



Neural Network Applications

Term Project: EE550

Submitted to:

Dr. Samir Al-Baiyat
Electrical Engineering Dept. KFUPM

By:

Imran Nadeem & Naveed R. Butt
220504 & 230353

May 28, 2005

Table of Contents

1	Introduction to Neural Networks.....	4
1.1	Introduction.....	4
1.2	Neuron & Artificial Neuron	5
1.3	Adaptation in NN's.....	6
1.4	COMMON NEURAL NETWORK ARCHITECTURES.....	7
1.4.1	Single-Layer Feed-forward Networks	8
1.4.2	Multilayer Feed-Forward Networks.....	8
1.4.3	Recurrent Networks.....	10
1.5	Applications of Neural Networks.....	11
2	LMS and RBF-NN's	13
2.1	The Least Mean Square (LMS) Adaptation Algorithm	13
2.2	RBF Neural Networks.....	15
2.3	DETAILED LEARNING ALGORITHM FOR RBF-NN's.....	17
2.3.1	Unsupervised learning.....	17
2.3.2	Supervised Learning.....	19
2.4	Relative Advantages of RBF-NN's.....	19
3	Neural Network Applications.....	20
3.1	Nonlinear Plant Identification.....	20
3.2	Adaptive Tracking of Nonlinear Dynamic Plants.....	22
3.2.1	The Plant	22
3.2.2	The Identifying Model	22
3.2.3	The Control Law	23
3.2.4	Simulation Results.....	23
	Bibliography	26

Abstract

Neural networks are parameterized nonlinear functions. Their parameters are the weights and biases of the network. Adjustment of these parameters results in different shaped nonlinearities. Typically these adjustments are achieved by a gradient descent approach on an error function that measures the difference between the output of the neural network and output of the actual system. Additionally there is no restriction on the unknown function to be linear. In this way, neural networks provide a logical extension to create nonlinear robust control schemes where there is no need to assume that the plant is a linear parameterization of known nonlinear functions.

These features of the Neural Networks make them an important area of research. We find Neural Networks applications in a variety of areas. They are used mainly for the purpose of identification and control. This report focuses on some advanced applications of Neural Networks in the area of nonlinear plant identification and adaptive control.

1 Introduction to Neural Networks

1.1 Introduction

Work on artificial neural networks commonly referred to as "neural networks" (NN) has been motivated right from its origin by the recognition that the human brain computes in an entirely different way than the conventional computer. The brain is a highly complex, nonlinear and parallel computer (information processing system). It has the capability to organize its structural constituents, known as neurons, so as to perform certain computations (e.g. pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. Consider for example, human vision, which is an information-processing task. It is the function of the visual system to provide a representation of the environment around us and, more important, to supply the information we need to interact with the environment. To be specific, the brain routinely accomplish perceptual recognition task (e.g. recognizing a familiar face embedded in an un-familiar scene) in approximately 100-200 ms, where as tasks of much lesser complexity may take days on a conventional computer.

How, then, does a human brain do it? At birth, a brain has great structure and the ability to built-up its own rules through what we usually refer to as "experience". Indeed, experience is built up over time, with the most dramatic development (i.e. hard wiring) of the human brain taking place during the first two years from birth: but the development continues well beyond that stage.

A "developing" neuron is synonymous with a plastic brain: Plasticity permits the developing nervous system to adapt to its surrounding environment. Just as plasticity appears to be essential to the functioning of neurons as information-processing units in the human brain, so it is with neural networks made up of artificial neurons. In its most general form, a neural network is a machine that is designed to model the way in which the brain performs a particular task or function of interest; the network is usually implemented by electronic components

or is simulated in software on a digital computer. The interest is confined to an important class of neural networks that perform useful computations through a process of learning. To achieve good performance, neural networks employ a massive interconnection of simple computing definition of a neural network viewed as an adaptive machine.

A neural network is a massively equivalent distributed process or made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network from its environment through a learning process.
- Inter neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

1.2 Neuron & Artificial Neuron

The two figures 1 & 2 compare the human neuron and the artificial neuron. For the human neuron the main functioning parts are:

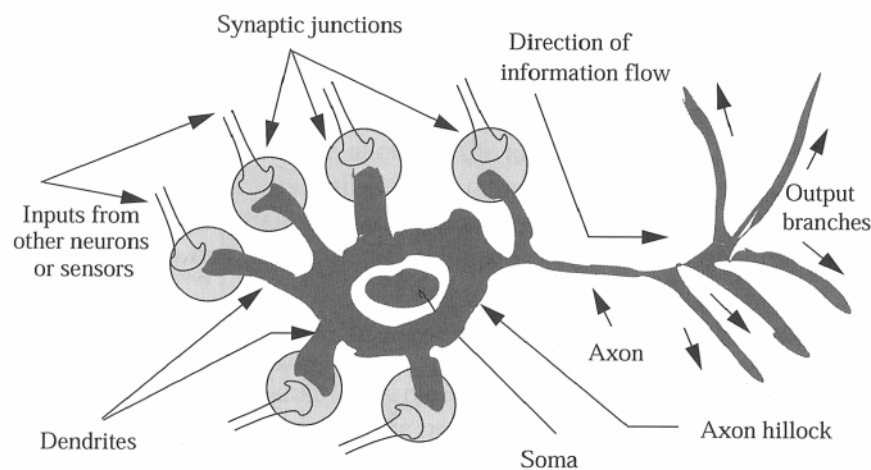


Figure 1: Human Neuron

- *Dendrites*: These act as the input points to the main body of the neuron.
- *Synapse*: This is the storage area of the past experience.
- *Soma*: It receives synaptic information and performs further processing on the information.
- *Axon*: This is the output line for the neuron.

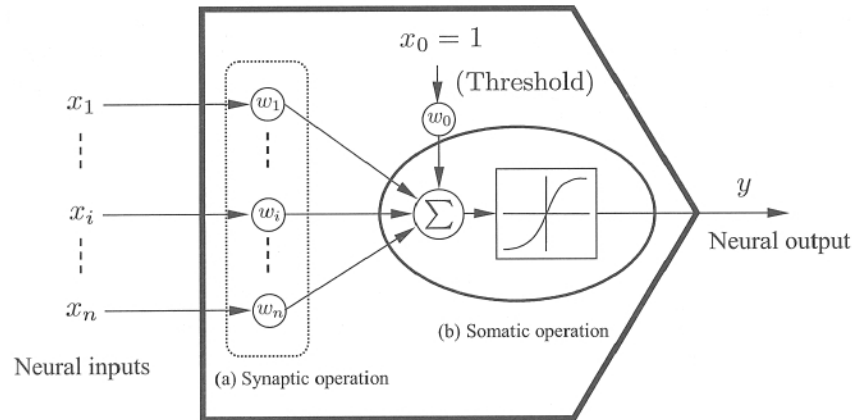


Figure 2: Artificial Neuron

In artificial neural networks, the synaptic and somatic operations are emulated as follows:

- *Synaptic Operation*: The input weights act as storage for knowledge (and therefore, as memory for previous experiences).
- *Somatic Operation*: The somatic operation is provided by various mathematical operations such as aggregation, thresholding, nonlinear activation and dynamic processing to the synaptic inputs.

1.3 Adaptation in NN's

The procedure that is used to perform the learning process is called a learning algorithm (fig. 3), the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective.

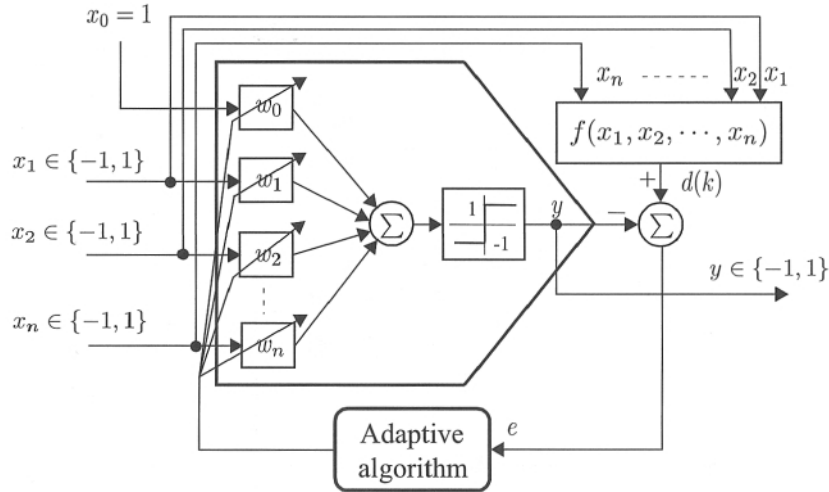


Figure 3: Adaptation in NN's

The modification of synaptic weights provides the traditional method for the design of neural networks. Such an approach is the closest to linear adaptive filter theory, which is already well established and successfully applied in many diverse fields. However, it is also possible for a neural network to modify its own topology, which is motivated by the fact that neurons in the human brain can die and then new synaptic connections can grow.

1.4 COMMON NEURAL NETWORK ARCHITECTURES

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. We may therefore speak of algorithms (rules) used in the design of neural networks as being structured. In general we may identify three fundamentally different classes of network architectures:

1.4.1 Single-Layer Feed-forward Networks

In a layered neural network the neurons are organized in the form, of layers. In the simplest form of layered network, we have an input layer of source nodes that projects onto an output layer of neurons (computation nodes), but not vice versa. In other words, this network is strictly a feed-forward or acyclic type. It is illustrated in the figure for the case if four nodes in both the input and output layers. Such a network is called a single-layered network, with the name "single-layer" referring to the output layer of computation nodes (neurons). We do not count the input layer of source nodes because no computation is performed there.

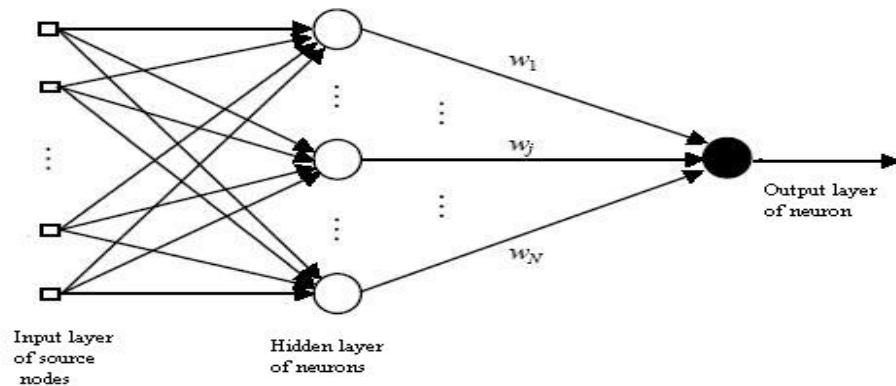


Figure 4: Single Layer Feedforward NN

1.4.2 Multilayer Feed-Forward Networks

The second class of a feed-forward neural network distinguishes its self by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. The function of the hidden neuron is to interfere between the external input and the network output in some useful manner.

Multilayer feed forward networks are an important class of neural networks. Typically, the network consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes, and

an out-put layer of computation nodes. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These neural networks are commonly referred to as multilayer perceptrons (MLP's), which represent a generalization of the single-layer perceptron.

The source nodes in the input layer of the network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computation nodes) in the second layer (i.e., the first hidden layer). The output signals of the second layer are used as an input to the third layer, and so on for the rest of the network. Typically the neurons at each layer of the network have as their inputs the outputs of the preceding layers only. The set of output signals of the neurons in the output (final layer) constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer.

Multilayer perceptrons have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as the error back-propagation algorithm. This algorithm is based on the error-correction learning rule. As such, it may be viewed as a generalization of an equally popular adaptive filtering algorithm: the least-mean-square (LMS) algorithm.

Basically, error back-propagation learning consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the networks are all fixed. During the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with an error-correction rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is then propagated backward through the network, against the

direction of synaptic connections, hence the name "error back-propagation." The synaptic weights are adjusted to make the actual response of the network move closer to the desired response in a statistical sense. The error back-propagation algorithm is also referred to in the literature as the back-propagation algorithm.

Figure 5 shows the architectural graph of a multilayer perceptron with two hidden layers and an output layer. Signal flow through the network progresses in a forward direction, from left to right and on a layer-by-layer basis.

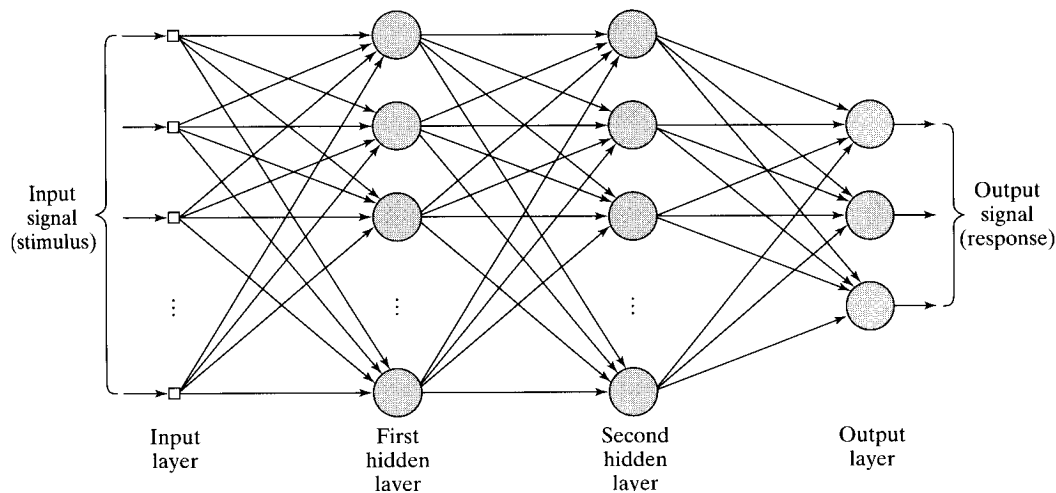


Figure 5: Multi-layer Feedforward NN's

The neural network in the figure is said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. If, however, some of the communication links are missing from the network, we say that the network is partially connected.

1.4.3 Recurrent Networks

A recurrent neural network distinguishes itself from the feed-forward network in that it has at least one feedback loop. For example, a recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the input of all input neurons.

The presence of feedback loops has a profound impact on the learning capability of the network and on its performance. Moreover, the feedback loops involve the

use of particular branches composed of unit-delay elements which result in a nonlinear dynamical behavior, assuming that the neural network contains nonlinear units.

1.5 Applications of Neural Networks

Neural networks are applicable in virtually every situation in which a relationship between the predictor variables (independents, inputs) and predicted variables (dependents, outputs) exists, even when that relationship is very complex and not easy to articulate in the usual terms of "correlations" or "differences between groups." A few representative examples of problems to which neural network analysis has been applied successfully are:

- **Detection of medical phenomena.** A variety of health-related indices (e.g., a combination of heart rate, levels of various substances in the blood, respiration rate) can be monitored. The onset of a particular medical condition could be associated with a very complex (e.g., nonlinear and interactive) combination of changes on a subset of the variables being monitored. Neural networks have been used to recognize this predictive pattern so that the appropriate treatment can be prescribed.
- **Stock market prediction.** Fluctuations of stock prices and stock indices are another example of a complex, multidimensional, but in some circumstances at least partially-deterministic phenomenon. Neural networks are being used by many technical analysts to make predictions about stock prices based upon a large number of factors such as past performance of other stocks and various economic indicators.
- **Credit assignment.** A variety of pieces of information are usually known about an applicant for a loan. For instance, the applicant's age, education, occupation, and many other facts may be available. After training a neural network on historical data, neural network analysis can identify the most

relevant characteristics and use those to classify applicants as good or bad credit risks.

- **Condition Monitoring.** Neural networks can be instrumental in cutting costs by bringing additional expertise to scheduling the preventive maintenance of machines. A neural network can be trained to distinguish between the sounds a machine makes when it is running normally ("false alarms") versus when it is on the verge of a problem. After this training period, the expertise of the network can be used to warn a technician of an upcoming breakdown, before it occurs and causes costly unforeseen "downtime."
- **Engine management.** Neural networks have been used to analyze the input of sensors from an engine. The neural network controls the various parameters within which the engine functions, in order to achieve a particular goal, such as minimizing fuel consumption.
- **Signature analysis,** as a mechanism for comparing signatures made (e.g. in a bank) with those stored. This is one of the first large-scale applications of neural networks in the USA, and is also one of the first to use a neural network chip.
- **Process control,** most processes cannot be determined as computable algorithms. Neural Networks can be used to adaptively control the process
- **Nonlinear Identification & Adaptive Control,** This is one of the main areas of application of the neural networks. Neural Networks find applications in situations where the plant dynamics are uncertain or unmodeled.

2 LMS and RBF-NN's

2.1 The Least Mean Square (LMS) Adaptation Algorithm

As discussed in section 1, the learning process in the neurons involves updating of certain “weights”. A number of adaptation algorithms are available in literature. The criteria/cost functions used for adaptation and the methods are usually derived from the richly developed field of adaptive filter theory. In the following we present one of the most commonly used adaptation algorithms, the LMS.

Let:

$w[n]$: time varying neuron tap weights

$v[n]$: input to neuron

$d[n]$: desired response

$\tilde{s}[n]$: actual output of neuron

J : cost function (the mean square error)

The estimation error is the difference between the desired response and the estimated output:

$$e[n] = d[n] - \tilde{s}[n]$$

using $\tilde{s}[n] = w^H v[n]$ gives,

$$e[n] = d[n] - w^H v[n]$$

The mean square error (cost function) is defined as:

$$J = E \left\{ |e[n]|^2 \right\} = E \left\{ |d[n] - w^H v[n]|^2 \right\}$$

Thus the cost function J is a function of vector w . Minimizing J with respect to the complex tap weights w leads to the set of equations called the Wiener-Hopf equations. If we limit the number of taps to M then we obtain the matrix formulation of the Wiener- Hopf equations from which the solution is obtained as

$$w_o = R^{-1}p$$

where

$$R = E\{v[n]v^H[n]\} \quad \text{and} \quad p = E\{v[n]d^*[n]\}$$

This method of solution requires inversion of a matrix. An alternative adaptive method of solution is the Steepest Descent algorithm. It can be shown that the cost function J has the shape of an M -dimensional bowl whose minimum is at the optimal solution of w . The steepest descent algorithm moves the tap weights towards the minimum of the J bowl at every iteration by moving them in the direction opposite to the gradient vector:

$$\nabla_w J = 2Rw - 2p$$

Thus we have an iterative definition for the tap weight updates:

$$\begin{aligned} w[n+1] &= w[n] - \frac{\mu}{2} \nabla_w J \\ &= w[n] + \mu[p - Rw[n]] \end{aligned}$$

μ is known as the adaptation step. ($\mu > 0$)

In practice, although we do not know R and p , we can use their instantaneous estimates:

$$\hat{R}[n] = v[n]v^H[n]$$

$$\hat{p} = v[n]d^*[n]$$

This approach is known as the Least Mean Squares (LMS) method which is a member of a particular class of algorithms called the stochastic gradient algorithms. Thus the adaptation equation is given as:

$$\hat{w}[n+1] = \hat{w}[n] + \mu v[n]e^*[n]$$

As we shall see, this equation is used in a variety of adaptation algorithms each having its own definition for the error functions.

2.2 RBF Neural Networks

Among the vast variety of neural networks, the RBF-NN is a quire commonly used structure. The design of a RBF-NN in its most basic form consists of three separate layers. The input layer is the set of source nodes (sensory units). The second layer is a hidden layer of high dimension. The output layer gives the response of the network to the activation patterns applied to the input layer. The transformation from the input space to the hidden-unit space is nonlinear. On the other hand, the transformation from the hidden space to the output space is linear.

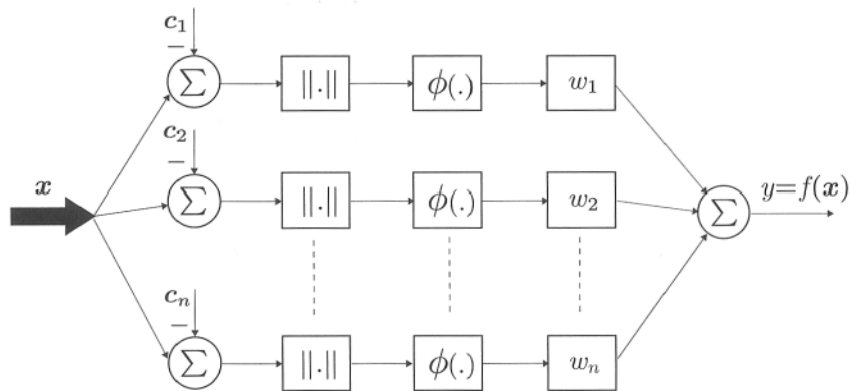


Figure 6: RBF-NN Basic Structure

With reference to the figure above, the output $y(t)$ is a weighted sum of the outputs of the hidden layer, given by

$$\hat{y}(t) = \sum_{i=1}^n w_i \phi(\|u(t) - c_i\|), \quad (3.3)$$

where

$u(t)$ is the input

$\phi(\cdot)$ is an arbitrary nonlinear radial basis function

$\|\cdot\|$ denotes the norm that is usually assumed to be Euclidean

c_i are the known centers of the radial basis functions

w_i are the weights

Radial functions are a special class of functions. Their characteristic feature is that their response decreases (or increases) monotonically with distance from a central point and they are radially symmetric. The centre, the distance scale, and the precise shape of the radial function are parameters of the model. There are a variety of radial functions available in literature. The most commonly used one is the Gaussian radial filter, which in case of a scalar input is

$$h(x) = \exp\left(-\frac{(x-c)^2}{\beta^2}\right)$$

Its parameters are its centre c and its radius β (width), Figure 1.3 illustrates a Gaussian RBF with centre $c = 0$ and radius $\beta = 1$. A Gaussian RBF monotonically decreases with distance from the centre

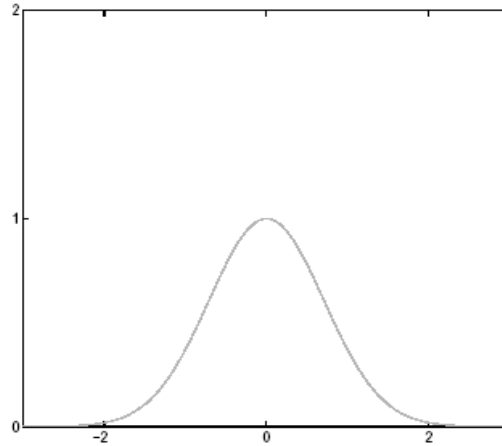


Figure 7: Gaussian function Profile

A summary of the characteristics of the RBF-NN's is given below

- They are two-layer feed-forward networks.
- The hidden nodes implement a set of radial basis functions (e.g. Gaussian functions).
- The output nodes implement linear summation functions as in an MLP.
- The network training is divided into two stages: first the weights from the input to hidden layer are determined, and then the weights from the hidden to output layer.
- The training/learning is very fast.
- The networks are very good at interpolation.

2.3 DETAILED LEARNING ALGORITHM FOR RBF-NN's

The whole algorithm of RBF network learning may be split into two phases:

- Hidden layer learning or basis function selection (unsupervised learning).
- Fitting of outputs in a transformed feature space (supervised learning).

2.3.1 Unsupervised learning

The first phase is an unsupervised learning. It does not use any information on target outputs and deals only with a set of inputs. At this stage we have to:

1. Select a number of radial basis functions.
2. Select a center for each basis function and
3. Select a value for the parameter β (width), which characterizes the basis function range of definition (the range of its influence). A too large value of β forms too narrow basis functions.

In step1, usually all the RBF's are chosen to be the same. There are many algorithms available for the selection of the centers (step-2), one of the more popular ones is the K-means clustering algorithm which goes as follows

Given m data points, select l as the number of clusters such that $l < m$

Take the first l learning data as the center vectors for the l clusters

$$c_j = x_j, \quad j = 1, 2, \dots, l;$$

Assign the remaining data points to one of the clusters with the least distance criterion. Recompute the center vectors using the new mean, that is

$$c_j = \frac{1}{m_j} \sum_{i \in c_j} x_i; \quad 1 \leq j \leq l$$

where m_j is the number of data points belonging to the j th cluster.

As soon as the clustering algorithm is complete we may move to the selection of the variance or width parameter β (step-3). These parameters control the amount of overlap of the radial basis functions as well as the network generalizations. A small value yields a rapidly decreasing function, whereas a large value results in a more gently varying function. The mostly commonly used method for the selection of the width parameter for a cluster is to take it equal to the average distance between the data in the cluster and center of the cluster.

2.3.2 Supervised Learning

The second phase is a supervised learning. The goal is to fit outputs with a linear function of nonlinear transformed inputs. Any gradient optimization method may be used, but the LMS (discussed above) is used most often.

2.4 Relative Advantages of RBF-NN's

- Many pattern recognition experiments show that the RBF-NN's are superior over other neural network approaches in the following senses.
- RBF-NN's are capable of approximating nonlinear mappings effectively.
- The training time of the RBF-NN's is quite low compared to that of other neural network approaches such as the multi-layer perceptron.
- The RBF-NN's produce classification accuracies from 5% to 10% higher than accuracies produced by the back propagation algorithm.
- The RBF-NN's are quite successful for identifying regions of sample data not in any known class because they use a non-monotonic transfer function based on the Gaussian density function.

3 Neural Network Applications

3.1 Nonlinear Plant Identification

One of the major areas of application of the neural networks is in the identification of nonlinear plants. The RBF-NN was introduced in section 2. Here we make use of the Gaussian RBF-NN to identify the nonlinear system known as the continuous stirred tank reactor. The nonlinear model of the continuous stirred tank reactor when the sampling time is chosen as 0.05 seconds is as follows.

$$\begin{aligned} y(t+1) = & 0.8606y(t) - 0.0401y^2(t) + 0.0017y^3(t) - 0.000125y^4(t) + 0.0464u(t) \\ & - 0.045y(t)u(t) + 0.0034y^2(t)u(t) - 0.00025y^3(t)u(t) - 0.0012u^2(t) + \\ & 0.0013y(t)u^2(t) - 0.0001458y^2(t)u^2(t) + 0.00002083u^3(t) - 0.00002083y(t)u^3(t) \end{aligned}$$

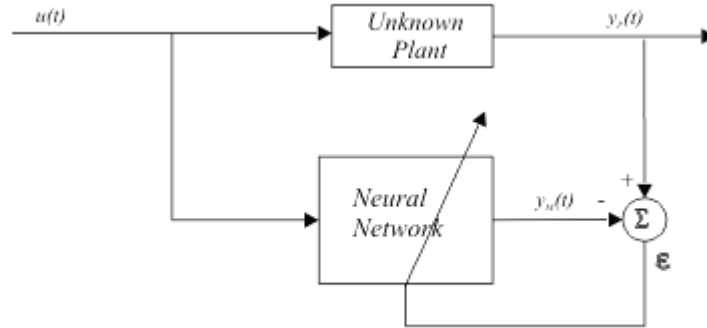


Figure 8: Identification Structure

The RBF-NN assumes no prior knowledge of the system parameters and tries to identify the system online. Simulations were carried out using SIMULINK. The number of linear combiner weights was chosen to be ten. The neural network worked well and was able to identify the nonlinear plant online. The simulation results are given in the figures 9 & 10.

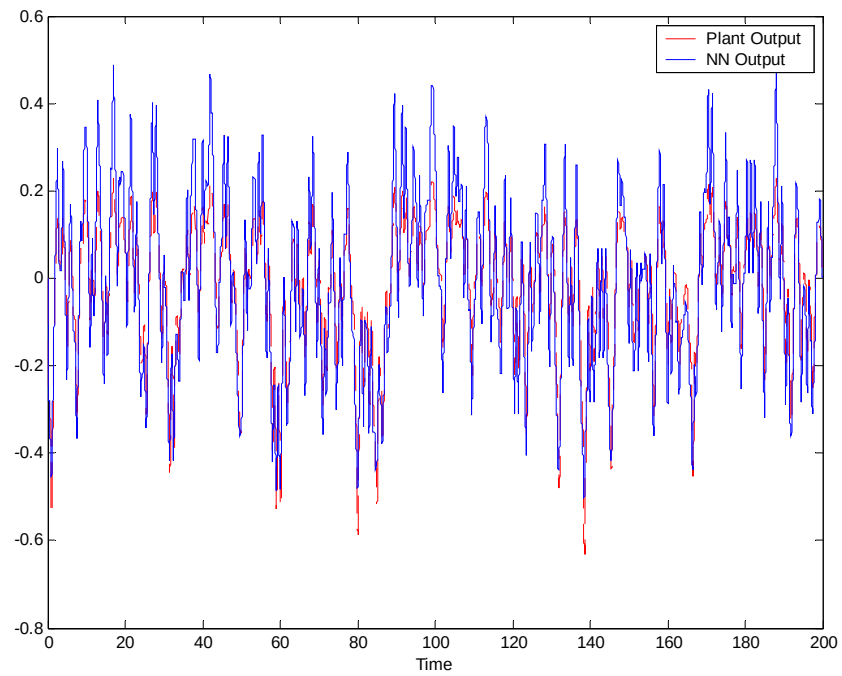


Figure 9: Identification simulation

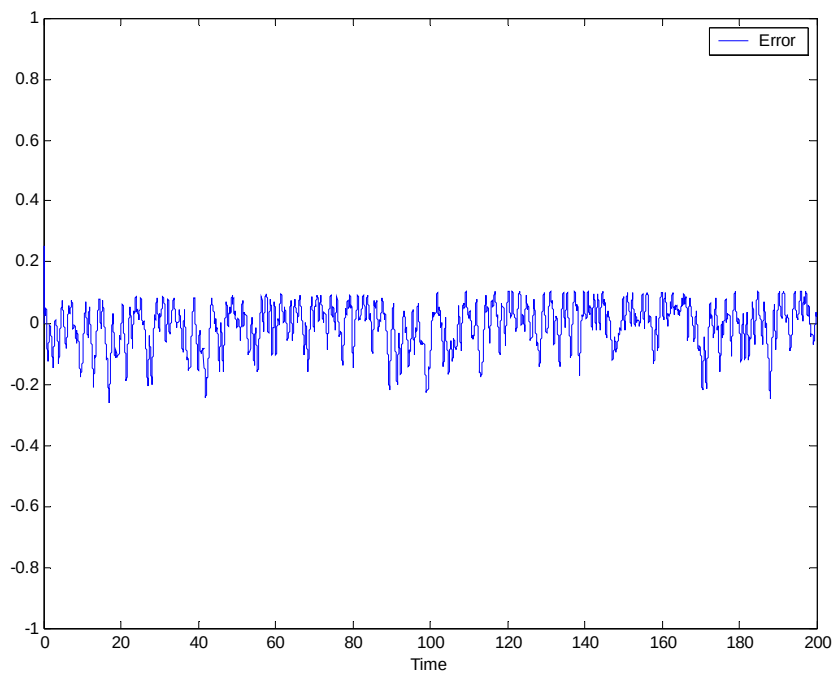


Figure 10: Mismatch error

3.2 Adaptive Tracking of Nonlinear Dynamic Plants

The adaptive control of nonlinear dynamic plants is an extremely important area of research. It involves the online identification of the plant and development of a controller based on this identified plant. The identification part is generally carried out using powerful neural networks. A number of techniques have been suggested in literature. Here we utilize one of the more recent approaches. The main structure is depicted below. The basic structure is that of an IMC (Internal Model Control). The identification task is carried out utilizing the Gaussian RBF-NN. A control law is synthesized which is based on the identified system parameters.

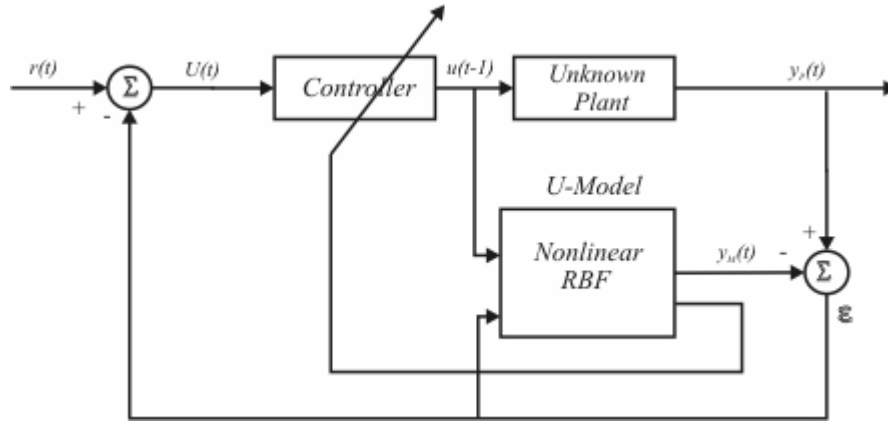


Figure 11: Adaptive Tracking

3.2.1 The Plant

We assume a stable nonlinear dynamic plant whose functional parameters or the functional structure need not be known.

3.2.2 The Identifying Model

We identify the plant online using the radial basis function with the following structure:

$$y_M(t) = a_1 u(t-1) + \hat{b}_1 \Phi(u(t-1)) + \hat{b}_2 \Phi(u(t-2)) + \dots + \hat{b}_n \Phi(u(t-n)) \quad (3.1)$$

where the parameter a_1 is selected in advance and the parameters $\hat{b}_1, \hat{b}_2, \dots, \hat{b}_n$ are estimated using the normalized least mean square algorithm. Φ can be any function used in neural networks. Here we use the Gaussian radial basis function.

3.2.3 The Control Law

To simplify the synthesis of the control law we use the equivalent U-model for the RBF of equation (3.1):

$$y_M(t) = \alpha_0(t) + \alpha_1(t)u(t-1) \quad (3.2)$$

where

$$\alpha_0(t) = \hat{b}_1 \Phi(u(t-1)) + \hat{b}_2 \Phi(u(t-2)) + \dots + \hat{b}_n \Phi(u(t-n))$$

$$\alpha_1(t) = a_1$$

Using the U-model of equation (3.2) which is linear with respect to the control term $u(t-1)$, the controller has the simplified form as follows:

$$u(t-1) = \frac{U(t) - \alpha_0(t)}{\alpha_1(t)} \quad (3.3)$$

This controller is clearly an inverse of the identified plant.

3.2.4 Simulation Results

We carried out simulations on the following nonlinear Hammerstein model

$$y(t) = 0.5y(t-1) + x(t-1) + 0.1x(t-2)$$

$$x(t) = 1 + u(t) - u^2(t) + 0.2u^3(t)$$

The system was modeled according to equation 3.1, and then its equivalent U-model (equation 3.2) was used to synthesize the control law (equation 3.3). The first parameter a_1 was selected as 5, while the number of linear combination weights was four ($\hat{b}_1, \hat{b}_2, \hat{b}_3$ & $\hat{b}_4$). All weights were initialized to 0 and the step size was chosen to be 0.1

The results are depicted in figures 12,13 & 14.

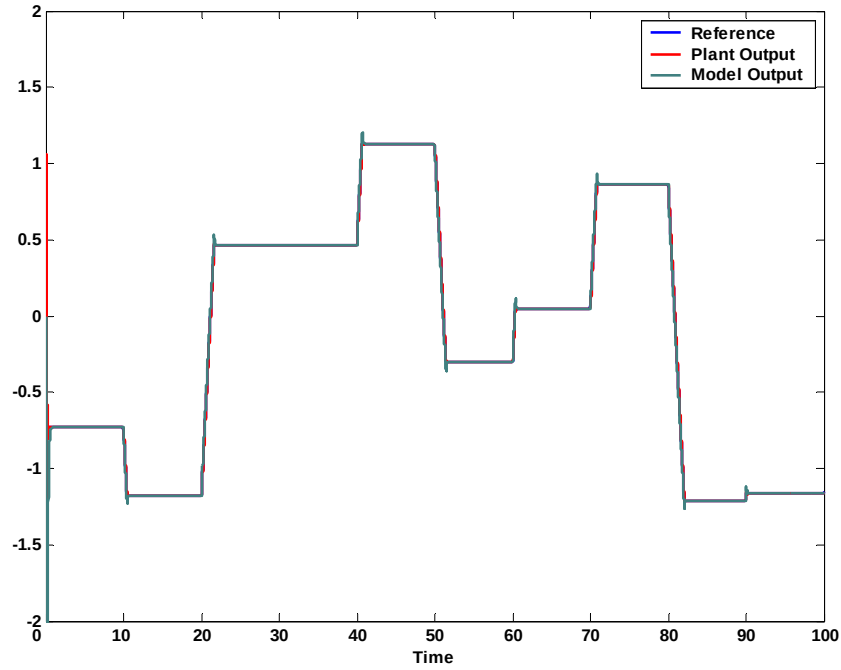


Figure 12: Tracking simulation

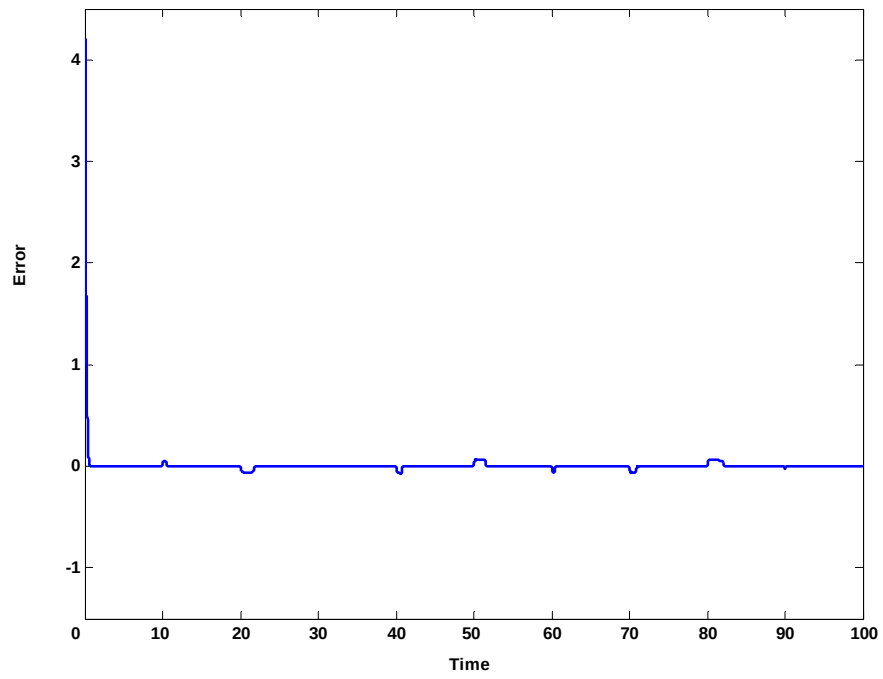


Figure 13: Tracking error

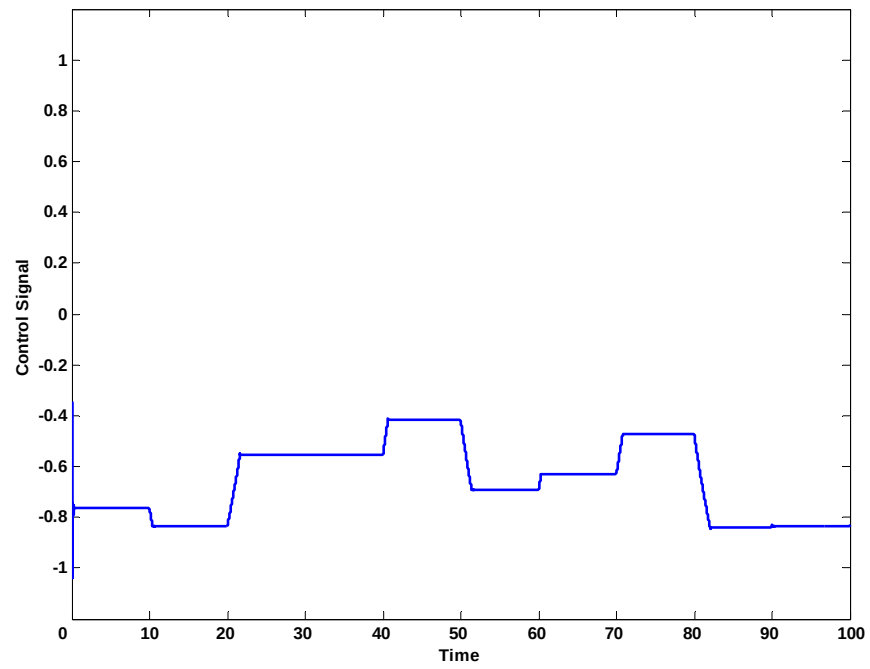


Figure 14: Control Input

Bibliography

The following sources were consulted in the making of this report

- Gupta, M. M., Jin, L. and Homma, N., "Static and dynamic Neural Networks", *IEEE Press*, 2003.
- Shafiq M. and Riyaz S.H., "Internal Model Control Structure Using Adaptive Inverse Control Strategy.", *The 4th Int. Conf. on Control and Automation (ICCA)*, pp.59-59, 2003.
- Spooner, J. T. , Maggiore, M. , Ordonez, R. and Passino, K. M., "Stable adaptive control and estimation for nonlinear systems", *Wiley-Interscience*, NY , 2002.
- Zhu, Q. M. and Guo. L. Z., "A pole placement controller for nonlinear dynamic plants", *J. Systems and Control Engineering*, Vol. 216 (part I), pp. 467 – 476 , 2002.