unpredictable inputs
- Noise propagation along a series of unit processes
- Unknown parameters
- Changes in load dynamics

Under these conditions, the conventional constant gain feedback controller fails to maintain the performance of the system at acceptable levels. The incorporation of feed forward in artificial neural networks is important for several reasons the dynamical properties of the system, and in practice it may improve the performance. They are generally present in most non-linear dynamical system and can be used to implement specific structures.

Advantages of using ANNs:

- Learning ability
- Massive parallelism
- Fast adaptation
- Inherent approximation capability
- High degree of tolerance

Speed control techniques in separately excited dc motor:

- Varying the armature voltage in the constant torque region.
- In the constant power region, field flux should be reduced to achieve speed above the rated speed.

Methods of speed control:

- Traditionally rheostat armature control method was used for low power dc motors.
- Use of conventional PID controllers.
- Neural network controllers (NNC).
- Constant power field weakening controller based on load-adaptive multi-input multi-output linearization technique (in high speed regimes).
- A single phase uniform PWM ac-dc buck-boost converter with only one switching device used for armature voltage control.
- Use of NARMA-L2 (Non-linear Auto-regressive Moving Average) controller in the constant torque region.

Through experience gained in designing trajectory controllers based on self-tuning and PID control, it is seen that the neural network controller gives comparable performance in speed tracking. In addition to those mentioned above, a unique advantage of the neural network controller is its ability to cope with bad measurement data that occur during training and testing. However, a key drawback is the inadequate integral gain in the feedback loop, resulting in steady-state errors of the shaft position. Direct position tracking can alleviate this problem.

The traditional means adapted by the motion control industry with motor drives has been the approach of linearizing the system dynamics and designing a linear feedback controller. However, in high-performance motor drives such an approach is seldom satisfactory, as it results in poor speed and position tracking when sudden changes in load result in continuous acceleration or deceleration of the motor/load system. Adaptation is necessary to ensure optimal performance.

Chapter 2

BACKGROUND INFORMATION

2.1 OVERVIEW:

The neural network consists of junctions which are connected with LINKS, also called processing units. For each junction a number is ordered, this number is called weight. The weights are the tools for the long distance information storing in the neural network, the learning process occurring with the appropriate modification of weights. These weights are modified so that the network input/output behaviour is in consonance with the environment, which provide the input data.

The calculation algorithm consists of two basic steps:

- Calculation of the output of the network, with inputs and weights.
- Modification of weights with learning algorithm.

A single input neuron consists of a scalar input ‘p’ multiplied by the scalar weight ‘w’ to form ‘wp’ which is fed to the summer along with bias ‘b’ multiplied by ‘1’.

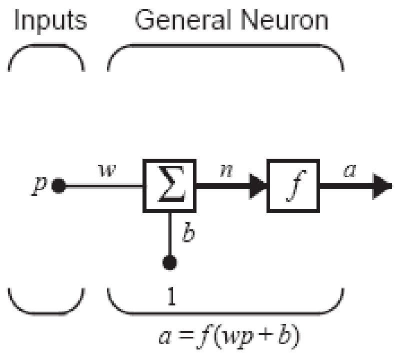

Figure 1: Basic Neural Network

The net input is ‘wp+b’ and the output ‘a’ is;

a=f (WP+b);

f- Transfer function

W & b can be adjusted by learning rule.

2.2 TRANSFER FUNCTION:

- HARD-LIMIT TF:

  a = hardlim (n)

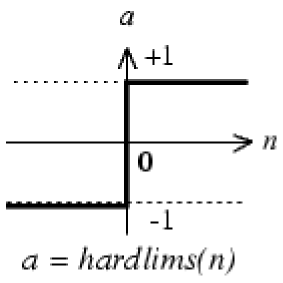

Figure 2: Symmetric Hard-limit Transfer Function

- LINEAR TF:

  a = purelin (n)

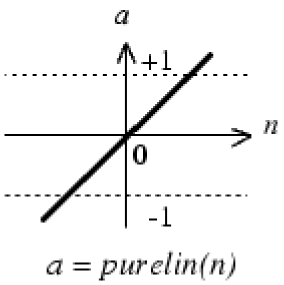

Figure 3: Linear Transfer Function

- LOG-SIGMOID TF:

  a = logsig (n)

  i.e. a = 1/ (1+exp (-n));

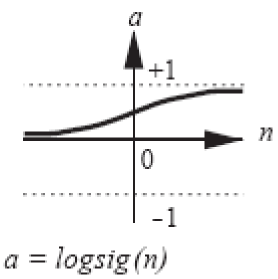

Figure 4: Log-Sigmoid Transfer Function

Used in multilayer n/w and trained using BPA as it is differentiable.

2.3 THREE-LAYER NETWORK:

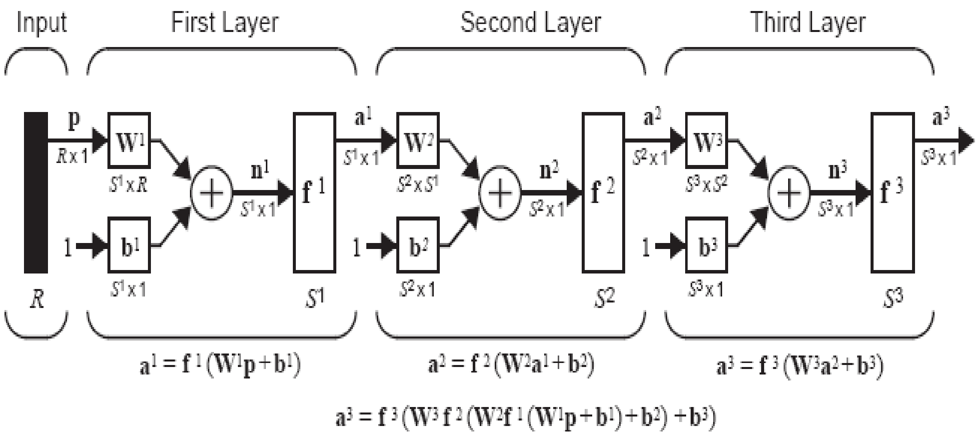

Figure 5: Three-Layer Network

3^{rd} layer - OUTPUT LAYER

1^{st} & 2^{nd} layer – HIDDEN LAYER

2.4 UNIVERSAL APPROXIMATORS:

$$f^1(n) = \frac{1}{1+e^{-n}} \quad \text{and} \quad f^2(n) = n \; .$$

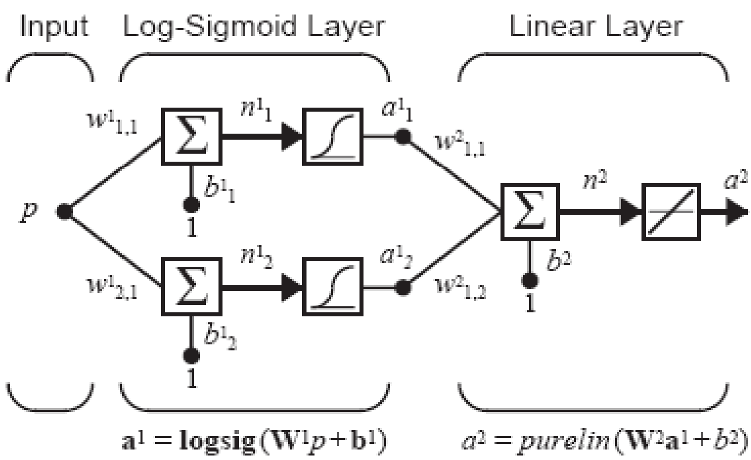

Figure 6: Universal Approximators

2.5 BACK PROPAGATION ALGORITHM:

- Procedure for selecting parameters- TRAINING the n/w.
- BPA is a method of TRAINING.
- Based on gradient descent.
- For a multilayer n/w, BPA is a gradient descent optimization procedure where we minimize a mean square error called <u>PERFORMANCE INDEX (PI).</u>

P_q – input to n/w

A_q – n/w output

T_q – corresponding target output

$$F(\mathbf{x}) = \sum_{q=1}^{Q} e_q^2 = \sum_{q=1}^{Q} (t_q - a_q)^2 .$$

$e_q = T_q - A_q$

X – Vector containing all n/w w & b.

Q- Total number of Errors in the network

F(x)- Performance Index

<u>For Multiple outputs</u>;

$$F(\mathbf{x}) = \sum_{q=1}^{Q} \mathbf{e}_q{}^T \mathbf{e}_q = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) .$$

<u>Using stochastic approx.;</u>

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k) - \mathbf{a}(k))^T (\mathbf{t}(k) - \mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k) ,$$

Where the expectation of the squared error has been replaced by the squared error at iteration ‘k’.

2.5.1 STEEPEST DESCENT ALGORITHM FOR THE APPROX. MEAN SQUARE ERROR:

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial w_{i,j}^{m}},$$

$$b_{i}^{m}(k+1) = b_{i}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial b_{i}^{m}},$$

where α is the learning rate.

i, j, m- Layers of neurons

F- Perfromance Index

w, b- respective weights and bias as per their sub-script and superscript.

Chapter 3

SEPARATELY EXCITED DC MOTOR

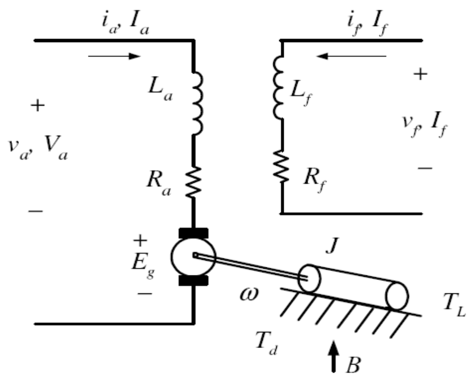

Figure 7: Separately Excited DC Motor

- The field windings are used to excite the field flux.
- Armature current is supplied to the rotor via brush and commutator for the mechanical work.
- Interaction of field flux and armature current in the rotor produces torque.

3.1 OPERATION:

• When a separately excited motor is excited by a field current of i_f and an armature current of i_a flows in the circuit, the motor develops a back emf and a torque to balance the load torque at a particular speed.

• The i_f is independent of the i_a. Each windings are supplied separately. Any change in the armature current has no effect on the field current.

• The i_f is normally much less than the i_a.

3.1.1 FIELD AND ARMATURE EQUATIONS:

Instantaneous field current:

$$v_f = R_f i_f + L_f \frac{di_f}{dt}$$
where R_f and L_f are the field resistor and inductor, respectively

Instantaneous armature current :

$$v_a = R_a i_a + L_a \frac{di_a}{dt} + e_g$$
where R_f and L_f are the armature resistor and inductor, respectively.

The motor back emf, which is also known as speed voltage, is expressed as :

$$e_g = K_v \omega i_f$$
K_v is the motor voltage constant (in V/A - rad/s) and ω is the motor speed (in rad/sec)

3.1.2 BASIC TORQUE EQUATION:

The torque develped by the motor is :

$$T_d = K_t i_f i_a$$
where ($K_t = K_v$) is the torque constant. (in V/A - rad/s)

Sometimes it is written as :

$$T_d = K_t \phi i_a$$

For normal operation, the developed torque must be equal to the load torque plus the friction and inertia, i.e. :

$$T_d = J \frac{d\omega}{dt} + B\omega + T_L$$
where

B : viscous friction constant, (N.m/rad/s)

T_L : load torque (N.m)

J : inertia of the motor (kg.m^2)

3.2 STEADY STATE OPERATION:

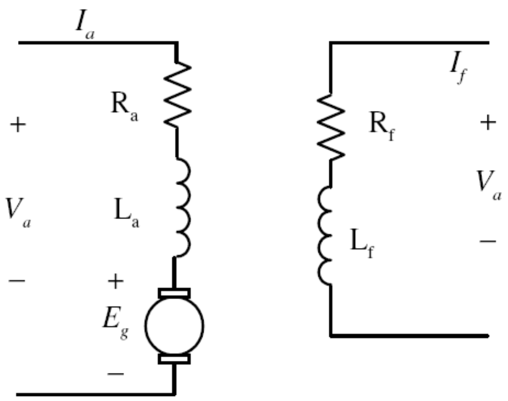

Figure 8: Separately Excited DC Motor In Steady State

Under steady - state operations, time derivatives is zero. Assuming the motor is not saturated.

For field circuit,

$V_f = I_f R_f$

The back emf is given by :

$E_g = K_v \omega I_f$

The armature circuit

$V_a = I_a R_a + E_g = I_a R_a + K_v \omega I_f$

3.2.1 STEADY-STATE TORQUE AND SPEED:

The motor speed can be easily derived:

$$\omega = \frac{V_a - I_a R_a}{K_v I_f}$$

If R_a is a small value (which is usual), or when the motor is lightly loaded, i.e. I_a is small,

$$\omega = \frac{V_a}{K_v I_f}$$

That is if the field current is kept constant, the motor speed depends only on the supply voltage.

The developed torque is:

$$T_d = K_t I_f I_a = B\omega + T_L$$

The required power is:

$$P_d = T_d \omega$$

3.2.2 TORQUE AND SPEED CONTROL:

• From the derivation, several important facts can be deduced for steady-state operation of DC motor.

• For a fixed field current, or flux (I_f), the torque demand can be satisfied by varying the armature current (I_a).

• The motor speed can be varied by:

– controlling V_a (voltage control)

– controlling V_f (field control)

• These observations lead to the application of variable DC voltage for controlling the speed and torque of DC motor.

3.2.3 VARIABLE SPEED OPERATION:

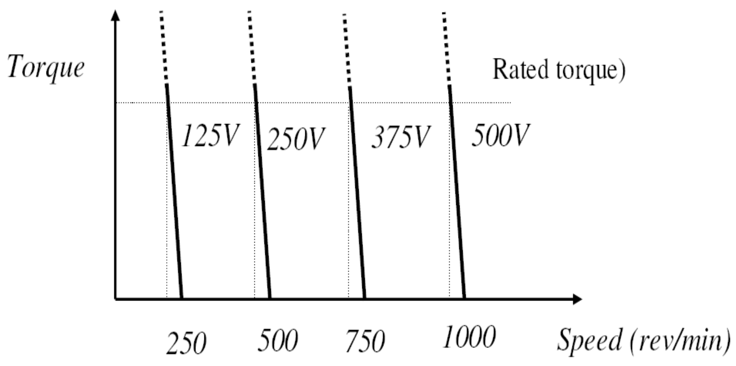

Figure 9: Torque Vs Speed Characteristic For Different Armature Voltages

• Family of steady-state torque speed curves for a range of armature voltage can be drawn as above.

• The speed of DC motor can simply be set by applying the correct voltage.

• Note that speed variation from no-load to full load (rated) can be quite small. It depends on the armature resistance.

3.2.4 BASE SPEED AND FIELD-WEAKENING:

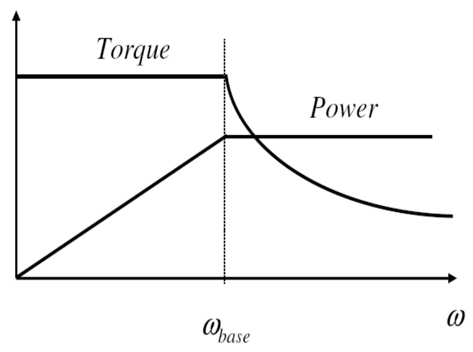

Figure 10: Torque Vs Speed And Power Vs Speed Characteristic Of Separately Excited DC Motor

• **Base speed**: (w_{base})

– the speed which correspond to the rated V_a, rated I_a and rated I_f.

• **Constant Torque** region (w > w_{base})

– I_a and I_f are maintained constant to met torque demand. V_a is varied to control the speed. Power increases with speed.

• **Constant Power** region (w > w_{base})

– V_a is maintained at the rated value and i_f is reduced to increase speed. However, the power developed by the motor (= torque x speed) remains constant. This phenomenon is known as ***field weakening.***

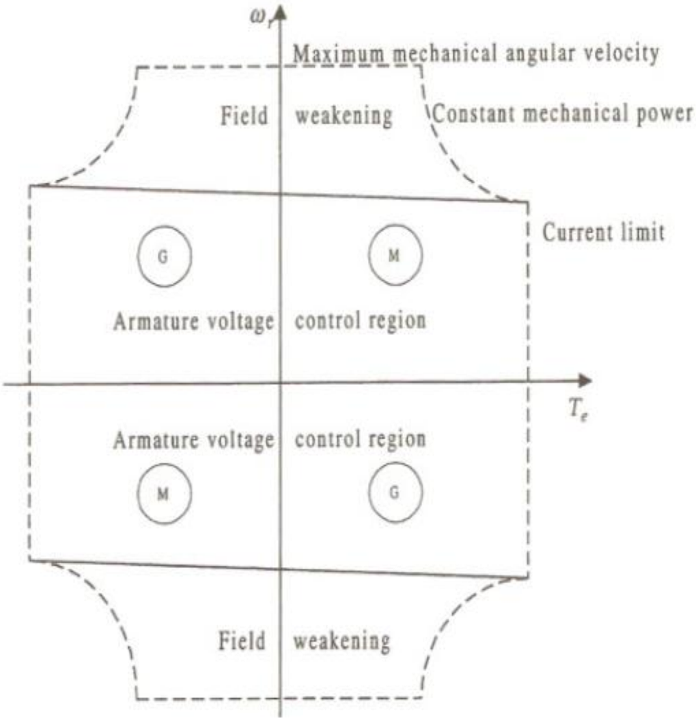

Figure 11: Typical Operating Regions Of Separately Excited DC Machines

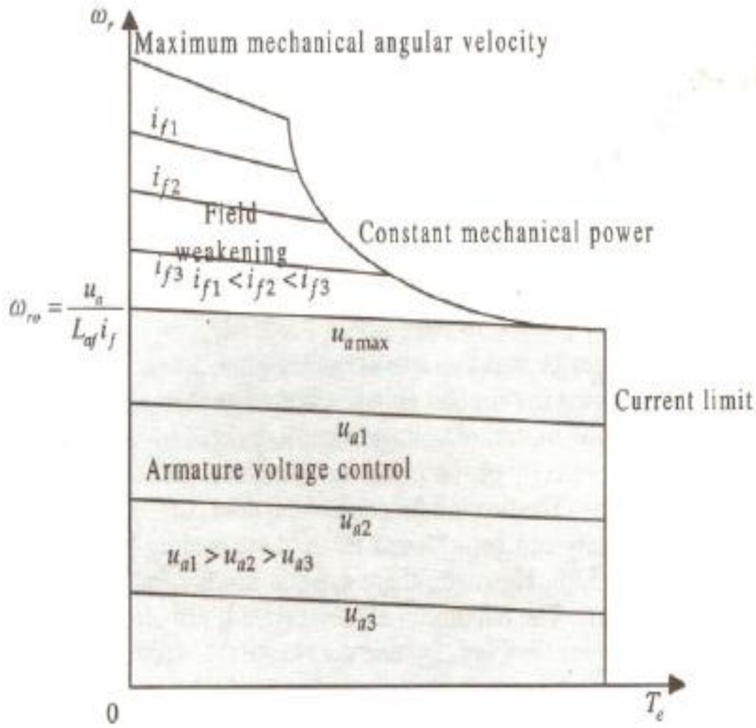

Figure 12: Steady-State Torque-Speed Operating Characteristics Of Separately Excited DC Motors

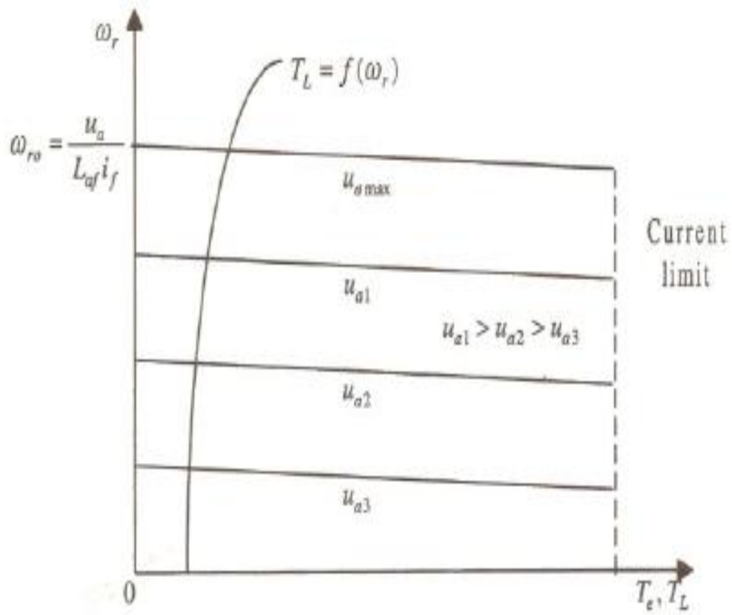

Figure 13: Torque-Speed Characteristic Curves For I_f=Const.

A high performance drive system consists of a motor, a converter, and a controller integrated to perform a precise mechanical manoeuvre. This requires the shaft speed and/or the motor to closely follow a specified trajectory regardless of unknown load variations and other parameter uncertainties. One way to compensate for these uncertainties is to identify the motor/load dynamics through the parameters of a pre-defined model. These model parameters can then be manipulated using different control strategies to come up with the controller design.

If the unknown dynamics are non-stationary, the identification has to be performed online. However, if the dynamics are stationary, offline identification is adequate.

e.g. in self-tuning control, motor/load parameters are identified through a linear parametric (ARMAX) model using a Kalman Filter.

But identifying non-linear dynamics through a linear model does not guarantee an accurate functional representation.

The ability of a multi-layer perceptron type neural network to learn a large class of non-linear functions is well known. The neural network identifier evolves through the learning of a suitable time sequence of input/output patterns generated by the motor model. The ability to successfully train without explicit knowledge of the motor/load dynamics is the key advantage in this type of identification. Moreover the superior generalizing capability of the neural network allows accurate emulation of the motor dynamics for previously untrained inputs. The inherent noise rejection property of the neural network is useful in dealing with noisy input/output characteristics.

Chapter 4

OBJECTIVE

The neural network's role is to identify the unknown mapping between the voltage command and the speed of the motor. The trained neural network is then used for controller design.

4.1 SYSTEM DYNAMICS:

$$K\omega(t) = - R_a I_a(t) - L_a [di_a(t)/dt] + V_t(t) \qquad (1)$$

$$K i_a(t) = J[d\omega(t)/dt] + D\omega(t) + T_L(t) \qquad (2)$$

Where,

$\omega(t)$ - rotor speed (rad/s)

$V_t(t)$ - terminal voltage (V)

$I_a(t)$ - armature current (A)

$T_L(t)$ - load torque (Nm)

J - Rotor inertia (Nm^2)

K - Torque & back emf constant (Nm/A)

D - viscose friction coefficient (Nms)

R_a - armature resistance (Ω)

L_a - armature inductance (H)

4.2 CONTROL SYSTEM APPLICATIONS:

- Predictive control method
- NARMA-L2 control method
- Model reference control method.

All are based on standard linear control architecture.

Two steps involved when used for control:

- System identification

- Control design
- **SI stage**, a neural network model of the plant that we want to control developed.
- **Control design stage**, the neural network plant model used to train the controller.

4.3 SYSTEM IDENTIFICATION:

It is to train a neural network to represent the forward dynamics of the plant. The prediction error between the plant output and the neural network output is used as the neural network training signal.

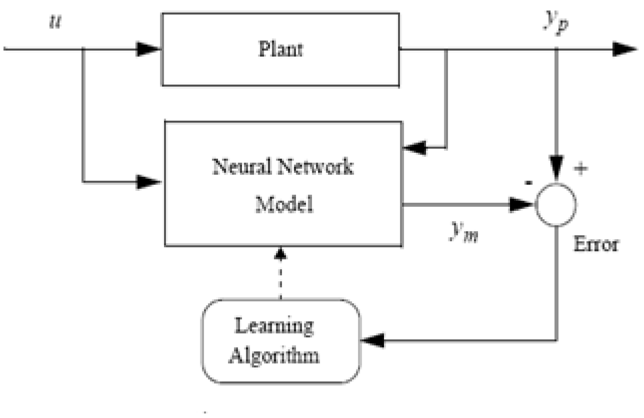

Figure 14: Plant Identification

The standard model used for non-linear identification in **NARMA** model.

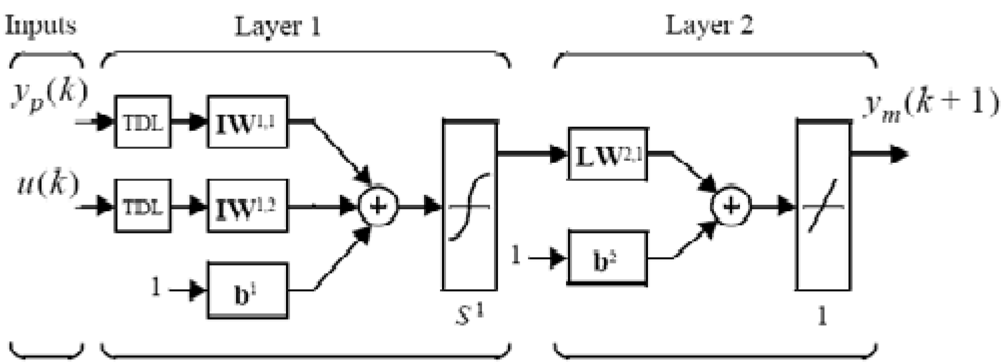

Figure 15: Neural Network Plant Model

TDL- Tapped Delay Lines that stores previous values of the input signal

IW (i, j) - Weight matrix from layer number ‘i’ to ‘j’

IW (j, i) - Weight matrix from layer number ‘j’ to ‘i’

4.4 CONTROL DESIGN:

We use Model reference Adaptive Control (MRAC) strategy. This architecture consists of two neural networks;

- Controller network
- Plant model network

After SI, the controller is trained so that the plant output follows the reference model output. The controller training is done separately and requires the use of dynamic back-propagation. DBP gets applied to a larger class of plant and hence the controller requires minimal online computation. Generally indirect MRAC is preferred.

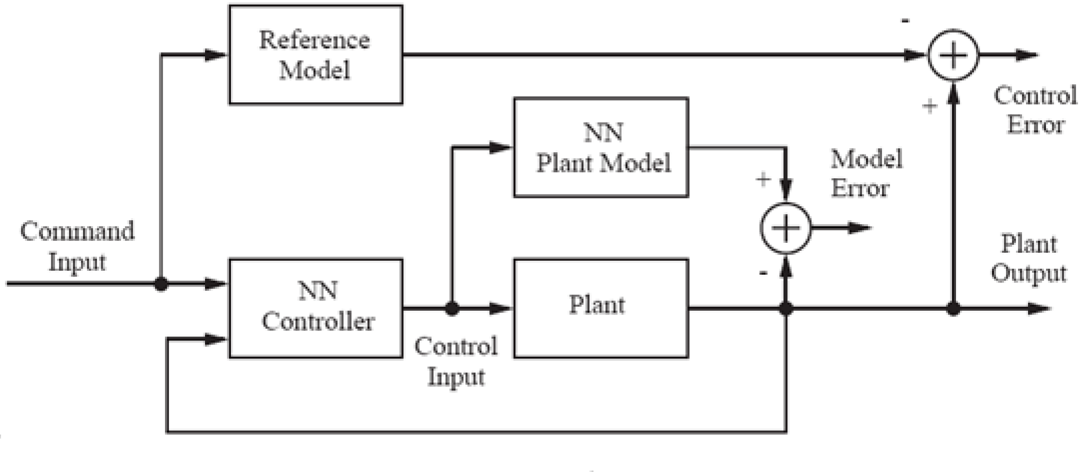

Figure 16: Model Reference Control Architecture

4.5 DISCRETE TIME DC MOTOR MODEL:

$\mathbf{T_L(t) = \mu W_P^2(t)\ [sign(W_p(t)]}$ **;**

where **μ** is a constant.

The discrete time model speed equation governing the system dynamics is given by;

$$W_p(k+1) = \alpha\, W_p(k) + \beta\, W_p(k-1) + \gamma\, \left[sign\left(W_p(k)\right)\right] W_p^2(k) + \delta \left[sign\left(W_p(k-1)\right)\right] W_p^2(k-1) + \xi\, V_t(k)\ ;$$

4.6 IDENTIFICATION OF DC MOTOR MODEL:

The equation can be manipulated to the form;

$$\mathbf{V_t(k) = g[W_p(k+1), W_p(k), W_p(k-1)]}$$

where the function g[.] is given by;

$$\mathbf{g[W_p(k+1),\ W_p(k),\ W_p(k-1)] = \{\ W_p(k+1) - \alpha\ W_p(k) - \beta\ W_p(k-1) - \gamma\ [sign\ (W_p(k))]\ W_p^2(k) - \delta\ [sign\ (W_p(k-1))]\ W_p^2(k-1)\ \}/\ \xi\ ;}$$

and is assumed to be unknown. An ANN is trained to emulate the unknown function **g[.]**. However as $\mathbf{W_p(k+1)}$ cannot be readily available.

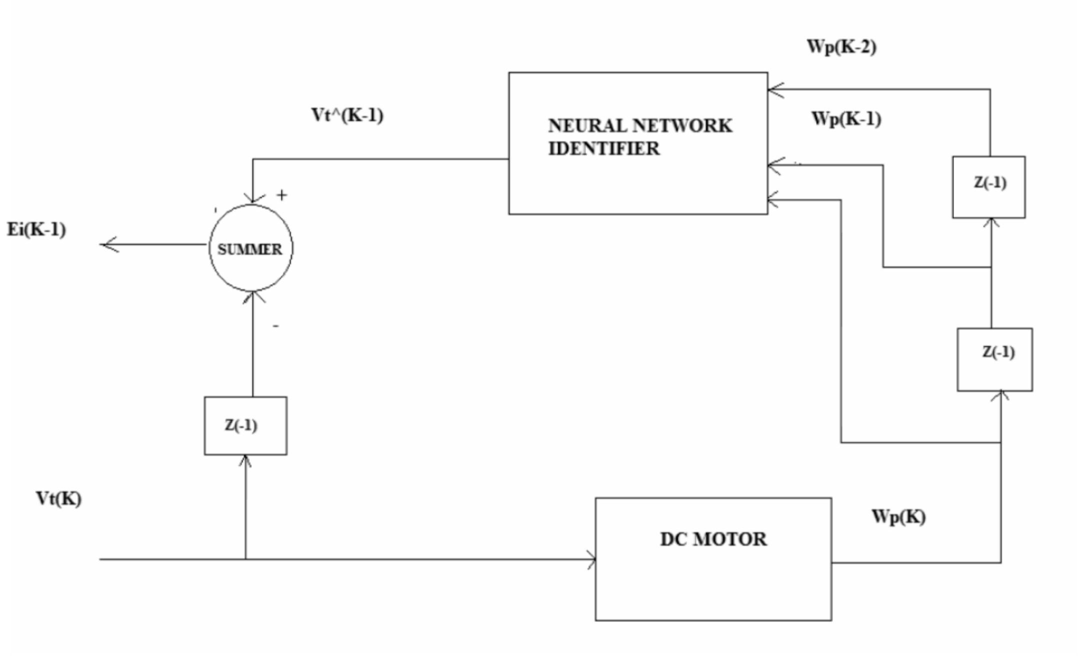

Figure 17: Structure Of The ANN Identifier

4.7 TRAJECTORY CONTROL OF DC MOTOR USING TRAINED ANN:

The objective of the control system is to drive the motor so that its speed $\mathbf{W_p(k)}$, follows a pre-specified trajectory, $\mathbf{W_m(k)}$**.** The following second order reference model is chosen;

$$\mathbf{W_m(k+1) = 0.6\ W_m(k) + 0.2\ W_m(k-1) + r(k)\ ;}$$

For a given desired sequence $\{\mathbf{W_m(k)}\}$, the corresponding control sequence $\{\mathbf{r(k)}\}$ can be calculated using the above relation.

The speed at (k+1)th time step can be predicted from;

$\widehat{\boldsymbol{W_P}}(\boldsymbol{k+1})$ **= 0.6** $\mathbf{W_p(k)}$ **+ 0.2** $\mathbf{W_p(k\text{-}1)}$ **+ r(k) ;**

This result can be fed to the ANN to estimate the control input at kth time step using;

$\widehat{\boldsymbol{V_t}}(\boldsymbol{k-1})$ **= N** $\mathbf{[W_p(k), W_p(k\text{-}1), W_p(k\text{-}2)]}$ **;**

Chapter 5

PROBLEM STATEMENT

A separately excited dc motor with name plate ratings of 1 hp, 220V, 550 rpm is used in all simulations. Following parameter values are associated with it.

$J = 0.068\ Kgm^2$

$K = 3.475\ Nm\ A^{-1}$

$R_a = 7.56\ \Omega$

$L_a = 0.055\ H$

$D = 0.03475\ Nms$

$\mu = 0.0039\ Nms^2$

$T = 40ms$

As from the system dynamics equations, $\alpha, \beta\ and\ \xi$ are constant values based on the motor parameters J, K, D, R_a, L_a and the sampling period T. While $\gamma\ and\ \delta$ in addition to being functions of the above parameters are also functions of μ**.** The value k denotes the k^th time step.

α = 2*(L_a J + R_a J + L_a D – R_a D T – K^2 T)/(L_a J + 2*R_a J + 2*L_a D) ;

β = (-) L_a J/(L_a J + 2*R_a J + 2*L_a D) ;

γ = (-) 2*μ (L_a + R_a T)/(L_a J + 2*R_a J + 2*L_a D) ;

δ = 2*L_a μ/(L_a J + 2*R_a J + 2*L_a D) ;

ξ = 2*K*T/(L_a J + 2*R_a J + 2*L_a D) ;

Using the given values of the system parameters in the above stated equations we get;

α = 0.0506 ;

β = (-)0.003611 ;

γ = (-)0.002692 ;

δ = 0.000414 ;

ξ = 0.26795 ;

The system equation becomes;

$$(K^2 + R_a D)\, W(t) = K\, V_t(t) - J\, L_a \frac{d^2W}{dt^2} - (R_a J + L_a D) \frac{dW}{dt} - L_a \frac{dT_L}{dt} - R_a T_L(t) \;;$$

LIMITS TAKEN INTO CONSIDERATION :

$-30.0 < W_P(k) < 30.0$ rad/s

$-1.0 < W_p(k-1) - W_p(k) < 1.0$ rad/s

-100 v $< V_t(k) < 100$ v

Chapter 6

GENERATION OF THE SIGNAL

6.1 OVERVIEW:

$\widehat{W_P}(k+1) =$ N [$W_p(k)$, $W_p(k-1)$] + ξ $V_t(k)$;

So, now the system dynamic equations get converted into two first order differential equations as follows :

$\dot{i_a}(t) = (V_t(t) / L_a) – (R_a\, i_a(t) / L_a) – (K\, W(t) / L_a)$;

$\dot{W}(t) = (K\, i_a(t) / J) – (D\, W(t) / J) – (T_L(t) / J)$;

TAKING INITIAL VALUES AS,

$W_p(0) = W(0) = 0$;

$i_a(0) = 0$;

Substituting the values of the motor parameters in the above two equations along with the initial conditions, the above two equations get transformed into:

$\dot{y_1} = 18.182\, V_t – 137.454\, y_1 – 63.182\, y_2$;

$\dot{y_2} = 51.1\, y_1 – 0.511\, y_2 – 14.7\, T_L$;

Considering **$y_1 = i_a(t)$ and $y_2 = W(t)$**

$V_t(k) = 50 \sin(2\pi kT/7) + 45 \sin(2\pi kT/3) \;\; \forall\, kT \in [0, 20]$ is assumed.

6.2 RESULT :

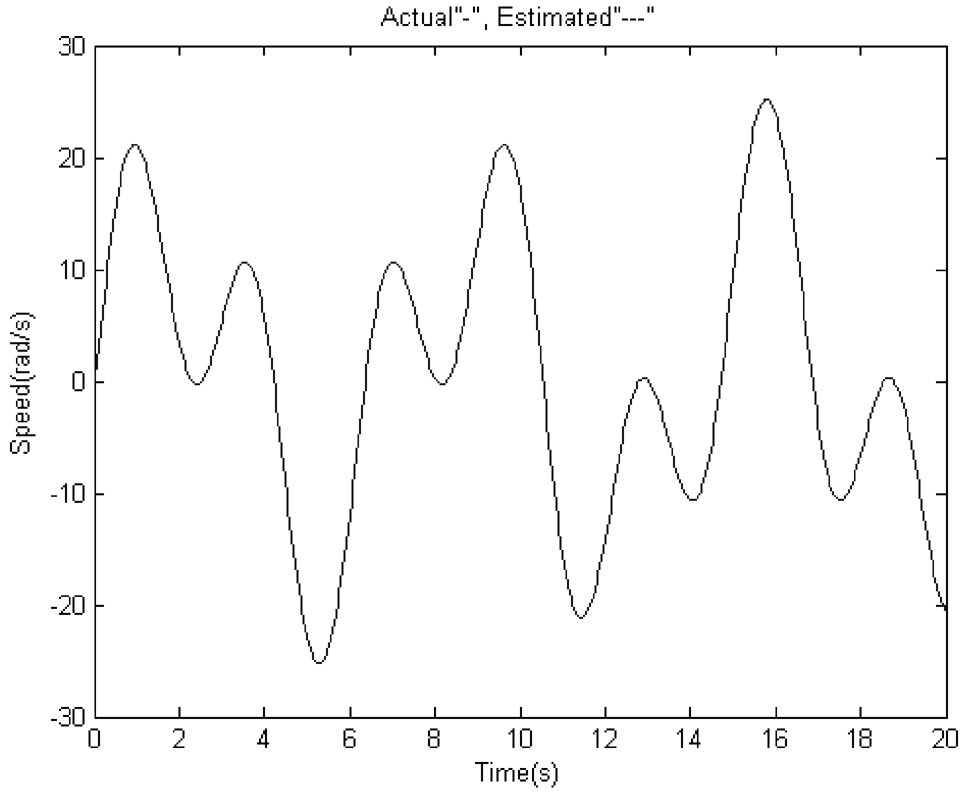

Figure 18: Actual And Estimated Rotor Speeds

Chapter 7

SYSTEM IDENTIFICATION

7.1 OVERVIEW:

The 3-layer feed-forward ANN used :

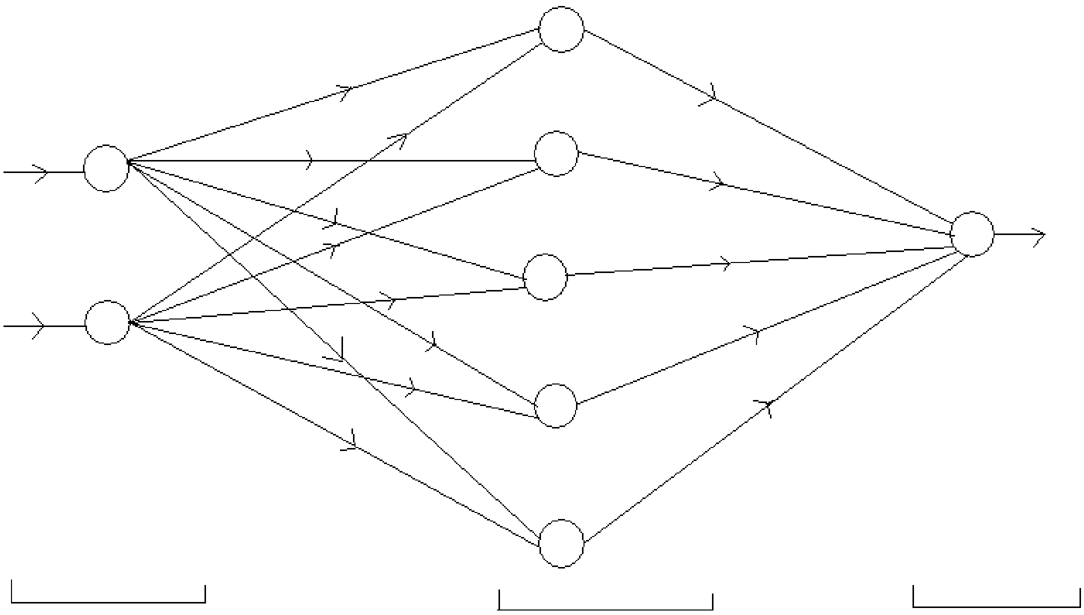

Figure 19: Three-Layer Feed forward ANN

As discussed earlier, the motor dynamics can be partitioned as;

$W_p(k+1) = f[\ W_p(k), W_p(k-1)\ ] + \xi\ V_t(k)$;

Where the function f[.] is given by;

$$\mathbf{f[\ W_p(k), W_p(k\text{-}1)\ ]} = \boldsymbol{\alpha\, W_p(k) + \ \beta\, W_p(k-1) + \ \gamma\ \left[sign\ \left(W_p(k)\right)\right] W_p^2(k) +}$$

$$\boldsymbol{\delta\ \left[sign\ \left(W_p(k-1)\right)\right]\ W_p^2(k-1)\ ;}$$

which is supposed to be an **UNKNOWN NON-LINEAR FUNCTION**.

The values $W_p(k)$ and $W_p(k - 1)$ which are the independent variables of f [.], are selected as the inputs to the ANN. The corresponding *ANN* output target f [$W_p(k)$, $W_p(k\text{-}1)$] is given by above equation.

The *ANN* is trained off-line using randomly generated input patterns of [$W_p(k)$, $W_p(k-1)$] and the corresponding target f [$W_p(k)$, $W_p(k-1)$]. The choice of $W_p(k)$, $W_p(k-1)$ has to satisfy the constraints as specified earlier.

The motor speed is estimated by the trained *ANN* predictor as

$\widehat{W_P}(k+1) =$ N [$W_p(k)$, $W_p(k-1)$] + ξ $V_t(k)$;

Where N[.] denotes the ANN output for a given set of "." Inputs.

ANN topology and the training effort are briefly described by the following statistics:

Number of inputs	2
Number of outputs	1
Number of Hidden layers	1
Number of Hidden neurons	5
Number of training patterns 'P'	500
Number of training sweeps	1000
Learning Step 'η'	0.1
Momentum Gain 'α'	0.1
E_{total} threshold 'ε'	0.04

As mentioned earlier, except for the number of inputs/outputs of the *ANN*, all other design and learning parameters are selected by trial and error.

7.2 BLOCK DIAGRAM :

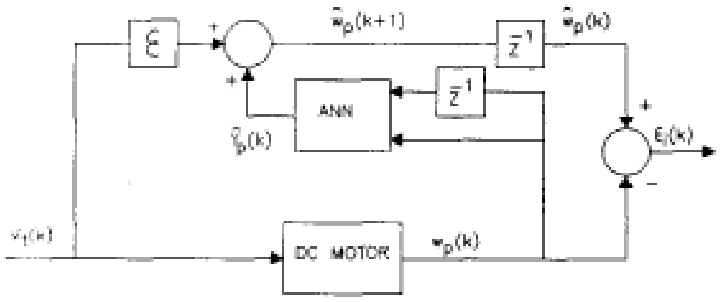

Figure 20: Structure Of The ANN For Identification Of The Dc Motor

The trained *ANN* is applied as a series-parallel type identifier to estimate the value of the function f [.]. $\mathbf{z^{-1}}$ in any figure indicates a unit time delay. The performance of the trained *ANN* identifier is evaluated by comparing the actual and estimated speeds as calculated from the afore mentioned equations above for the following arbitrarily selected terminal voltage sequence,

$V_t(k) = 50\sin(2\pi kT/7) + 45\sin(2\pi kT/3) \ \ \forall\, kT \ \in [0, 20]$.

7.3 C-CODE FOR TRAINING WITH BACK-PROPAGATION OF THE ANN:

A single neuron (i.e. processing unit) takes it total input *In* and produces an output activation *Out*. I shall take this to be the sigmoid function

```
Out = 1.0/(1.0 + exp(-In));        /* Out = Sigmoid(In) */
```

though other activation functions are often used (e.g. linear or hyperbolic tangent). This has the effect of squashing the infinite range of *In* into the range 0 to 1. It also has the convenient property that its derivative takes the particularly simple form

```
Sigmoid_Derivative = Sigmoid * (1.0 - Sigmoid) ;
```

Typically, the input *In* into a given neuron will be the weighted sum of output activations feeding in from a number of other neurons. It is convenient to think of the activations flowing through layers of neurons. So, if there are *NumUnits1* neurons in layer 1, the total activation flowing into our layer 2 neuron is just the sum over *Layer1Out[i]*Weight[i]*, where *Weight[i]* is the strength/weight of the connection between unit *i* in layer 1 and our unit in layer 2. Each neuron will also have a bias, or resting state, that is added to the sum of inputs, and it is convenient to call this *weight[0]*. We can then write

```
Layer2In=Weight[0];                /* start with the bias */
for( i = 1 ; i <= NumUnits1 ; i++ ) {        /* i loop over layer 1 units */
        Layer2In += Layer1Out[i] * Weight[i] ;      /* add in weighted contributions
        from layer 1 */
}
Layer2Out = 1.0/(1.0 + exp(-Layer2In)) ;    /* compute sigmoid to give activation */
```

Normally layer 2 will have many units as well, so it is appropriate to write the weights between unit *i* in layer 1 and unit *j* in layer 2 as an array *Weight[i][j]*. Thus to get the output of unit *j* in layer 2 we have

```
Layer2In[j]=Weight[0][j];
for( i = 1 ; i <= NumUnits1 ; i++ ) {
        Layer2In[j] += Layer1Out[i] * Weight[i][j] ;
}
Layer2Out[j] = 1.0/(1.0 + exp(-Layer2In[j])) ;
```

Remember that in C the array indices start from zero, not one, so we would declare our variables as

```
double Layer1Out[NumUnits1+1] ;
double Layer2In[NumUnits2+1] ;
double Layer2Out[NumUnits2+1] ;
double Weight[NumUnits1+1][NumUnits2+1] ;
```

(or, more likely, declare pointers and use *calloc* or *malloc* to allocate the memory). Naturally, we need another loop to get all the layer 2 outputs

```
for( j = 1 ; j <= NumUnits2 ; j++ ) {
    Layer2In[j] = Weight[0][j] ;
    for( i = 1 ; i <= NumUnits1 ; i++ ) {
        Layer2In[j] += Layer1Out[i] * Weight[i][j] ;
    }
    Layer2Out[j] = 1.0/(1.0 + exp(-Layer2In[j])) ;
}
```

Three layer networks are necessary and sufficient for most purposes, so our layer 2 outputs feed into a third layer in the same way as above

```
for( j = 1 ; j <= NumUnits2 ; j++ ) {        /* j loop computes layer 2 activations */
    Layer2In[j] = Weight12[0][j] ;
    for( i = 1 ; i <= NumUnits1 ; i++ ) {
        Layer2In[j] += Layer1Out[i] * Weight12[i][j] ;
    }
    Layer2Out[j] = 1.0/(1.0 + exp(-Layer2In[j])) ;
}
for( k = 1 ; k <= NumUnits3 ; k++ ) {        /* k loop computes layer 3 activations */
```

```
        Layer3In[k] = Weight23[0][k] ;
        for( j = 1 ; j <= NumUnits2 ; j++ ) {
            Layer3In[k] += Layer2Out[j] * Weight23[j][k] ;
        }
        Layer3Out[k] = 1.0/(1.0 + exp(-Layer3In[k])) ;
    }
```

The code can start to become confusing at this point - I find that keeping a separate index *i*, *j*, *k* for each layer helps, as does an intuitive notation for distinguishing between the different layers of weights *Weight12* and *Weight23*. For obvious reasons, for three layer networks, it is traditional to call layer 1 the *Input* layer, layer 2 the *Hidden* layer, and layer 3 the *Output* layer.

Also, to save getting all the *In*'s and *Out*'s confused, we can write *LayerNIn* as *SumN*. Our code can thus be written

```
    for( j = 1 ; j <= NumHidden ; j++ ) {        /* j loop computes hidden unit activations
    */
        SumH[j] = WeightIH[0][j] ;
        for( i = 1 ; i <= NumInput ; i++ ) {
            SumH[j] += Input[i] * WeightIH[i][j] ;
        }
        Hidden[j] = 1.0/(1.0 + exp(-SumH[j])) ;
```

```
}
for( k = 1 ; k <= NumOutput ; k++ ) {        /* k loop computes output unit activations
*/
        SumO[k] = WeightHO[0][k] ;
        for( j = 1 ; j <= NumHidden ; j++ ) {
                SumO[k] += Hidden[j] * WeightHO[j][k] ;
        }
        Output[k] = 1.0/(1.0 + exp(-SumO[k])) ;
}
```

Generally we will have a whole set of *NumPattern* training patterns, i.e. pairs of input and target output vectors,

```
Input[p][i] , Target[p][k]
```

labelled by the index *p*. The network learns by minimizing some measure of the error of the network's actual outputs compared with the target outputs. For example, the sum squared error overall output unit's *k* and all training patterns *p* will be given by

```
Error = 0.0 ;
for( p = 1 ; p <= NumPattern ; p++ ) {
        for( k = 1 ; k <= NumOutput ; k++ ) {
                Error += 0.5 * (Target[p][k] - Output[p][k]) * (Target[p][k] -
                Output[p][k]) ;
        }
}
```

(The factor of 0.5 is conventionally included to simplify the algebra in deriving the learning algorithm.) If we insert the above code for computing the network outputs into the *p* loop of this, we end up with

```
Error = 0.0 ;
for( p = 1 ; p <= NumPattern ; p++ ) {        /* p loop over training patterns */
    for( j = 1 ; j <= NumHidden ; j++ ) {        /* j loop over hidden units */
        SumH[p][j] = WeightIH[0][j] ;
        for( i = 1 ; i <= NumInput ; i++ ) {
            SumH[p][j] += Input[p][i] * WeightIH[i][j] ;
        }
        Hidden[p][j] = 1.0/(1.0 + exp(-SumH[p][j])) ;
    }
    for( k = 1 ; k <= NumOutput ; k++ ) {        /* k loop over output units */
        SumO[p][k] = WeightHO[0][k] ;
        for( j = 1 ; j <= NumHidden ; j++ ) {
            SumO[p][k] += Hidden[p][j] * WeightHO[j][k] ;
        }
        Output[p][k] = 1.0/(1.0 + exp(-SumO[p][k])) ;
        Error += 0.5 * (Target[p][k] - Output[p][k]) * (Target[p][k] -
        Output[p][k]) ;     /* Sum Squared Error */
    }
}
```

The next stage is to iteratively adjust the weights to minimize the network's error. A standard way to do this is by 'gradient descent' on the error function. We can compute how much the error is changed by a small change in each weight (i.e. compute the partial derivatives d*Error*/d*Weight*) and shift the weights by a small amount in the direction that reduces the error.

```
Error = 0.0 ;
for( p = 1 ; p <= NumPattern ; p++ ) {        /* repeat for all the training patterns */
        for( j = 1 ; j <= NumHidden ; j++ ) {        /* compute hidden unit activations */
                SumH[p][j] = WeightIH[0][j] ;
                for( i = 1 ; i <= NumInput ; i++ ) {
                        SumH[p][j] += Input[p][i] * WeightIH[i][j] ;
                }
                Hidden[p][j] = 1.0/(1.0 + exp(-SumH[p][j])) ;
        }
        for( k = 1 ; k <= NumOutput ; k++ ) {        /* compute output unit activations and
        errors */
                SumO[p][k] = WeightHO[0][k] ;
                for( j = 1 ; j <= NumHidden ; j++ ) {
                        SumO[p][k] += Hidden[p][j] * WeightHO[j][k] ;
                }
                Output[p][k] = 1.0/(1.0 + exp(-SumO[p][k])) ;
                Error += 0.5 * (Target[p][k] - Output[p][k]) * (Target[p][k] - Output[p][k]) ;
                DeltaO[k] = (Target[p][k] - Output[p][k]) * Output[p][k] * (1.0 -
                Output[p][k]) ;
        }
        for( j = 1 ; j <= NumHidden ; j++ ) {        /* 'back-propagate' errors to hidden layer */
```

```
            SumDOW[j] = 0.0 ;
            for( k = 1 ; k <= NumOutput ; k++ ) {
                SumDOW[j] += WeightHO[j][k] * DeltaO[k] ;
            }
            DeltaH[j] = SumDOW[j] * Hidden[p][j] * (1.0 - Hidden[p][j]) ;
        }
        for( j = 1 ; j <= NumHidden ; j++ ) {       /* update weights WeightIH */
            DeltaWeightIH[0][j] = eta * DeltaH[j] + alpha * DeltaWeightIH[0][j] ;
            WeightIH[0][j] += DeltaWeightIH[0][j] ;
            for( i = 1 ; i <= NumInput ; i++ ) {
                DeltaWeightIH[i][j] = eta * Input[p][i] * DeltaH[j] + alpha *
                DeltaWeightIH[i][j];
                WeightIH[i][j] += DeltaWeightIH[i][j] ;
            }
        }
        for( k = 1 ; k <= NumOutput ; k ++ ) {      /* update weights WeightHO */
            DeltaWeightHO[0][k] = eta * DeltaO[k] + alpha * DeltaWeightHO[0][k] ;
            WeightHO[0][k] += DeltaWeightHO[0][k] ;
            for( j = 1 ; j <= NumHidden ; j++ ) {
                DeltaWeightHO[j][k] = eta * Hidden[p][j] * DeltaO[k] + alpha *
                DeltaWeightHO[j][k]                                            ;
                WeightHO[j][k] += DeltaWeightHO[j][k] ;
            }
        }

    }
```

he weight changes *DeltaWeightIH* and *DeltaWeightHO* are each made up of two components. First, the *eta* component that is the gradient descent contribution. Second, the *alpha* component that is a 'momentum' term which effectively keeps a moving average of the gradient descent weight change contributions, and thus smoothes out the overall weight changes. Fixing good values of the learning parameters *eta* and *alpha* is usually a matter of trial and error. Certainly *alpha* must be in the range 0 to 1, and a non-zero value does usually speed up learning. Finding a good value for *eta* will depend on the problem, and also on the value chosen for *alpha*. If it is set too low, the training will be unnecessarily slow. Having it too large will cause the weight changes to oscillate wildly, and can slow down or even prevent learning altogether.

The complete training process will consist of repeating the above weight updates for a number of epochs (using another *for* loop) until some error crierion is met, for example the *Error* falls below some chosen small number.

```
for( epoch = 1 ; epoch < LARGENUMBER ; epoch++ ) {
        /*    ABOVE    CODE    FOR    ONE    ITERATION    */
        if( Error < SMALLNUMBER ) break ;
}
```

If the training patterns are presented in the same systematic order during each epoch, it is possible for weight oscillations to occur. It is therefore generally a good idea to use a new random order for the training patterns for each epoch. If we put the *NumPattern* training pattern indices *p* in random order into an array *ranpat[]*, then it is simply a matter of replacing our training pattern loop

```
for( p = 1 ; p <= NumPattern ; p++ ) {
```

with

```
for( np = 1 ; np <= NumPattern ; np++ ) {
        p = ranpat[np] ;
```

Generating the random array *ranpat[]* is not quite so simple, but the following code will do the job

```
for( p = 1 ; p <= NumPattern ; p++ ) {        /* set up ordered array */
        ranpat[p] = p ;
}
for( p = 1 ; p <= NumPattern ; p++) {        /* swap random elements into each position */
        np = p + rando() * ( NumPattern + 1 - p ) ;
        op = ranpat[p] ; ranpat[p] = ranpat[np] ; ranpat[np] = op ;
}
```

Naturally, one must set some initial network weights to start the learning process. Starting all the weights at zero is generally not a good idea, as that is often a local minimum of the error function. It is normal to initialize all the weights with small random values. If *rando()* is your favourite random number generator function that returns a flat distribution of random numbers in the range 0 to 1, and *smallwt* is the maximum absolute size of your initial weights, then an appropriate section of weight initialization code would be

```
for( j = 1 ; j <= NumHidden ; j++ ) {        /* initialize WeightIH and DeltaWeightIH */
```

```
        for( i = 0 ; i <= NumInput ; i++ ) {
            DeltaWeightIH[i][j] = 0.0 ;
            WeightIH[i][j] = 2.0 * ( rando() - 0.5 ) * smallwt ;
        }
}
for( k = 1 ; k <= NumOutput ; k ++ ) {     /* initialize WeightHO and
DeltaWeightHO */
        for( j = 0 ; j <= NumHidden ; j++ ) {
            DeltaWeightHO[j][k] = 0.0 ;
            WeightHO[j][k] = 2.0 * ( rando() - 0.5 ) * smallwt ;
        }
}
```

Chapter 8

TRAJECTORY CONTROL

8.1 OVERVIEW:

The objective of the control system is to drive the motor so that its speed, $\mathbf{W_p(k)}$**,** follows a prespecified trajectory,$\mathbf{W_m(k)}$. This is done by letting the dc motor follow the output of a selected reference model throughout the trajectory. The following second order reference model is selected;

$$\mathbf{W_m(k+1) = 0.6\, W_m(k) + 0.2\, W_m(k-1) + r(k)\, ;}$$

r(k) is the bounded input to the reference model. The coefficients are selected to ensure that its poles are within the unit circle and has the type of response that can be achieved by the dc motor. For a given desired sequence ($W_m(k)$) (trajectory), the corresponding control sequence (r(k)) can be calculated using the above equation.

The controller topology trained in the previous chapter is now used to estimate the motor terminal voltage $V_t(k)$ which enables accurate trajectory control of the shaft speed $W_p(k)$. Performance of the controller is simulated for arbitrarily selected speed tracks ($W_m(k)$). A graphical comparison of the specified and actual speed trajectories are hence presented.

Let's for a moment assume that the tracking error $\mathbf{E_c(k)}$ is zero, and that the nonlinear function **f[.]** is known. The control input $\mathbf{V_t(k)}$ to the motor at the $\mathbf{k^{th}}$ time step can be calculated as

$$V_t(k) = [-f[W_p(k), W_p(k-1)] + 0.6*W_p(k) + 0.2*W_p(k-1) + r(k)]/\xi\ ;$$

Hence the tracking error difference equation becomes

$$E_c(k+1) = 0.6*E_c(k) + 0.2*E_c(k-1)\ ;$$

Since the reference model is asymptotically stable, it follows that $\mathbf{lim}_{k\to\infty}\, \boldsymbol{E_c}\mathbf{(k+l) = 0}$ for arbitrary initial conditions. However since f[.**]** is not **known,** its estimated value from the *ANN* trained under the controller topology is used to estimate the controlled terminal voltage.

$\widehat{V_t}$(k) = [-N[W_p(k), W_p(k-1)] + 0.6*W_p(k) + 0.2*W_p(k-1) + r(k)]/ξ ;

The tracking control capability of the model was investigated for different arbitrarily specified trajectories. Only a specific result is shown for brevity. In this case, the

specified speed trajectory is defined by

W_m(k) = 10*Sin(2.0*πkT/4) + 16*Sin(2.0*πkT/7) $\forall$ $kT \in [0,20]$;

For the above trajectory, the corresponding (r(k)) is derived by using the first equation. This is applied to the model shown below. The matrix α_m^T corresponds to the reference model coefficients [0.6 0.2].

8.3 PLOTS:

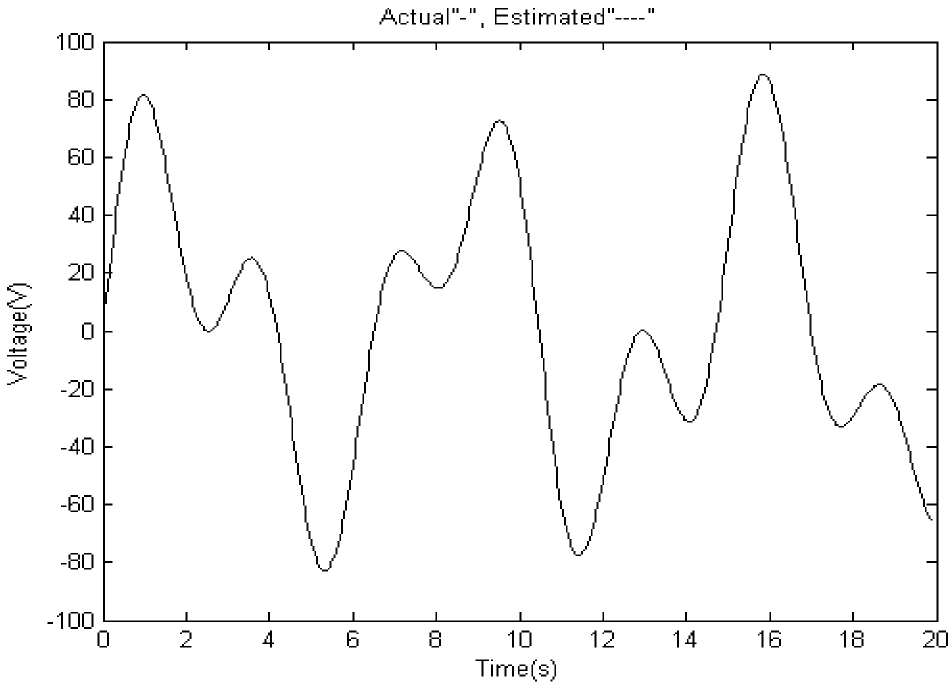

Figure 21: Actual And Estimated Terminal Voltage Of The DC Motor

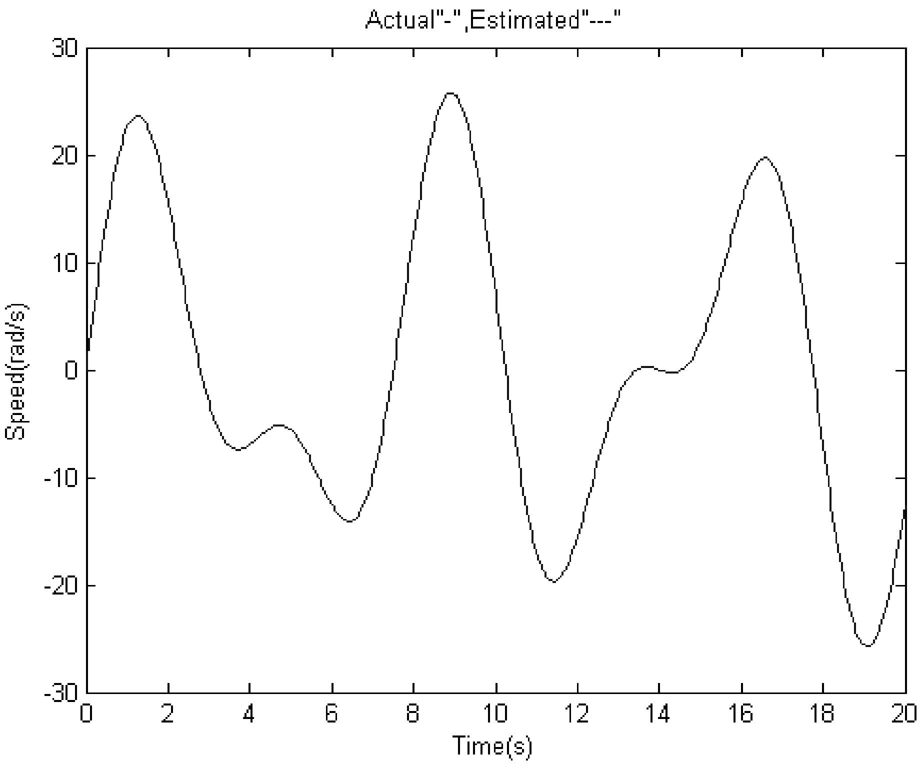

Figure 22: Tracking Performance For A Sinusoidal Reference Track

8.4 OVERALL CONTROLLER STRUCTURE :

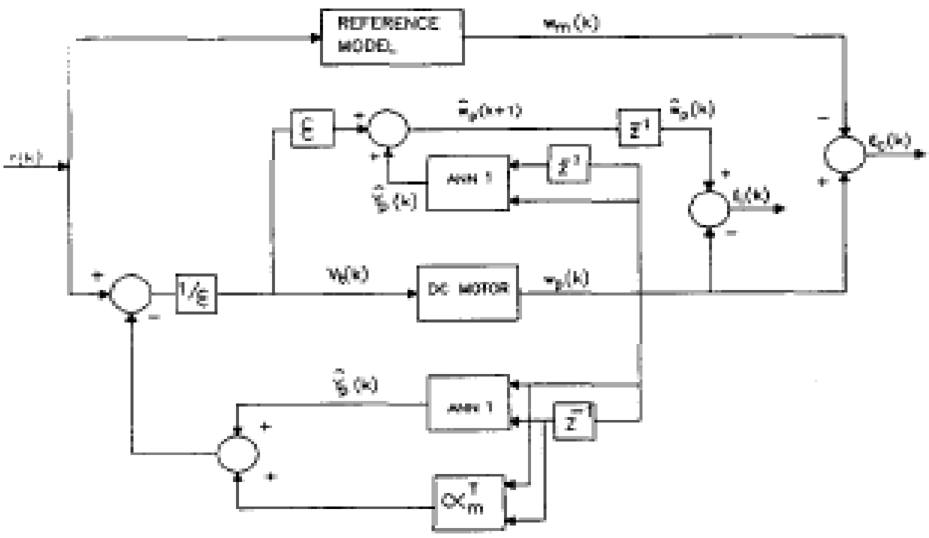

Figure 23: The Overall Structure Of The Controller In Topology

The maximum tracking error is ***0.55*** rad/s.

Both the ANN is the Block Diagram can be trained using the C-code mentioned in **Section 7.4.**

The overall structure of the identification and control system is displayed in Figure 20.

Chapter 9

RESULTS

9.1 TABLE OF OUTPUTS FOR COMPARISON:

KT	$W_p(k)$	$\widehat{W_p}(k+1)$	RMS ERROR	$W_m(k)$	$V_t(k)$	$\widehat{V_t}(k)$	$W_p(k)$
0	0	0	0	0	0	0	0
0.04	0.4478	0	0.4478	1.2022	5.5603	0	0
0.08	1.6429	1.5126	0.1303	2.4013	11.0919	7.1898	1.9486
0.12	3.0828	3.0041	0.0787	3.5939	16.5662	10.8739	2.988
0.16	4.5727	4.4738	0.0989	4.7769	21.9551	15.6482	4.3185
0.2	6.0509	5.919	0.1319	5.9471	27.231	20.7295	5.7224
0.24	7.4945	7.3335	0.161	7.1014	32.367	25.8374	7.1229
0.28	8.8921	8.7104	0.1817	8.2368	37.3371	30.8718	8.4934
0.32	10.2362	10.0428	0.1934	9.3503	42.1166	35.7857	9.8221
0.36	11.5206	11.324	0.1966	10.4389	46.6817	40.5486	11.1015
0.4	12.7401	12.5478	0.1923	11.4998	51.0103	45.1353	12.3261
0.44	13.8896	13.7082	0.1814	12.5303	55.0816	49.5232	13.4908
0.48	14.9648	14.7996	0.1652	13.5277	58.8768	53.6916	14.5913
0.52	15.9619	15.8168	0.1451	14.4895	62.3785	57.621	15.6236
0.56	16.8774	16.7554	0.122	15.4132	65.5715	61.2935	16.5841
0.6	17.7082	17.611	0.0972	16.2966	68.4425	64.6931	17.4695
0.64	18.4521	18.3802	0.0719	17.1374	70.9802	67.8052	18.2773
0.68	19.1069	19.06	0.0469	17.9338	73.1756	70.6172	19.0049
0.72	19.6712	19.6478	0.0234	18.6837	75.0219	73.1186	19.6505
0.76	20.144	20.1419	0.0021	19.3855	76.5142	75.3006	20.2126
0.8	20.5248	20.5409	-0.0161	20.0376	77.6504	77.1566	20.6902
0.84	20.8136	20.8443	-0.0307	20.6386	78.4303	78.6822	21.0826
0.88	21.0108	21.052	-0.0412	21.1872	78.856	79.8748	21.3897
0.92	21.1172	21.1646	-0.0474	21.6825	78.9319	80.734	21.6116
0.96	21.1342	21.1833	-0.0491	22.1234	78.6646	81.2614	21.7489
1	21.0634	21.1098	-0.0464	22.5093	78.0627	81.4605	21.8027
1.04	20.9073	20.9465	-0.0392	22.8396	77.1371	81.3367	21.7742
1.08	20.6684	20.6962	-0.0278	23.1139	75.9006	80.8977	21.6654
1.12	20.3501	20.3625	-0.0124	23.3321	74.3677	80.1531	21.4783
1.16	19.9556	19.9493	0.0063	23.4941	72.5548	79.1143	21.2158
1.2	19.4888	19.4609	0.0279	23.6001	70.4801	77.7939	20.8806
1.24	18.954	18.9023	0.0517	23.6504	68.1631	76.2061	20.4759
1.28	18.3559	18.2787	0.0772	23.6455	65.6248	74.3667	20.0054
1.32	17.6994	17.5958	0.1036	23.586	62.8873	72.2931	19.473
1.36	16.9896	16.8595	0.1301	23.4729	59.974	70.0038	18.883
1.4	16.2323	16.0761	0.1562	23.3071	56.9089	67.5177	18.2398
1.44	15.4331	15.2521	0.181	23.0897	53.7168	64.8554	17.5482
1.48	14.5982	14.3942	0.204	22.8221	50.4232	62.0378	16.8132
1.52	13.7338	13.5093	0.2245	22.5057	47.0539	59.0865	16.0401
1.56	12.8462	12.6042	0.242	22.1421	43.6346	56.0235	15.2342
1.6	11.9421	11.6859	0.2562	21.733	40.1915	52.8713	14.4011
1.64	11.028	10.7614	0.2666	21.2803	36.75	49.6521	13.5466
1.68	10.1105	9.8377	0.2728	20.786	33.3357	46.3886	12.6764
1.72	9.1966	8.9214	0.2752	20.252	29.9733	43.1025	11.7963
1.76	8.2928	8.0193	0.2735	19.6806	26.6868	39.8165	10.9123
1.8	7.4055	7.1377	0.2678	19.0741	23.4993	36.5515	10.0303

1.84	6.5413	6.283	0.2583	18.4347	20.433	33.3286	9.156
1.88	5.7063	5.461	0.2453	17.765	17.5087	30.1678	8.2952
1.92	4.9067	4.6773	0.2294	17.0674	14.7459	27.0884	7.4535
1.96	4.1481	3.937	0.2111	16.3445	12.1626	24.1089	6.6363
2	3.4359	3.2451	0.1908	15.5988	9.7753	21.2467	5.8489
2.04	2.775	2.6059	0.1691	14.8331	7.5984	18.5178	5.0961
2.08	2.1701	2.0234	0.1467	14.0499	5.6448	15.9369	4.3825
2.12	1.6252	1.5008	0.1244	13.2519	3.9255	13.5179	3.7124
2.16	1.1436	1.0411	0.1025	12.4418	2.4492	11.2729	3.0898
2.2	0.7287	0.6468	0.0819	11.6223	1.2229	9.2124	2.518
2.24	0.3824	0.3195	0.0629	10.796	0.2512	7.3459	2.0002
2.28	0.1068	0.0605	0.0463	9.9656	-0.4632	5.6808	1.5387
2.32	-0.0972	-0.1294	0.0322	9.1336	-0.9199	4.2232	1.1357
2.36	-0.229	-0.2503	0.0213	8.3027	-1.1205	2.9773	0.7925
2.4	-0.2889	-0.3024	0.0135	7.4753	-1.0689	1.9454	0.5102
2.44	-0.278	-0.287	0.009	6.6539	-0.7708	1.1295	0.2891
2.48	-0.1978	-0.2055	0.0077	5.8409	-0.2343	0.5291	0.1289
2.52	-0.0507	-0.0601	0.0094	5.0385	0.5309	0.1412	0.0289
2.56	0.1608	0.1465	0.0143	4.249	1.5129	-0.0387	-0.0122
2.6	0.4332	0.4111	0.0221	3.4745	2.6982	-0.0167	0.0038
2.64	0.7626	0.7302	0.0324	2.7171	4.0715	0.2008	0.0746
2.68	1.1443	1.0996	0.0447	1.9786	5.6159	0.6048	0.1976
2.72	1.5735	1.5147	0.0588	1.2608	7.3131	1.1848	0.3693
2.76	2.0448	1.9707	0.0741	0.5654	9.1434	1.9288	0.5862
2.8	2.5523	2.4624	0.0899	-0.106	11.0861	2.8237	0.8442
2.84	3.0902	2.984	0.1062	-0.752	13.1193	3.8549	1.1389
2.88	3.652	3.5298	0.1222	-1.3712	15.2203	5.0068	1.4657
2.92	4.2313	4.0937	0.1376	-1.9626	17.3657	6.2626	1.8197
2.96	4.8216	4.6693	0.1523	-2.5251	19.5318	7.6046	2.1959
3	5.4161	5.2504	0.1657	-3.0579	21.6942	9.0146	2.589
3.04	6.0083	5.8304	0.1779	-3.5601	23.8286	10.4735	2.9937
3.08	6.5911	6.4028	0.1883	-4.0312	25.9108	11.962	3.4047
3.12	7.1583	6.9611	0.1972	-4.4707	27.9166	13.4597	3.8164
3.16	7.7033	7.4988	0.2045	-4.8783	29.8224	14.947	4.2235
3.2	8.2199	8.0095	0.2104	-5.254	31.605	16.4038	4.6208
3.24	8.7018	8.4871	0.2147	-5.5976	33.2421	17.8104	5.0031
3.28	9.1434	8.9256	0.2178	-5.9092	34.7122	19.147	5.3652
3.32	9.5392	9.3191	0.2201	-6.1892	35.9949	20.3946	5.7023
3.36	9.8837	9.6623	0.2214	-6.4379	37.0713	21.535	6.0096
3.4	10.1723	9.95	0.2223	-6.6559	37.9235	22.5504	6.2827
3.44	10.4001	10.1774	0.2227	-6.8439	38.5353	23.4241	6.5174
3.48	10.5634	10.3403	0.2231	-7.0025	38.8923	24.1403	6.7096
3.52	10.6584	10.4347	0.2237	-7.1327	38.9816	24.6849	6.8557
3.56	10.6815	10.4572	0.2243	-7.2355	38.7923	25.0448	6.9523
3.6	10.6303	10.4049	0.2254	-7.3121	38.3156	25.208	6.9964
3.64	10.5021	10.2753	0.2268	-7.3636	37.5444	25.1647	6.9852
3.68	10.2951	10.0667	0.2284	-7.3914	36.474	24.9062	6.9165
3.72	10.0078	9.7778	0.23	-7.3968	35.1017	24.4256	6.7882
3.76	9.6394	9.4078	0.2316	-7.3815	33.427	23.7178	6.5987
3.8	9.1892	8.9566	0.2326	-7.3468	31.4516	22.7794	6.3467

3.84	8.6575	8.4248	0.2327	-7.2944	29.1795	21.6088	6.0314
3.88	8.0445	7.8133	0.2312	-7.226	26.6167	20.2061	5.6523
3.92	7.3515	7.124	0.2275	-7.1433	23.7715	18.5735	5.2094
3.96	6.58	6.3589	0.2211	-7.0481	20.6541	16.7146	4.7029
4	5.7319	5.521	0.2109	-6.9421	17.277	14.635	4.1337
4.04	4.8097	4.6135	0.1962	-6.8272	13.6544	12.342	3.5028
4.08	3.8166	3.6403	0.1763	-6.7052	9.8028	9.8442	2.8118
4.12	2.756	2.6059	0.1501	-6.5778	5.7399	7.1525	2.0626
4.16	1.6319	1.515	0.1169	-6.447	1.4856	4.2786	1.2577
4.2	0.4488	0.373	0.0758	-6.3144	-2.9389	1.2357	0.3997
4.24	-0.7879	-0.8145	0.0266	-6.1819	-7.5111	-1.9615	-0.5082
4.28	-2.0663	-2.0412	-0.0251	-6.0512	-12.2071	-5.3086	-1.4622
4.32	-3.3742	-3.301	-0.0732	-5.9241	-17.0019	-8.8219	-2.4543
4.36	-4.7023	-4.5872	-0.1151	-5.8021	-21.8697	-12.4678	-3.4751
4.4	-6.0428	-5.893	-0.1498	-5.6869	-26.7837	-16.2198	-4.517
4.44	-7.3881	-7.2112	-0.1769	-5.5799	-31.7168	-20.0556	-5.5736
4.48	-8.73	-8.5344	-0.1956	-5.4827	-36.6413	-23.9547	-6.6389
4.52	-10.0622	-9.8555	-0.2067	-5.3967	-41.5293	-27.894	-7.7067
4.56	-11.376	-11.1667	-0.2093	-5.323	-46.353	-31.8537	-8.7718
4.6	-12.6646	-12.4607	-0.2039	-5.263	-51.0847	-35.8092	-9.8279
4.64	-13.9215	-13.73	-0.1915	-5.2177	-55.697	-39.7376	-10.8693
4.68	-15.1394	-14.9672	-0.1722	-5.1881	-60.163	-43.6175	-11.8907
4.72	-16.3117	-16.1652	-0.1465	-5.1751	-64.4566	-47.4253	-12.8866
4.76	-17.4326	-17.3169	-0.1157	-5.1795	-68.5526	-51.1379	-13.8517
4.8	-18.4963	-18.4156	-0.0807	-5.2019	-72.4267	-54.7335	-14.7811
4.84	-19.4972	-19.4547	-0.0425	-5.2429	-76.0561	-58.1911	-15.6701
4.88	-20.4302	-20.428	-0.0022	-5.3028	-79.4191	-61.4899	-16.5143
4.92	-21.2907	-21.3298	0.0391	-5.3821	-82.4956	-64.61	-17.3094
4.96	-22.0744	-22.1547	0.0803	-5.4807	-85.2673	-67.5326	-18.0514
5	-22.7773	-22.8976	0.1203	-5.5988	-87.7175	-70.2399	-18.7368
5.04	-23.396	-23.5542	0.1582	-5.7363	-89.8316	-72.7158	-19.3621
5.08	-23.9276	-24.1205	0.1929	-5.8929	-91.5967	-74.9454	-19.9245
5.12	-24.3696	-24.5931	0.2235	-6.0683	-93.0023	-76.9154	-20.4211
5.16	-24.7197	-24.969	0.2493	-6.262	-94.0399	-78.6142	-20.8497
5.2	-24.9765	-25.2461	0.2696	-6.4733	-94.7031	-80.0318	-21.2082
5.24	-25.1388	-25.4225	0.2837	-6.7016	-94.9881	-81.1602	-21.4951
5.28	-25.2058	-25.4974	0.2916	-6.9459	-94.8931	-81.993	-21.7089
5.32	-25.1774	-25.4701	0.2927	-7.2054	-94.4186	-82.5257	-21.8487
5.36	-25.0537	-25.3409	0.2872	-7.4788	-93.5675	-82.756	-21.914
5.4	-24.8356	-25.1105	0.2749	-7.765	-92.345	-82.683	-21.9044
5.44	-24.5241	-24.7803	0.2562	-8.0628	-90.7585	-82.3081	-21.8201
5.48	-24.1207	-24.3523	0.2316	-8.3706	-88.8174	-81.6344	-21.6615
5.52	-23.6276	-23.8291	0.2015	-8.687	-86.5336	-80.6666	-21.4293
5.56	-23.047	-23.2138	0.1668	-9.0103	-83.9207	-79.4113	-21.1247
5.6	-22.3821	-22.5102	0.1281	-9.3391	-80.9943	-77.8768	-20.7492
5.64	-21.6363	-21.7224	0.0861	-9.6713	-77.7722	-76.0734	-20.3045
5.68	-20.8131	-20.8553	0.0422	-10.0054	-74.2734	-74.0131	-19.793
5.72	-19.9167	-19.9139	-0.0028	-10.3394	-70.5188	-71.7085	-19.217
5.76	-18.9517	-18.904	-0.0477	-10.6714	-66.5308	-69.1739	-18.5793
5.8	-17.923	-17.8315	-0.0915	-10.9994	-62.3329	-66.4253	-17.883

5.84	-16.8358	-16.7028	-0.133	-11.3214	-57.9499	-63.4792	-17.1317
5.88	-15.6958	-15.5246	-0.1712	-11.6354	-53.4074	-60.3535	-16.3288
5.92	-14.5088	-14.3039	-0.2049	-11.9395	-48.7321	-57.0667	-15.4786
5.96	-13.2809	-13.0477	-0.2332	-12.2314	-43.9509	-53.638	-14.5851
6	-12.0187	-11.7635	-0.2552	-12.5093	-39.0916	-50.0874	-13.6529
6.04	-10.7289	-10.4585	-0.2704	-12.7711	-34.1818	-46.435	-12.6866
6.08	-9.4183	-9.1404	-0.2779	-13.0147	-29.2496	-42.7014	-11.6913
6.12	-8.0939	-7.8165	-0.2774	-13.2382	-24.3226	-38.9075	-10.6719
6.16	-6.7628	-6.4944	-0.2684	-13.4397	-19.4284	-35.0734	-9.6338
6.2	-5.4326	-5.1813	-0.2513	-13.6172	-14.5938	-31.2195	-8.5822
6.24	-4.1103	-3.8846	-0.2257	-13.769	-9.8452	-27.3667	-7.5229
6.28	-2.8031	-2.6113	-0.1918	-13.8932	-5.208	-23.5341	-6.4611
6.32	-1.5184	-1.3682	-0.1502	-13.9882	-0.7067	-19.7409	-5.4024
6.36	-0.2634	-0.1618	-0.1016	-14.0524	3.6354	-16.0057	-4.3525
6.4	0.954	1.0016	-0.0476	-14.0842	7.7966	-12.3466	-3.3169
6.44	2.1206	2.1162	0.0044	-14.0823	11.7562	-8.764	-2.3016
6.48	3.2265	3.1766	0.0499	-14.0453	15.4954	-5.2548	-1.3159
6.52	4.266	4.178	0.088	-13.9719	18.9971	-1.8575	-0.3683
6.56	5.2349	5.1156	0.1193	-13.8612	22.2461	1.4047	0.5359
6.6	6.1298	5.9855	0.1443	-13.7122	25.2288	4.5152	1.3931
6.64	6.9479	6.7839	0.164	-13.5239	27.934	7.4603	2.2004
6.68	7.6872	7.508	0.1792	-13.2958	30.3524	10.2281	2.9557
6.72	8.3457	8.1551	0.1906	-13.0273	32.4767	12.8086	3.6569
6.76	8.9223	8.7234	0.1989	-12.7179	34.3021	15.1929	4.3024
6.8	9.4166	9.2114	0.2052	-12.3675	35.8256	17.3742	4.8913
6.84	9.8284	9.6185	0.2099	-11.9758	37.0468	19.3476	5.4227
6.88	10.1578	9.9445	0.2133	-11.5429	37.9671	21.1099	5.8964
6.92	10.406	10.1898	0.2162	-11.0691	38.5904	22.659	6.3122
6.96	10.5742	10.3553	0.2189	-10.5546	38.9224	23.9951	6.6706
7	10.6642	10.4428	0.2214	-10	38.9711	25.1204	6.9723
7.04	10.6784	10.4542	0.2242	-9.4059	38.7465	26.0376	7.2184
7.08	10.6194	10.3924	0.227	-8.7732	38.2604	26.7523	7.4102
7.12	10.4905	10.2604	0.2301	-8.1028	37.5265	27.2708	7.5496
7.16	10.2952	10.062	0.2332	-7.3959	36.5601	27.6011	7.6386
7.2	10.0375	9.8013	0.2362	-6.6537	35.3782	27.7527	7.6796
7.24	9.7218	9.4829	0.2389	-5.8776	33.9993	27.7362	7.6753
7.28	9.3527	9.1116	0.2411	-5.0692	32.4431	27.5635	7.6286
7.32	8.9355	8.6929	0.2426	-4.2303	30.7305	27.2475	7.5428
7.36	8.4754	8.2324	0.243	-3.3626	28.8835	26.8021	7.4213
7.4	7.9781	7.736	0.2421	-2.4682	26.9248	26.2419	7.2678
7.44	7.4494	7.2097	0.2397	-1.549	24.8778	25.5825	7.0863
7.48	6.8955	6.6599	0.2356	-0.6074	22.7664	24.8397	6.8809
7.52	6.3227	6.093	0.2297	0.3543	20.615	24.0299	6.6558
7.56	5.7374	5.5155	0.2219	1.3338	18.4477	23.1698	6.4155
7.6	5.1461	4.934	0.2121	2.3285	16.2889	22.2764	6.1644
7.64	4.5554	4.3549	0.2005	3.3359	14.1628	21.3663	5.9071
7.68	3.9718	3.7847	0.1871	4.3531	12.0928	20.4566	5.6484
7.72	3.4019	3.2298	0.1721	5.3776	10.1021	19.5635	5.3927
7.76	2.8521	2.6963	0.1558	6.4065	8.2129	18.7034	5.1447
7.8	2.3288	2.1902	0.1386	7.4368	6.4466	17.892	4.9091

7.84	1.8379	1.7172	0.1207	8.4659	4.8234	17.1443	4.6901
7.88	1.3856	1.2828	0.1028	9.4906	3.3624	16.4749	4.4922
7.92	0.9771	0.8919	0.0852	10.508	2.0811	15.8975	4.3194
7.96	0.6178	0.5494	0.0684	11.5152	0.9956	15.4247	4.1757
8	0.3124	0.2596	0.0528	12.5093	0.1204	15.0683	4.0646
8.04	0.0651	0.0262	0.0389	13.4872	-0.5318	14.839	3.9896
8.08	-0.12	-0.1473	0.0273	14.4461	-0.9502	14.7463	3.9535
8.12	-0.2398	-0.2581	0.0183	15.3831	-1.1259	14.7983	3.9589
8.16	-0.2915	-0.3037	0.0122	16.2952	-1.052	15.002	4.0082
8.2	-0.2733	-0.2823	0.009	17.1797	-0.7237	15.3639	4.1031
8.24	-0.1838	-0.1927	0.0089	18.0339	-0.1382	15.8884	4.2447
8.28	-0.0221	-0.0341	0.012	18.855	0.705	16.5779	4.4338
8.32	0.212	0.1936	0.0184	19.6405	1.8045	17.433	4.6707
8.36	0.5179	0.4899	0.028	20.3879	3.1565	18.4537	4.9554
8.4	0.8943	0.854	0.0403	21.0948	4.7553	19.6392	5.2871
8.44	1.3393	1.2841	0.0552	21.7588	6.593	20.986	5.6645
8.48	1.8502	1.7783	0.0719	22.3779	8.6597	22.4894	6.0861
8.52	2.4241	2.3338	0.0903	22.9499	10.9435	24.143	6.5494
8.56	3.057	2.9475	0.1095	23.473	13.4308	25.9392	7.052
8.6	3.7447	3.6157	0.129	23.9454	16.106	27.8687	7.5906
8.64	4.4825	4.3343	0.1482	24.3654	18.9519	29.9206	8.1618
8.68	5.265	5.0985	0.1665	24.7315	21.9497	32.0829	8.7616
8.72	6.0866	5.9034	0.1832	25.0425	25.0793	34.3425	9.386
8.76	6.9415	6.7436	0.1979	25.2971	28.3191	36.6845	10.0303
8.8	7.8233	7.6133	0.21	25.4945	31.6465	39.0932	10.6899
8.84	8.7254	8.5064	0.219	25.6337	35.0379	41.5519	11.3598
8.88	9.6411	9.4167	0.2244	25.7141	38.4689	44.0427	12.0349
8.92	10.5639	10.3375	0.2264	25.7352	41.9144	46.5466	12.7098
8.96	11.4867	11.2621	0.2246	25.6969	45.3489	49.0453	13.3795
9	12.4027	12.1837	0.219	25.5988	48.7464	51.5186	14.0385
9.04	13.3052	13.0953	0.2099	25.4413	52.0811	53.9462	14.6815
9.08	14.187	13.99	0.197	25.2244	55.3271	56.3086	15.3035
9.12	15.0419	14.8608	0.1811	24.9486	58.4589	58.5842	15.8989
9.16	15.8635	15.7009	0.1626	24.6145	61.4512	60.7535	16.463
9.2	16.6454	16.5035	0.1419	24.223	64.2795	62.7969	16.991
9.24	17.3815	17.262	0.1195	23.775	66.9203	64.6946	17.4782
9.28	18.0661	17.9701	0.096	23.2716	69.3506	66.4273	17.92
9.32	18.6939	18.6216	0.0723	22.7142	71.549	67.9768	18.3122
9.36	19.2597	19.2108	0.0489	22.1042	73.4951	69.3259	18.6509
9.4	19.7588	19.7322	0.0266	21.4433	75.1702	70.4585	18.9325
9.44	20.1868	20.1807	0.0061	20.7333	76.5569	71.3596	19.1536
9.48	20.5398	20.5517	-0.0119	19.9761	77.6396	72.0155	19.3113
9.52	20.8142	20.841	-0.0268	19.1737	78.4048	72.4137	19.4027
9.56	21.0067	21.045	-0.0383	18.3284	78.8406	72.5436	19.4255
9.6	21.1147	21.1605	-0.0458	17.4426	78.9373	72.3959	19.3777
9.64	21.1358	21.1848	-0.049	16.5186	78.6872	71.9631	19.2576
9.68	21.0683	21.1159	-0.0476	15.5592	78.0848	71.2396	19.0639
9.72	20.9108	20.9524	-0.0416	14.5668	77.1269	70.2219	18.7957
9.76	20.6625	20.6934	-0.0309	13.5444	75.8125	68.9082	18.4526
9.8	20.3226	20.3386	-0.016	12.4947	74.1427	67.2986	18.0344

9.84	19.8913	19.8883	0.003	11.4208	72.1212	65.3945	17.5411
9.88	19.3693	19.3435	0.0258	10.3254	69.7536	63.1998	16.9734
9.92	18.7572	18.7058	0.0514	9.2118	67.048	60.7208	16.3322
9.96	18.0567	17.9774	0.0793	8.083	64.0145	57.9648	15.619
10	17.2695	17.1609	0.1086	6.9421	60.6653	54.9413	14.8354
10.04	16.3982	16.2597	0.1385	5.7923	57.0149	51.6615	13.9835
10.08	15.4454	15.2777	0.1677	4.6367	53.0793	48.1384	13.0657
10.12	14.4145	14.2193	0.1952	3.4784	48.8769	44.3864	12.0848
10.16	13.3092	13.0895	0.2197	2.3206	44.4273	40.4217	11.044
10.2	12.1337	11.8935	0.2402	1.1664	39.7522	36.2614	9.9468
10.24	10.8925	10.6372	0.2553	0.019	34.8744	31.9245	8.7971
10.28	9.5908	9.3268	0.264	-1.1187	29.8181	27.4303	7.5989
10.32	8.2338	7.9688	0.265	-2.2437	24.6089	22.8	6.3567
10.36	6.8275	6.57	0.2575	-3.3529	19.2731	18.0552	5.0755
10.4	5.378	5.1376	0.2404	-4.4436	13.838	13.2182	3.7601
10.44	3.8919	3.6789	0.213	-5.513	8.3315	8.3119	2.4161
10.48	2.3758	2.2014	0.1744	-6.5583	2.782	3.3598	1.049
10.52	0.837	0.7128	0.1242	-7.5769	-2.782	-1.6151	-0.3355
10.56	-0.7166	-0.7794	0.0628	-8.5664	-8.3315	-6.5889	-1.7313
10.6	-2.2695	-2.2673	-0.0022	-9.5244	-13.838	-11.5475	-3.1318
10.64	-3.8039	-3.7434	-0.0605	-10.4486	-19.2731	-16.5222	-4.5257
10.68	-5.3086	-5.2003	-0.1083	-11.3369	-24.6089	-21.4654	-5.8991
10.72	-6.7761	-6.6307	-0.1454	-12.1873	-29.8181	-26.3347	-7.2414
10.76	-8.1991	-8.0271	-0.172	-12.998	-34.8744	-31.102	-8.5456
10.8	-9.5711	-9.3826	-0.1885	-13.7672	-39.7522	-35.7422	-9.8056
10.84	-10.8862	-10.6902	-0.196	-14.4933	-44.4273	-40.2315	-11.0159
10.88	-12.1388	-11.9435	-0.1953	-15.1751	-48.8769	-44.5472	-12.1716
10.92	-13.3238	-13.1363	-0.1875	-15.8111	-53.0793	-48.6683	-13.268
10.96	-14.4367	-14.2629	-0.1738	-16.4005	-57.0149	-52.5747	-14.3008
11	-15.4734	-15.3178	-0.1556	-16.9421	-60.6653	-56.2479	-15.2664
11.04	-16.43	-16.2962	-0.1338	-17.4354	-64.0145	-59.6709	-16.1614
11.08	-17.3035	-17.1938	-0.1097	-17.8797	-67.048	-62.828	-16.9827
11.12	-18.0912	-18.0066	-0.0846	-18.2745	-69.7536	-65.7055	-17.7278
11.16	-18.7908	-18.7314	-0.0594	-18.6197	-72.1212	-68.2913	-18.3945
11.2	-19.4005	-19.3655	-0.035	-18.9151	-74.1427	-70.5753	-18.9811
11.24	-19.9191	-19.9066	-0.0125	-19.1609	-75.8125	-72.5492	-19.4863
11.28	-20.346	-20.3533	0.0073	-19.3573	-77.1269	-74.2069	-19.9093
11.32	-20.6807	-20.7046	0.0239	-19.5047	-78.0848	-75.5442	-20.2495
11.36	-20.9236	-20.9601	0.0365	-19.6036	-78.6872	-76.559	-20.507
11.4	-21.0752	-21.1201	0.0449	-19.6549	-78.9373	-77.2513	-20.6821
11.44	-21.1367	-21.1856	0.0489	-19.6593	-78.8406	-77.623	-20.7755
11.48	-21.1096	-21.1579	0.0483	-19.6179	-78.4048	-77.6779	-20.7884
11.52	-20.9959	-21.0392	0.0433	-19.5318	-77.6396	-77.4218	-20.7224
11.56	-20.798	-20.832	0.034	-19.4024	-76.5569	-76.8623	-20.5792
11.6	-20.519	-20.5396	0.0206	-19.231	-75.1702	-76.0089	-20.3612
11.64	-20.1621	-20.1656	0.0035	-19.0192	-73.4951	-74.8733	-20.0711
11.68	-19.731	-19.7142	-0.0168	-18.7687	-71.549	-73.4685	-19.712
11.72	-19.2296	-19.19	-0.0396	-18.4812	-69.3506	-71.8091	-19.2871
11.76	-18.6625	-18.5983	-0.0642	-18.1585	-66.9203	-69.9105	-18.8
11.8	-18.0346	-17.9443	-0.0903	-17.8026	-64.2795	-67.7902	-18.2548

11.84	-17.3508	-17.2339	-0.1169	-17.4156	-61.4512	-65.4667	-17.6556
11.88	-16.6165	-16.4733	-0.1432	-16.9995	-58.4589	-62.9595	-17.007
11.92	-15.8376	-15.6688	-0.1688	-16.5565	-55.3271	-60.2885	-16.3138
11.96	-15.0198	-14.827	-0.1928	-16.0889	-52.0811	-57.4752	-15.5809
12	-14.1693	-13.9547	-0.2146	-15.5988	-48.7464	-54.5409	-14.8136
12.04	-13.2925	-13.0588	-0.2337	-15.0887	-45.3489	-51.5077	-14.0171
12.08	-12.3958	-12.1462	-0.2496	-14.5608	-41.9144	-48.3979	-13.1971
12.12	-11.4858	-11.224	-0.2618	-14.0174	-38.4689	-45.2339	-12.3592
12.16	-10.5693	-10.299	-0.2703	-13.4609	-35.0379	-42.0381	-11.509
12.2	-9.6528	-9.3782	-0.2746	-12.8937	-31.6465	-38.8326	-10.6524
12.24	-8.743	-8.4682	-0.2748	-12.3181	-28.3191	-35.6392	-9.7952
12.28	-7.8466	-7.5755	-0.2711	-11.7364	-25.0793	-32.4793	-8.9432
12.32	-6.9701	-6.7066	-0.2635	-11.1509	-21.9497	-29.3736	-8.102
12.36	-6.1197	-5.8675	-0.2522	-10.5638	-18.9519	-26.3419	-7.2774
12.4	-5.3017	-5.064	-0.2377	-9.9773	-16.106	-23.4035	-6.4747
12.44	-4.5219	-4.3014	-0.2205	-9.3936	-13.4308	-20.5764	-5.6995
12.48	-3.7859	-3.5848	-0.2011	-8.8148	-10.9435	-17.8778	-4.9566
12.52	-3.0988	-2.9187	-0.1801	-8.2427	-8.6597	-15.3236	-4.2511
12.56	-2.4653	-2.3073	-0.158	-7.6794	-6.593	-12.9281	-3.5873
12.6	-1.8899	-1.7544	-0.1355	-7.1267	-4.7553	-10.7046	-2.9695
12.64	-1.3763	-1.2629	-0.1134	-6.5863	-3.1565	-8.665	-2.4014
12.68	-0.9277	-0.8357	-0.092	-6.0599	-1.8045	-6.8195	-1.8865
12.72	-0.5468	-0.4746	-0.0722	-5.5489	-0.705	-5.1769	-1.4276
12.76	-0.2357	-0.1814	-0.0543	-5.0548	0.1382	-3.7443	-1.0271
12.8	0.0042	0.0431	-0.0389	-4.579	0.7237	-2.5272	-0.6871
12.84	0.1721	0.1985	-0.0264	-4.1225	1.052	-1.5293	-0.4087
12.88	0.2679	0.2849	-0.017	-3.6864	1.1259	-0.7525	-0.193
12.92	0.2922	0.3031	-0.0109	-3.2716	0.9502	-0.1976	-0.0402
12.96	0.2464	0.2544	-0.008	-2.8791	0.5318	0.1361	0.0501
13	0.1325	0.1406	-0.0081	-2.5093	-0.1204	0.2518	0.0788
13.04	-0.0472	-0.0357	-0.0115	-2.1629	-0.9956	0.1549	0.0473
13.08	-0.2897	-0.2718	-0.0179	-1.8402	-2.0811	-0.1477	-0.0424
13.12	-0.5911	-0.5641	-0.027	-1.5415	-3.3624	-0.6485	-0.188
13.16	-0.9472	-0.9089	-0.0383	-1.2669	-4.8234	-1.3383	-0.3866
13.2	-1.3533	-1.3017	-0.0516	-1.0165	-6.4466	-2.2057	-0.6345
13.24	-1.8042	-1.738	-0.0662	-0.79	-8.2129	-3.238	-0.9281
13.28	-2.2944	-2.2124	-0.082	-0.5871	-10.1021	-4.4212	-1.2631
13.32	-2.8178	-2.7198	-0.098	-0.4076	-12.0928	-5.7401	-1.6348
13.36	-3.3685	-3.2542	-0.1143	-0.2509	-14.1628	-7.178	-2.0384
13.4	-3.9399	-3.8098	-0.1301	-0.1162	-16.2889	-8.7174	-2.4688
13.44	-4.5255	-4.3804	-0.1451	-0.0029	-18.4477	-10.3398	-2.9207
13.48	-5.1188	-4.9596	-0.1592	0.0899	-20.615	-12.0259	-3.3884
13.52	-5.7129	-5.541	-0.1719	0.1632	-22.7664	-13.7559	-3.8665
13.56	-6.3013	-6.118	-0.1833	0.2181	-24.8778	-15.5092	-4.3493
13.6	-6.8771	-6.6841	-0.193	0.2559	-26.9248	-17.2651	-4.831
13.64	-7.434	-7.2329	-0.2011	0.2776	-28.8835	-19.0023	-5.306
13.68	-7.9655	-7.7579	-0.2076	0.2848	-30.7305	-20.7003	-5.7688
13.72	-8.4655	-8.2528	-0.2127	0.2788	-32.4431	-22.338	-6.2137
13.76	-8.928	-8.7116	-0.2164	0.2611	-33.9993	-23.8949	-6.6356
13.8	-9.3474	-9.1283	-0.2191	0.2333	-35.3782	-25.3509	-7.0291

13.84	-9.7181	-9.4973	-0.2208	0.1969	-36.5601	-26.6868	-7.3893
13.88	-10.0353	-9.8134	-0.2219	0.1538	-37.5265	-27.8839	-7.7114
13.92	-10.294	-10.0715	-0.2225	0.1054	-38.2604	-28.9247	-7.9911
13.96	-10.4901	-10.2672	-0.2229	0.0536	-38.7465	-29.7922	-8.2239
14	-10.6197	-10.3963	-0.2234	0	-38.9711	-30.4714	-8.4061
14.04	-10.6791	-10.4552	-0.2239	-0.0536	-38.9224	-30.9486	-8.5341
14.08	-10.6654	-10.4406	-0.2248	-0.1054	-38.5904	-31.2108	-8.6046
14.12	-10.5759	-10.3499	-0.226	-0.1538	-37.9671	-31.2475	-8.6149
14.16	-10.4086	-10.181	-0.2276	-0.1969	-37.0468	-31.0491	-8.5622
14.2	-10.1616	-9.9323	-0.2293	-0.2333	-35.8256	-30.6084	-8.4447
14.24	-9.8338	-9.6029	-0.2309	-0.2611	-34.3021	-29.9195	-8.2604
14.28	-9.4245	-9.1923	-0.2322	-0.2788	-32.4767	-28.9784	-8.0081
14.32	-8.9335	-8.7007	-0.2328	-0.2848	-30.3524	-27.7831	-7.6868
14.36	-8.3611	-8.1289	-0.2322	-0.2776	-27.934	-26.3336	-7.296
14.4	-7.708	-7.4783	-0.2297	-0.2559	-25.2288	-24.6316	-6.8357
14.44	-6.9754	-6.7508	-0.2246	-0.2181	-22.2461	-22.6807	-6.3061
14.48	-6.1654	-5.9489	-0.2165	-0.1632	-18.9971	-20.4862	-5.708
14.52	-5.2798	-5.0757	-0.2041	-0.0899	-15.4954	-18.0557	-5.0426
14.56	-4.3218	-4.1349	-0.1869	0.0029	-11.7562	-15.3979	-4.3115
14.6	-3.2945	-3.1305	-0.164	0.1162	-7.7966	-12.5237	-3.5166
14.64	-2.2017	-2.0672	-0.1345	0.2509	-3.6354	-9.4454	-2.6605
14.68	-1.0475	-0.95	-0.0975	0.4076	0.7067	-6.1768	-1.746
14.72	0.1634	0.2154	-0.052	0.5871	5.208	-2.733	-0.7764
14.76	1.4227	1.4233	-0.0006	0.79	9.8452	0.8701	0.2447
14.8	2.7171	2.6674	0.0497	1.0165	14.5938	4.6479	1.3114
14.84	4.0362	3.9412	0.095	1.2669	19.4284	8.5859	2.4139
14.88	5.3715	5.2381	0.1334	1.5415	24.3226	12.6501	3.5432
14.92	6.7154	6.551	0.1644	1.8402	29.2496	16.8156	4.692
14.96	8.0601	7.8726	0.1875	2.1629	34.1818	21.0602	5.854
15	9.3983	9.1957	0.2026	2.5093	39.0916	25.3603	7.0227
15.04	10.7218	10.5127	0.2091	2.8791	43.9509	29.6934	8.1919
15.08	12.0238	11.8163	0.2075	3.2716	48.7321	34.034	9.3551
15.12	13.2975	13.0989	0.1986	3.6864	53.4074	38.3584	10.5062
15.16	14.5357	14.353	0.1827	4.1225	57.9499	42.6433	11.6395
15.2	15.7316	15.5715	0.1601	4.579	62.3329	46.8641	12.7491
15.24	16.879	16.7473	0.1317	5.0548	66.5308	50.9958	13.8291
15.28	17.9719	17.8733	0.0986	5.5489	70.5188	55.0146	14.8739
15.32	19.0049	18.943	0.0619	6.0599	74.2734	58.8977	15.8785
15.36	19.9725	19.9499	0.0226	6.5863	77.7722	62.6228	16.8379
15.4	20.8698	20.8882	-0.0184	7.1267	80.9943	66.1682	17.7474
15.44	21.6924	21.7522	-0.0598	7.6794	83.9207	69.513	18.6025
15.48	22.4362	22.5367	-0.1005	8.2427	86.5336	72.6378	19.3991
15.52	23.0974	23.237	-0.1396	8.8148	88.8174	75.5244	20.1334
15.56	23.6729	23.8489	-0.176	9.3936	90.7585	78.1561	20.8019
15.6	24.16	24.3687	-0.2087	9.9773	92.345	80.5177	21.4014
15.64	24.5562	24.7933	-0.2371	10.5638	93.5675	82.5958	21.9292
15.68	24.8599	25.12	-0.2601	11.1509	94.4186	84.3785	22.3829
15.72	25.0695	25.347	-0.2775	11.7364	94.8931	85.8559	22.7603
15.76	25.1842	25.4727	-0.2885	12.3181	94.9881	87.0201	23.0598
15.8	25.2035	25.4965	-0.293	12.8937	94.7031	87.8649	23.28

15.84	25.1274	25.4182	-0.2908	13.4609	94.0399	88.3863	23.42
15.88	24.9564	25.2383	-0.2819	14.0174	93.0023	88.5819	23.4793
15.92	24.6914	24.9577	-0.2663	14.5608	91.5967	88.4518	23.4575
15.96	24.3338	24.5784	-0.2446	15.0887	89.8316	87.9978	23.3549
16	23.8852	24.1024	-0.2172	15.5988	87.7175	87.2234	23.172
16.04	23.348	23.5327	-0.1847	16.0889	85.2673	86.1343	22.9098
16.08	22.7249	22.8728	-0.1479	16.5565	82.4956	84.7376	22.5693
16.12	22.0191	22.1265	-0.1074	16.9995	79.4191	83.0427	22.1523
16.16	21.2341	21.2985	-0.0644	17.4156	76.0561	81.061	21.661
16.2	20.3737	20.3935	-0.0198	17.8026	72.4267	78.8049	21.0976
16.24	19.4425	19.4172	0.0253	18.1585	68.5526	76.288	20.4648
16.28	18.445	18.3752	0.0698	18.4812	64.4566	73.5261	19.7658
16.32	17.3864	17.2738	0.1126	18.7687	60.163	70.5358	19.0038
16.36	16.2721	16.1195	0.1526	19.0192	55.697	67.3349	18.1826
16.4	15.1078	14.9191	0.1887	19.231	51.0847	63.9422	17.3063
16.44	13.8995	13.6798	0.2197	19.4024	46.353	60.3775	16.379
16.48	12.6537	12.4087	0.245	19.5318	41.5293	56.6609	15.4055
16.52	11.3768	11.1131	0.2637	19.6179	36.6413	52.8136	14.3905
16.56	10.0758	9.8007	0.2751	19.6593	31.7168	48.8569	13.339
16.6	8.7574	8.4787	0.2787	19.6549	26.7837	44.8125	12.2566
16.64	7.4287	7.1548	0.2739	19.6036	21.8697	40.702	11.1484
16.68	6.0971	5.8363	0.2608	19.5047	17.0019	36.5471	10.0202
16.72	4.77	4.5305	0.2395	19.3573	12.2071	32.3692	8.8778
16.76	3.4543	3.2446	0.2097	19.1609	7.5111	28.1902	7.727
16.8	2.1575	1.9855	0.172	18.9151	2.9389	24.03	6.5737
16.84	0.8868	0.76	0.1268	18.6197	-1.4856	19.9092	5.4238
16.88	-0.3507	-0.4256	0.0749	18.2745	-5.7399	15.8474	4.2832
16.92	-1.5444	-1.5653	0.0209	17.8797	-9.8028	11.8612	3.1579
16.96	-2.6815	-2.6535	-0.028	17.4354	-13.6544	7.936	2.0559
17	-3.7549	-3.685	-0.0699	16.9421	-17.277	4.1012	0.9872
17.04	-4.7595	-4.655	-0.1045	16.4005	-20.6541	0.3889	-0.0411
17.08	-5.6918	-5.5592	-0.1326	15.8111	-23.7715	-3.1802	-1.0243
17.12	-6.5486	-6.3938	-0.1548	15.1751	-26.6167	-6.5896	-1.9587
17.16	-7.3275	-7.1554	-0.1721	14.4933	-29.1795	-9.8252	-2.8414
17.2	-8.0266	-7.8413	-0.1853	13.7672	-31.4516	-12.8745	-3.6701
17.24	-8.6443	-8.4492	-0.1951	12.998	-33.427	-15.7267	-4.4425
17.28	-9.1798	-8.9775	-0.2023	12.1873	-35.1017	-18.3722	-5.1568
17.32	-9.6328	-9.4251	-0.2077	11.3369	-36.474	-20.8039	-5.8118
17.36	-10.0033	-9.7916	-0.2117	10.4486	-37.5444	-23.0165	-6.4066
17.4	-10.292	-10.0772	-0.2148	9.5244	-38.3156	-25.0058	-6.9407
17.44	-10.5	-10.2824	-0.2176	8.5664	-38.7923	-26.7694	-7.4138
17.48	-10.6288	-10.4087	-0.2201	7.5769	-38.9816	-28.3076	-7.8262
17.52	-10.6806	-10.4579	-0.2227	6.5583	-38.8923	-29.6214	-8.1785
17.56	-10.6579	-10.4323	-0.2256	5.513	-38.5353	-30.7138	-8.4715
17.6	-10.5635	-10.335	-0.2285	4.4436	-37.9235	-31.5898	-8.7066
17.64	-10.4009	-10.1693	-0.2316	3.3529	-37.0713	-32.2553	-8.8855
17.68	-10.1739	-9.9392	-0.2347	2.2437	-35.9949	-32.7184	-9.01
17.72	-9.8866	-9.649	-0.2376	1.1187	-34.7122	-32.9881	-9.0825
17.76	-9.5436	-9.3035	-0.2401	-0.019	-33.2421	-33.0749	-9.1057
17.8	-9.1498	-8.9079	-0.2419	-1.1664	-31.605	-32.9904	-9.0823

17.84	-8.7105	-8.4675	-0.243	-2.3206	-29.8224	-32.7476	-9.0156
17.88	-8.231	-7.9883	-0.2427	-3.4784	-27.9166	-32.36	-8.909
17.92	-7.7173	-7.4762	-0.2411	-4.6367	-25.9108	-31.8424	-8.7662
17.96	-7.1752	-6.9373	-0.2379	-5.7923	-23.8286	-31.21	-8.5912
18	-6.6111	-6.3782	-0.2329	-6.9421	-21.6942	-30.4787	-8.388
18.04	-6.0312	-5.8052	-0.226	-8.083	-19.5318	-29.665	-8.161
18.08	-5.4421	-5.2248	-0.2173	-9.2118	-17.3657	-28.7854	-7.9145
18.12	-4.8502	-4.6437	-0.2065	-10.3254	-15.2203	-27.8571	-7.6532
18.16	-4.2623	-4.0683	-0.194	-11.4208	-13.1193	-26.8968	-7.3817
18.2	-3.6847	-3.5049	-0.1798	-12.4947	-11.0861	-25.9213	-7.1046
18.24	-3.1241	-2.96	-0.1641	-13.5444	-9.1434	-24.9474	-6.8267
18.28	-2.5867	-2.4394	-0.1473	-14.5668	-7.3131	-23.9914	-6.5526
18.32	-2.0789	-1.9492	-0.1297	-15.5592	-5.6159	-23.0693	-6.287
18.36	-1.6066	-1.4948	-0.1118	-16.5186	-4.0715	-22.1962	-6.0345
18.4	-1.1755	-1.0816	-0.0939	-17.4426	-2.6982	-21.3871	-5.7993
18.44	-0.791	-0.7144	-0.0766	-18.3284	-1.5129	-20.6559	-5.5858
18.48	-0.4581	-0.3977	-0.0604	-19.1737	-0.5309	-20.0155	-5.3978
18.52	-0.1812	-0.1356	-0.0456	-19.9761	0.2343	-19.4782	-5.2392
18.56	0.0354	0.0682	-0.0328	-20.7333	0.7708	-19.0548	-5.1133
18.6	0.1883	0.2107	-0.0224	-21.4433	1.0689	-18.7556	-5.0231
18.64	0.2743	0.2891	-0.0148	-22.1042	1.1205	-18.5887	-4.9715
18.68	0.2913	0.3015	-0.0102	-22.7142	0.9199	-18.5623	-4.9608
18.72	0.2375	0.2461	-0.0086	-23.2716	0.4632	-18.6832	-4.9926
18.76	0.112	0.122	-0.01	-23.775	-0.2512	-18.956	-5.0684
18.8	-0.0859	-0.0711	-0.0148	-24.223	-1.2229	-19.3834	-5.189
18.84	-0.356	-0.3332	-0.0228	-24.6145	-2.4492	-19.967	-5.3549
18.88	-0.6974	-0.6636	-0.0338	-24.9486	-3.9255	-20.708	-5.5661
18.92	-1.1084	-1.0609	-0.0475	-25.2244	-5.6448	-21.6055	-5.8219
18.96	-1.5867	-1.5234	-0.0633	-25.4413	-7.5984	-22.6571	-6.1214
19	-2.1295	-2.0486	-0.0809	-25.5988	-9.7753	-23.8584	-6.4629
19.04	-2.7335	-2.6336	-0.0999	-25.6969	-12.1626	-25.2043	-6.8444
19.08	-3.3943	-3.2751	-0.1192	-25.7352	-14.7459	-26.6876	-7.2633
19.12	-4.1077	-3.969	-0.1387	-25.7141	-17.5087	-28.2996	-7.7168
19.16	-4.8685	-4.711	-0.1575	-25.6337	-20.433	-30.0305	-8.2016
19.2	-5.6713	-5.4962	-0.1751	-25.4945	-23.4993	-31.8693	-8.7141
19.24	-6.5103	-6.3195	-0.1908	-25.2971	-26.6868	-33.8032	-9.2502
19.28	-7.3794	-7.1751	-0.2043	-25.0425	-29.9733	-35.8186	-9.8057
19.32	-8.2722	-8.0573	-0.2149	-24.7315	-33.3357	-37.9005	-10.3763
19.36	-9.182	-8.9598	-0.2222	-24.3654	-36.75	-40.0333	-10.9572
19.4	-10.1021	-9.8762	-0.2259	-23.9454	-40.1915	-42.1996	-11.5435
19.44	-11.0256	-10.7997	-0.2259	-23.473	-43.6346	-44.382	-12.1304
19.48	-11.9459	-11.7237	-0.2222	-22.9499	-47.0539	-46.5625	-12.7128
19.52	-12.856	-12.6412	-0.2148	-22.3779	-50.4232	-48.7225	-13.286
19.56	-13.749	-13.5452	-0.2038	-21.7588	-53.7168	-50.8429	-13.8449
19.6	-14.6182	-14.4288	-0.1894	-21.0948	-56.9089	-52.9043	-14.3845
19.64	-15.4572	-15.2851	-0.1721	-20.3879	-59.974	-54.8869	-14.9
19.68	-16.2598	-16.1073	-0.1525	-19.6405	-62.8873	-56.7724	-15.3868
19.72	-17.0195	-16.8886	-0.1309	-18.855	-65.6248	-58.5419	-15.8404
19.76	-17.7306	-17.6227	-0.1079	-18.0339	-68.1631	-60.1765	-16.2564
19.8	-18.3874	-18.3033	-0.0841	-17.1797	-70.4801	-61.6581	-16.6306

19.84	-18.9848	-18.9243	-0.0605	-16.2952	-72.5548	-62.9699	-16.9592
19.88	-19.5178	-19.4803	-0.0375	-15.3831	-74.3677	-64.0956	-17.2384
19.92	-19.982	-19.9658	-0.0162	-14.4461	-75.9006	-65.0202	-17.4649
19.96	-20.373	-20.3761	0.0031	-13.4872	-77.1371	-65.7295	-17.6357
20	-20.687	-20.7068	0.0198	-12.5093	-78.0627		

9.2 PLOT OF TRAINED ANN FROM SYSTEM IDENTIFICATION CODING:

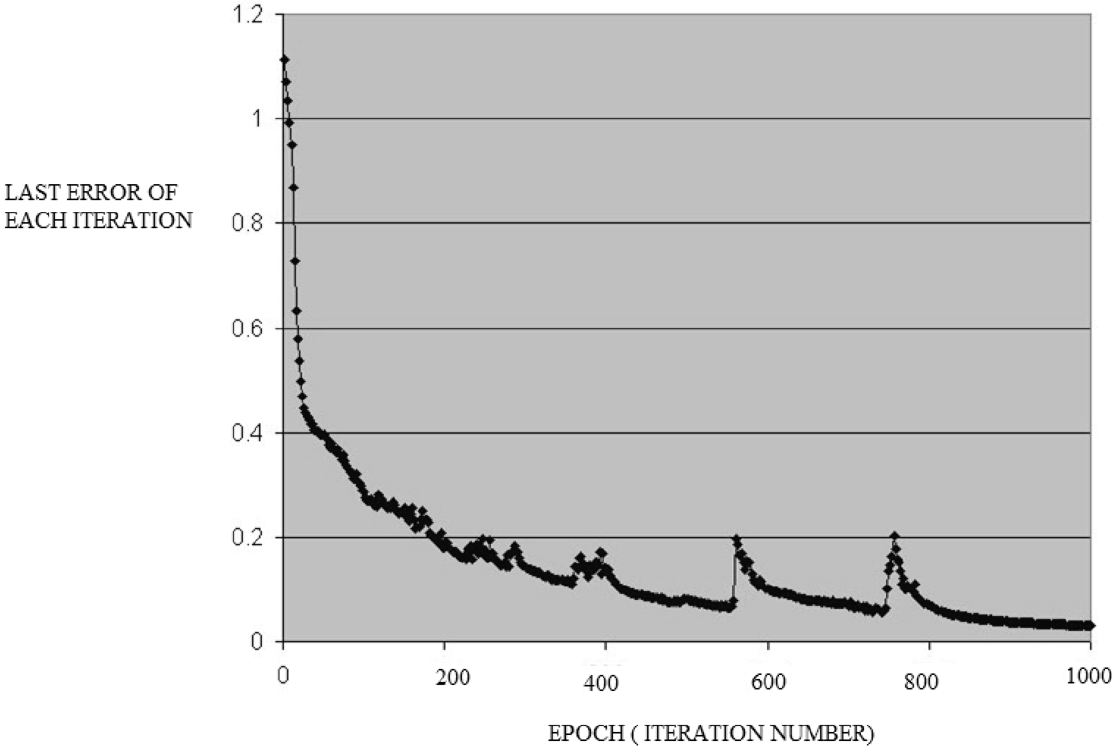

Figure 24: Trend of root-mean-squared (RMS) error for the entire training set over several training iterations

Chapter 10

CONCLUSION

10.1 DISCUSSIONS:

The speed trajectory of a dc motor has been successfully controlled by using a neural network based model reference adaptive control (MRAC) technique. A multi-layer perceptron is used to identify the motor dynamics. The trained neural network is then used to derive the appropriate control voltage for trajectory control of the rotor speed.

The controller design does not require explicit knowledge of the motor/load dynamics which is a useful feature when dealing with parameter and load uncertainties. The actual tuning of the controller only involves minimal effort. The motor displays good tracking performance for different speed trajectories. The robustness of the controller is positively established by evaluating its performance under noisy loading conditions. The controller remains stable for a wide range of sampling frequencies.

It is believed that better control performance can be obtained by training the neural network on-line. However, the resulting increased computational overhead at each sampling instant calls for a faster and more efficient software/hardware arrangement.

The unknown, time invariant, nonlinear operating characteristics of the dc motor and its load have been successfully captured by an ANN. The concepts of model reference adaptive control have been used in conjunction with the trained ANN to achieve trajectory control of the rotor speed.

10.2 FUTURE SCOPE:

The ability to deal with a noisy operating environment is always an important consideration for electric drives control. Noise can be introduced due to several reasons. Two main causes are motor-load parameter drift and quantization and resolution errors of the speed and/or position encoders. A high performance drive controller should be robust enough to maintain accurate tracking performance regardless of the noisy operating environment. *ANN* based controllers have the inherent noise rejection capability of the *ANN* built into them. Therefore *ANN* based controllers are expected to display superior performance under noisy operating environments.

The investigation of the controller performance under noise can be a cruitial phenomenon that can be studied through the necessary simulations.

Chapter 11

BIBLIOGRAPHY

Antsaklis, P. J. (April, 1990). Neural Networks in control systems. *IEEE control systems magazine, volume-10* , 3-5.

Cajueiro, D. O., & Hemerly, E. M. (December, 2003). Direct Adaptive Control using Feedforward Neural Networks. *Revista Control & Automation, Vol.14, No.4* , 348-358.

Chen, S., Billings, S., & Grant, P. (April, 1990). Non-linear system identification using neural networks. *IEEE Transactions, volume-51* , 1191-1214.

DOURATSOS, I., & BARRY, J. (2007). NEURAL NETWORK BASED MODEL REFERENCE ADAPTIVE. *International journal of information and systems sciences, volume-3* , 161-179.

Dzung, P., & Phuong, L. (2002). ANN - Control System DC Motor. *IEEE Transactions* , 82-90.

El-Sharkawi, M. A., & Weerasooriya, S. (September, 1989). Adaptive tracking control for high performance dc drives. *IEEE Transactions on Energy Conversion, volume-5* , 122-128.

El-Sharkawi, M. A., & Weerasooriya, S. (December, 1991). Identification and control of a dc motor using back-propagation neural networks. *IEEE Transactions on energy conversion, volume-6* , 663-669.

Hagan, M. T., Demuth, H. B., & Jesus, O. (2000). An Introduction To The Useof Neural Networks. *Electrical and Computer Engg.* , 1-23.

Kotaru, R., & Rubai, A. (June, 2000). Online Identification and control of a dc motor using learning adaptation of neural networks. *IEEE Transactions on industry applications, volume-36* , 935-942.

Liu, Z., Zhuang, X., & Wang, S. (2003). Speed Control of a DC Motor using BP Neural Networks. *IEEE Transactions* , 832-835.

Minkova, M., Rodgerson, J. L., & Harley, R. G. (1996). Current Limitations in the Adaptive Neural Speed Control of a DC Motor. *IEEE Transactions* , 837-842.

Moleykutty, G. (2008). Speed Control of Separately Excited DC Motor. *American Journal of Applied Sciences 5* , 227-233.

Odry, P., & Kavai, G. (2001). Neural Motor Control. *Polytechnical Engg.* , 55-66.

Parthasarathy, K., & Narendra, K. (March, 1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks, volume-1* , 4-27.

Weerasooriya, S., & El-Sharkawi, M. A. (March 1993). Laboratory Implementation of a Neural Network Trajectory Controller for a DC Motor. *IEEE Transactions on Energy Conversions, Vol. 8, No. 1* , 107-113.

Zilkova, J., Timko, J., & Girovsky, P. (2006). Nonlinear System Control Using Neural Networks. *Acta Polytechnica Hungarica, Vol.3; No.4* , 85-94.