NEURAL TREES AND THEIR APPLICATION TO CLASSIFICATION AND CONTROL

by

ALOK KANTI DEB

DEPARTMENT OF ELECTRICAL ENGINEERING

Submitted
In fulfillment of the requirements
for the degree of

Doctor of Philosophy

to the

INDIAN INSTITUTE OF TECHNOLOGY, DELHI

MAY 2005

Neural networks

To succeed, you must have
tremendous perseverance,
tremendous will:
'I will drink the ocean',
says the persevering soul,
'at my will',
'mountains will crumble up'.
Have that sort of energy,
that sort of will, work hard,
and you will reach the goal.

Swami Vivekananda

To

My Parents

CERTIFICATE

This is to certify that the thesis entitled "Neural Trees and their Application to Classification and Control", which is being submitted by Mr. Alok Kanti Deb to the Department of Electrical Engineering, Indian Institute of Technology, Delhi, for the award of the degree of Doctor of Philosophy, is a record of bonafide research work he has carried out under our guidance and, in our opinion, it has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted to any other university or institute for the award of a degree or a diploma.

Jayadeva

Prof. Jayadeva

M Gopal

Prof. M. Gopal

Department of Electrical Engineering

Indian Institute of Technology

Hauz Khas, New Delhi-110016

INDIA

ACKNOWLEDGEMENTS

I have been fortunate and indebted to work under my two supervisors at the Department of Electrical Engineering, IIT Delhi, Prof. Jayadeva and Prof. M. Gopal for their enthusiasm, guidance, interest, advice, and support, and for their understanding and help when I needed them. Apart from the academic inputs, I greatly cherish the qualities and skills I have imbibed by my association with them and hope that I shall continue to have their blessings in future. I am grateful to be a part of the P. G. Electronics Laboratory and have benefited greatly from the unique environment.

I am indebted to Prof. Jayadeva for introducing me to the area of neural networks and support vector machines and for the numerous valuable discussions and new ideas which also made research enjoyable and worthwhile. I am grateful to Prof. Gopal for his support and inspiration to my motivation to pursue Doctoral programme here and for his valuable inputs from aspects of control theory. I owe to both of them for having given so much of their time and effort.

I also owe a ton to Prof. Suresh Chandra who provided valuable inputs from mathematical programming techniques in my research. Apart from my supervisors, I would also express my gratitude to my other research committee members, Prof. A. N. Jha, Prof. S. Chandra and Dr. R. K. P. Bhatt for their valuable advice from time to time and guiding the course of the work. I also cherish my association with Prof. S. C. Dutta Roy, Prof. S. Prasad, Prof. G. S.

Visweswaran, Prof. B. Bhaumik, Prof. R. K. Patney, Prof. J. K. Chatterjee, Prof. V. Chandra, Dr. I. N. Kar, Dr. Akhila Sinha, and Mr. Ram Lal during my stay at IIT Delhi.

I am grateful to Prof. Donald C. Wunsch II, Dr. Derong Liu, Dr. Danil V. Prokhorov, Prof. Olvi Mangasarian, Prof. S. S. Keerthi, Dr. J. A. K. Suykens, and Dr. Oliver Chapelle for some valuable deliberations through correspondence that greatly helped during this work. I have been fortunate to have some interesting discussions on machine learning with Prof. Klaus Obermayer during his brief visit to India and the close proximity to Dr. Ravi Kothari helped augment my knowledge on statistical learning theory.

I am particularly thankful to Sameena Shah for meticulously reading the manuscript and thank Reshma Khemchandani for help during part of my work. Thanks are also due to many of my colleagues, Syed Atiqur Rehman, Mr. Akhil Ranjan Garg, Mr. Atanendu Sekhar Mandal, Dr. Mona Mathur, Mr. Lalit Jiwani, Mr. Rajan Bhatt, Mr N. K. Verma, Mr. R. Saha, Mr. A. Karmakar, Mr. Anindya Ghosh, Md. Ahmaruzaman and many others whose association during the entire period of work, I would remember throughout my life. Special thanks go to fellow colleague, Mr. Sukhendu Deb Roy who was a constant companion during extended hours in the laboratory. I would also acknowledge the support of non-teaching staff of the department notably, Mr. K. C. Sharma, Mr. Jaipal Singh, Mr. D. Jaitley, Mr. Rakesh Kumar, Mr. Jile Singh, Mr. Y. S. Aswal, Mr. C. S. Prasad and Mr. S. P. Singh.

I would like to thank IIT Delhi for granting me full-time teaching assistantship that provided financial support during my research work and I hope the experience gained in the tutorials and laboratories would help me in my ambition to pursue a career in engineering education and research.

I owe a debt of gratitude to my parents and family for their understanding, support, love and patience, without which this thesis would not have seen the light of the day. In spite of their difficulties and having to do at hard times because of my staying away for prolonged periods, I am particularly thankful to my parents who are always a pillar of support to all my constructive endeavors.

Finally, I acknowledge all others whose names could not be accommodated in this brief acknowledgement.

Alok Kanti Deb

(Alok Kanti Deb)

ABSTRACT

This thesis deals with the development of constructive neural networks for the problems of binary classification and one-dimensional function approximation by formulating the problems in a linear programming framework. The use of linear programming simplifies the problem definition and eases solvability. The final hypothesis is generated by using SVMs, widely known for their superior generalization power. In the case of binary classification problems where the numbers of samples of the two classes are skewed, two new types of objective functions have been proposed that enable 100% learning of the data. A complete training scheme, termed as SVM N-tree has been proposed, in the sequel, and its efficacy has been tested on real benchmark data sets. A linear programming formulation has also been proposed for developing tree type networks for one-dimensional function approximation. Its performance has been compared with that of multilayer networks employing different activation functions. A new technique for multi-dimensional function approximation has been formulated that involves a single matrix inversion with obvious computational advantages. We then discuss a set of applications of the proposed networks. In the first application, the Tree type neural network has been used as a 'critic' in an action-dependent heuristic dynamic programming type of adaptive critic design. Simulation results demonstrating the stability and robust performance of the trained controller have been presented. In the second application, the SVM based tree type neural network was used as the main

controller for generating the inverter switching sequences of a direct torque control drive of an induction motor. In conclusion, the salient contributions of the thesis have been outlined and the scope of future work has been presented.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

E	Sum-squared error (SSE) cost function
t_{pi} / o_{pi}	Target / Actual output of the i-th neuron due to p-th pattern
w_{ij}	Weight connecting the output of the j-th neuron to the input of the i-th neuron
Δw_{ij}	Incremental change in w_{ij}.
w'_{ij}	Updated value of w_{ij} after incremental change
Sigmoid	Sigmoidal activation function; $f(net) = \frac{1}{1+e^{-net}}$
Tanh	Tangant hyperbolic activation function given by $f(net) = \frac{1-e^{-net}}{1+e^{-net}}$
net	Total input to a neuron
η	Learning factor
d^+, d^-	Distance of the terminal patterns of a binary classification problem from the separating hyperplane
$R(f)$	Risk on the training data due to the discriminant function f
$R_{emp}(f)$	Empirical error on the training data due to the discriminant function f
h	VC-dimension / Hyperplane set
N	Number of samples of the training set
N_v	Number of samples of the validation set
Ω	A monotonic function
n	Dimension of the input space

w_0 / w	Bias / n-dimensional weight vector
w_d	d-th component of $[w_0 \quad w]^T$
$I / I_1 / I_2$	Index set of the samples of the training set / samples of Class 1 / samples of Class 0
x^i	i-th input vector
x_t^i	i-th input vector of training set
x_v^i	i-th input vector of validation set
x	input vector
y^i	i-th output vector
y_t^i	i-th output vector of training set
y_v^i	i-th output vector of validation set
y	output vector
$\Re / \Re^n$	$1/n$-dimensional real space
S_t / S_v	Training set / Validation set
S	Set of functions
H	Hyperplane set / Concatenation of X and e
Δ	Margin of separation
$x_d^{(i)}$	d-th component of $[1 \quad x^i]^T$
$w_d^{(+)}$, $w_d^{(-)}$	Non-negative variables used to represent w_d; $d = 0,1,\ldots,N$
σ^2	Square of the variance of a Gaussian kernel
m / k	Number of samples in Class 1 / Class 0
ϕ	Null set, Mapping function

$\cup$	Union
$\cap$	Intersection
$\sum$	Summation
C_1/ C_2/ C_3/ C_4	Class 1/2/3/4
threshold(net)	Threshold Logic activation function; $\text{threshold}(net) = \begin{cases} 1, & \text{if } net > 0 \\ 0, & \text{otherwise} \end{cases}$
N_1/ N_2/ N_3/ N_4	Number of samples in Class 1/2/3/4
w_a / w_b	Weight connecting neuron A / neuron B to the parent
card(S)	Cardinality of a set S
x_a^i / x_b^i	Desired output of neuron A / neuron B for the i-th sample
$f(.)$, $f_i(.)$	Nonlinear Function
$f'(.)$, $f_i'(.)$	Derivation of function
$u(.)$	Vector of Control Inputs
b_1, b_2, b_3, b_4	Bit in position 1/2/3/4
λ, μ	Weighting factors to the slack variables
M	Parameter that determines the value of λ and μ
$\|\|.\|\|_2$, L_2	2-norm
K	Number of subsets (folds) for cross validation
Θ	p-dimensional vector of parameters
Γ^p	Set of all values of Θ
D	Domain of the input vector x
G	Class of approximators

$\hat{f}$	Approximator to f
ε	Non-negative limit of approximation error
G_1, G_2	Classes of functions
d	Degree of a polynomial
s	Scale parameter
$I_k; k = 1,2,\ldots,m$	k-th disjoint sub-interval
g_j	j-th element of a basis
b_0	Constant term in the function approximator as a sum of basis functions
b_j	Weight associated with j-th basis function
purelin(net)	Purely linear activation function; purelin$(net) = net$
satlins(net)	Symmetric Saturating linear activation function $\text{satlins}(net) = \begin{cases} -1 & \text{, if } net \le -1 \\ net & \text{, if } -1 < net < 1 \\ 1 & \text{, if } net \ge 1 \end{cases}$
poslin(net)	Positive linear activation function; poslin$(net) = \max(0, net)$
satlin(net)	Saturating linear activation function $\text{satlin}(net) = \begin{cases} 0 & \text{, if } net \le 0 \\ net & \text{, if } 0 < net \le 1 \\ 1 & \text{, if } net > 1 \end{cases}$
$s_i^{(+)}, s_{i1}^{(+)}, s_{i2}^{(+)}$	Non-negative surplus variables
$s_i^{(-)}, s_{i1}^{(-)}, s_{i2}^{(-)}$	Non-negative slack variables
e^i	Error for the i-th sample
z^i / z	Actual output of i-th sample / Variable
y_A^i, y_B^i	Desired output of type-A and type-B neuron for the i-th sample

y^i_{parent} / z^i_{parent}	Required / actual outputs of the parent neuron (A or B) for the i-th input sample
H_1, H_2	Hypotheses
SSE / SSE_{lim}	Sum Squared Error / Limit of Sum Squared Error
$x(t)$	n-dimensional time dependent plant states
$u(t)$	m-dimensional time dependent control vector
P	Plant
t	Time / bias
$R(t)$	Vector of plant states or a concatenation of the plant states and the control vector
$J(t)$	Time dependant performance index
γ	Discount factor, regularization parameter
$U(.)$	Utility function
$E_1(t)$, $E_2(t)$	Error functions
W_c	Critic parameters
$\lvert a \rvert_\varepsilon$	Penalization of loss a depending on ε
ξ_i	Error in approximating the i-th sample
L_{LS} / L	Least Squares Lagrangian / Lagrangian
α	Vector of Lagrangian multipliers
α_i	Lagrangian multiplier for the i-th constraint
α	Angle between stator flux-linkage space vector and stator-current space vector
K_{ij}	ij-th element of the Kernel matrix $\boldsymbol{K}$
z^{-1}	Delay

A	Mapping function for the Action element
X	Column-wise inputs of a data set in matrix form.
T_c	Friction in the motion of the cart
m_c / m_p	Mass of the cart / pole
∇	Vector differential operator
g	Acceleration due to gravity
J	Moment of Inertia
f_c / f_p	Friction coefficient of cart / pole rotation
D_p	Moment of friction in the angular axis of the pendulum
l	Displacement between mass center and the axis of rotation
$x(t)$	Displacement of the cart
$\dot{x}(t)$	Cart velocity
$\theta(t)$	Angular displacement of the Pole
$\dot{\theta}(t)$	Pendulum Angular velocity
x^e	Equilibrium state
$\overline{\psi}_s$	Stator flux linkage space vector
$\bar{i}_s$	Stator current space vector
ρ_s	Angle of the stator flux-linkage space vector with respect to the direct-axis
α_s	Angle of the stator-current space vector with respect to the direct axis
t_e	Instantaneous electromagnetic torque
VC	Vapnik-Chervonenkis

LPP	Linear Programming Problem
P	Power rating / Plant
L_m	Magnetising Inductance
L_r	Rotor Inductance
L_{sc}	Stator Inductance
R_r	Rotor resistance
R_s	Stator resistance
V_{sd} / V_{sq}	Direct / Quadrature axis voltage
I_{sd} / I_{sq}	Direct / Quadrature axis current
$S_a / S_b / S_c$	Inverter switchings for the three phases, 'a'/'b'/'c'
r_i	Residual for the i-th input
e	Vector of 1's
c	Regularization factor
q	Slack variable
β	Lagrange multiplier/ parameter
u	$[w \quad w_0]^T$
f_{w,w_0}	Function parameterized by w and w_0
ρ_i	Measure of relevance
S_{ij}	Sensitivity measure for any weight w_{ij}
h_{ii}, h_{ij}	Components of the Hessian matrix
$\varepsilon_1, \varepsilon_2$	Weight decay parameters

Chapter 1

Introduction

This dissertation is an attempt to incrementally construct neural networks for classification and function approximation tasks and to investigate some control applications. Artificial Neural Networks (ANNs) [Her91, Zur97, Bos98, Hay98, Jay2K1, Kum2K4a] are modelled following the organizational principles of the central nervous system, with the hope that the biologically inspired computing capabilities of ANNs will allow cognitive tasks to be performed more easily and more satisfactorily than with conventional serial processors. The computational power of ANNs stem from their learning capabilities. Multilayer feed-forward ANNs have been successfully applied to function approximation and pattern classification. However, choosing an appropriate structure a priori is one of the core issues of neural network research, and choices vary with the intended application. A network that is too small may be unable to adequately learn a training set, while unduly large networks tend to overfit the data. In practice, the optimal number of neurons and a suitable architecture cannot be decided in advance. Choosing too few or too many neurons may lead to under fitting or over-fitting of the data. Therefore, there is a need for developing adaptive neural network architectures capable of good generalization. Another aspect of importance when developing a neural network is the choice of the training strategy that determines the final weights of the network to fulfill a given learning task.

Considerable effort has been spent on the development of modeling techniques, and for incrementally modifying an existing network model so that it can accomplish a given task. These techniques fall into three categories, viz. constructive methods, pruning techniques, and regularization.

1.1 Constructive Methods

Constructive techniques start with a small network or even a single node, and incrementally add units or layers in order to complete the learning task. Kwok et al. [Kwo97a] gives a survey of major constructive approaches. The Cascade-Correlation Learning Architecture [Fah89] adds new hidden units one by one to a small initial network, to create a multi-layer structure. Subsequently, the input weights of the added unit are frozen. They are now permanent feature detectors in the network, available for producing outputs, or for creating other, more complex feature detectors. The Upstart algorithm [Fre90] constructs a hierarchical tree network with linear threshold units to implement any Boolean mapping. The Pyramidal Delayed Perceptron proposed by Martinelli et al. [Mar90] follows an approach based on linear programming (LP) to yield a network of pyramidal shape. In the Feedforward Neural Network Construction Algorithm (FNNCA), Setiono et al. [Set95] starts with a single hidden layer consisting of two hidden units, and computes a local minimum associated with the network error function by employing a variant of the quasi-Newton method. The FNNCA minimizes a sequence of error functions associated with the growing

network. Prechelt [Pre97] discussed the problems of the Cascade-Correlation algorithm and how to overcome them by using different variants. Kwok et al. [Kwo97b] surveyed a number of objective functions for training new hidden units in constructive algorithms for multilayer feedforward networks. Ma et al. [Ma2K3] proposed a scheme for autonomously constructing a multi-hidden-layer feedforward neural network that adds both new hidden units and new hidden layers one at a time. In a nutshell, constructive algorithms aim at reducing model complexity and building a network of reasonable size.

1.2 Pruning Algorithms

Pruning techniques begin with a larger trained network and then remove redundant network connections and/or hidden units by following some performance criterion. Reed [Ree93] gives a good survey of some early pruning techniques.

Mozer and Smolensky [Moz88] proposed a method called "skeletonization" that uses a measure of relevance of the weights as the pruning criterion. They introduced a measure of the performance of the network when a unit i is removed by defining a measure of relevance $\rho_i = E_{\text{without unit } i} - E_{\text{with unit } i}$, where, E is the error of the network on the training set. They proposed calculating the relevance of a unit using a linear error function like

$$E' = \sum |t_{pj} - o_{pj}|$$

where, p is an index over the patterns, j over output units; t_{pj} is the target output, o_{pj} the actual output. The units that are most relevant to the performance of the network are retained while the least relevant ones are pruned to construct a skeleton version of the network.

Karnin [Kar90] devised a measure of the sensitivity of the error function to the exclusion/inclusion of each connection as the pruning criterion. Assuming an error function E, the sensitivity measure S_{ij} for any weight, w_{ij} was defined as

$$S_{ij} = -\frac{E(w_{ij}^f) - E(w_{ij}^i)}{w_{ij}^f - w_{ij}^i} w_{ij}^f,$$

where w_{ij}^f is the final value of the weight and w_{ij}^i is the initial value. Given a training set, on completion of the training procedure, a sensitivity value corresponding to each weight was computed. The weights having lower sensitivities were pruned.

Cun et al. [Cun90] proposed the Optimal Brain Damage (OBD) procedure that uses a 'saliency measure' as the pruning criterion. The approach uses the second derivative of the objective function with respect to the parameters, to compute the saliencies. According to this technique, the change in objective function, E due to a perturbation δU of the parameter vector, can be approximated as, $\delta E = \frac{1}{2}\sum_i h_{ii}\delta u_i^2$, where, δu_i's are the components of δU, and h_{ij}'s are the elements of the Hessian matrix H and are given by $h_{ij} = \frac{\partial^2 E}{\partial u_i \partial u_j}$. For

each parameter u_k, its second derivatives h_{kk} and the corresponding saliency, $s_k = h_{kk}\frac{u_k^2}{2}$ were computed. The parameters were sorted based on their saliency and some of the low-saliency parameters were pruned.

Hassibi et al. [Has92, Has93] proposed the Optimal Brain Surgeon (OBS) that is similar to the OBD, with some modification to the network's Hessian. While OBD assumes the Hessian to be diagonal, OBS makes no restrictive assumptions on the form of the network's Hessian and thereby proposes better learning of the network weights.

Castellano et al. [Cas97] proposed a new pruning algorithm based on iteratively eliminating units and adjusting the remaining weights such that the network performance does not deteriorate over the entire training set. If i be a unit in the h's projective field that is being removed, its net input upon presentation of pattern $\mu \in \{1,\ldots,M\}$ is given by

$$\sum_{j \in R_i} w_{ij} y_j^{(\mu)}$$

where, $y_j^{(\mu)}$ denotes the output of unit j corresponding to pattern μ and R_i is the receptive field of unit i. After removal of h, unit i will take its own input from $R_i - \{h\}$. In order to maintain the original network behavior, the remaining weights incoming into node i, i.e. the w_{ij}s for all $j \in R_i - \{h\}$, were adjusted such that the new net input is as close as possible to the old one for all training patterns. This requires that

$$\sum_{j \in R_i} w_{ij} y_j^{(\mu)} = \sum_{j \in R_i - \{h\}} (w_{ij} + \delta_{ij}) y_j^{(\mu)}$$

for all $\mu = 1, \ldots, M$ and $i \in P_h$, projective field of h, where δ_{ij}'s are the appropriate adjusting factors to be determined.

Setiono et al. [Set2K] proposed a neural network pruning for function approximation (N2PFA) which assumes that the networks have been trained with a larger number of hidden units. It consists of two stages. In the first stage, redundant hidden units are removed by comparing their errors on the training data set and the cross-validation data set. In the second stage, irrelevant input units are removed by comparing their errors on the training data set and cross-validation data set.

1.3 Regularization

The weight updation rule in conventional neural networks is based on the minimization of a network performance that is defined as the error between the desired output and the actual output generated by the network during the training phase. In regularization techniques, an additional term is added to the error term. This is referred to as the penalty term as it has originated from the theory of constrained optimization methods known as Sequential Unconstrained Minimization Techniques [Kam97]. Different types of penalty terms have been proposed in the literature.

Chauvin [Cha88] used an additional "energy" term along with the quadratic error term as the penalty function. The energy term was a function of the squared

activation of the hidden units. If o_{ij} denotes the activation of i-th hidden unit of a layer for the j-th pattern, the energy term was defined as,

$$\sum_{j=1}^{P}\sum_{i=1}^{h} e\left(o_{ij}^2\right),$$

where $e(.)$ is a positive monotonic function, P is the total number of training patterns, and h is the number of hidden units. To prevent the weights from growing too large, an additional weight decay term $\sum_{j=1}^{n}\sum_{i=1}^{h} w_{ij}^2$, is added to the objective function, where w_{ij} is the weight of the synapse from the j-th input onto the i-th hidden layer unit.

The N2P2F algorithm [Set97] uses two components in the penalty term: the first term discourages the use of redundant connections while the second term prevents the connection weights from taking excessively large values. If w_{ml} is the weight of the synapse from the l-th input onto the m-th hidden unit and v_{pm} is the weight of the synapse from the m-th hidden unit onto the p-th output, the penalty term is given by

$$\varepsilon_1 \sum_{m=1}^{h}\left(\sum_{l=1}^{n}\frac{\beta w_{ml}^2}{1+\beta w_{ml}^2}+\sum_{p=1}^{C}\frac{\beta v_{pm}^2}{1+\beta v_{pm}^2}\right)+\varepsilon_2\sum_{m=1}^{h}\left(\sum_{l=1}^{n} w_{ml}^2+\sum_{p=1}^{C} v_{pm}^2\right),$$

where β is a constant and the weight decay parameters $\varepsilon_1, \varepsilon_2 > 0$ were chosen to reflect the relative importance of the accuracy of the network versus its complexity. Regularization needs a priori choice of the network size, and one needs to choose appropriate values of the regularization parameters. This is often done by trial and error, or by using heuristics.

Thodberg [Tho96] used Bayesian methods in the network training to automatically select the regularization parameter. A weight decay parameter that depends on the number of inputs was used, leading to improved results.

Of the three types of adaptive structure neural networks, constructive algorithms offer some major advantages over pruning algorithms and regularization-based techniques. In pruning algorithms, nothing is known regarding the size of the initial network. However, in constructive learning techniques the initial network architecture can be specified a priori. Constructive algorithms tend to build smaller networks that depend on the complexity of the problem, whereas pruning algorithms may require large amounts of memory and computation time in order to prune unnecessary weights. Pruning and regularization based techniques are dependent on several problem specific parameters that affect the performance of the network; these are usually absent in constructive algorithms.

1.4 Neural Network Training

Apart from the choice of the network's architecture, another aspect of importance when developing a neural network is the choice of the training strategy. Conventional neural network learning methods employ iterative schemes for updating the weights. In a multi-layer feed-forward network, if w_{ij} connects the j-th output of a layer having $(n+1)$ neurons or the $(n+1)$-

dimensional input (n-dimensional input augmented by a bias term) to the input of i-th neuron of the next layer having r neurons, the updated value of w_{ij} on presentation of a training pattern is given by

$$w'_{ij} = w_{ij} + \Delta w_{ij} . \tag{1.1}$$

The incremental change in the weight, denoted by Δw_{ij}, is a function of the output error. For a specific pattern p $(p = 1, 2, \ldots, P)$ at the input, let t_{pi} and o_{pi} denote the target and actual outputs at the i-th neuron, respectively. The sum-squared-error (SSE) cost function over all output units is given by

$$E = \frac{1}{2}\sum_{i}\left|t_{pi} - o_{pi}\right|^2 . \tag{1.2}$$

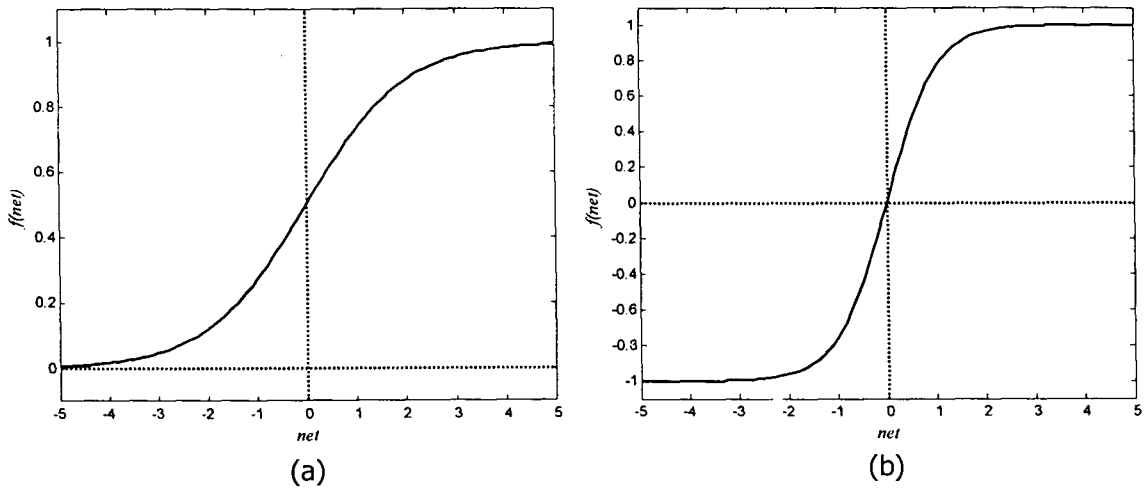
The individual weight adjustments that minimize SSE are given by

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \; i = 1,2,\ldots,r; \; j = 1,2,\ldots,(n+1) \tag{1.3}$$

where η denotes the learning rate. For the i-th neuron, if $f_i(.)$ and net_i denote its activation function and total input, respectively and if y_j denotes the output of the j-th neuron of the previous layer, Δw_{ij} can be shown to reduce to

$$\Delta w_{ij} = \eta\left(t_{pi} - o_{pi}\right) f_i'(net_i) y_j . \tag{1.4}$$

Some nonlinear activation functions and their derivatives, that are widely used in multi-layer feed-forward neural networks are given in Table 1.1, and have been plotted in Fig. 1.1.



(a) (b)

Fig. 1.1. Some Nonlinear Activation functions. a) "sigmoid" activation function; b) "tanh" activation function.

Table 1.1: Some Nonlinear Activation Functions

Name	Function	Derivative
Sigmoid	$f(net)=\frac{1}{1+e^{-net}}$	$f'(net)=\frac{e^{-net}}{\left[1+e^{-net}\right]^2}=f(net)\left[1-f(net)\right]$
Tanh	$f(net)=\frac{1-e^{-net}}{1+e^{-net}}$	$f'(net)=\frac{2e^{-net}}{\left[1+e^{-net}\right]^2}=\frac{1}{2}\left[1-\{f(net)\}^2\right]$

The most popular backpropagation algorithm [Her91, Zur97, Bos98, Hay98, Kum2K4a] computes the error (1.2) at the network's output and updates the weights layerwise. Convergence of algorithms such as 'backprop' and its variants depends on the initial values of the weights and the choice of various parameters of the algorithm. In general, classical derivative-based weight-updating algorithms are plagued by the need for a large number of training examples, occurrence of local minima, oscillations, and "forgetting" problems.

Although it works well for a wide range of problems, for the particular case of learning binary mappings, gradient descent based training of feed-forward neural networks using SSE as the cost function with nonlinear activation functions like the "sigmoid" and "tanh", often gets stuck with some outputs wrongly learnt [Bul2K3]. In such a case, the value of the incremental change in weight, Δw_{ij}, (1.4), tends to go to zero because the unipolar "sigmoid" (Fig.1.1a) or the bipolar "tanh" (Fig. 1.1b) saturates to 0/1 or –1/1, respectively. This is due to the component $f'(net)$ in Δw_{ij}, as is apparent from the expressions in Table 1.1. Thus, it is theoretically possible for the network to get stuck with some weights not getting updated, resulting in some totally wrong outputs being learnt. This is a very common phenomenon. Bullinaria [Bul2K3] has discussed this issue at length as well as its possible remedies, and has used evolutionary strategies to generate efficient neural network learning algorithms to learn binary mappings.

There exists another interesting set of neural network learning methods, that uses Linear Programming (LP) [Kam97] based formulations to determine network weights. Smith [Smi68] proposed a linear programming formulation for the discriminant function. Martinelli et al. [Mar90] proposed the Pyramidal Delayed perceptron-type neural network architecture for binary classification tasks, wherein network weights are obtained by solving a linear programming problem (LPP). Mangasarian [Man93] has shown that a neural network can be

trained for pattern classification using linear programming. In his approach, a succession of planes is used to divide the input space into polyhedral regions, each of which contains points of mostly one category. Bennett et al. [Ben94] showed that a linear programming based formulation is useful for discriminating between k-sets that are not linearly separable by using piecewise-linear surfaces. Roy et al. [Roy95] presented linear programming models to train RBF-like nets for solving classification problems. A new algorithm for function approximation using a combination of "truncated" RBF nets has been proposed by Roy et al. [Roy97a]. Roy's approach uses random clustering and linear programming to design and train the net. Another work of Roy et al. [Roy97b] uses linear programming models to construct and train a network consisting of higher-order perceptrons for classification problems. Lu et al. [Lu99] have shown that the problems of inverting Multilayer Perceptrons (MLP's) and RBF networks can be formulated as a separable programming problem, that can be solved by a modified Simplex method, which is a well-developed and efficient method for solving linear programming problems. The simplicity of the LP formulation and guaranteed convergence have increased the potential of LP based approaches for training neural networks.

1.5 Learning by Linear Programming and Support Vector Machines

Given a labelled binary data set, the task of pattern classification is to obtain a discriminant function w that is able to learn a part or the whole of the data set.

The shape of the discriminant function determines its learning capability. The simplest type of discriminant function is the linear discriminant one, that is computationally elegant and easy to implement. If there exists a linear discriminant function that can correctly learn the complete pattern set, the data set is termed as linearly separable, else it is called linearly inseparable. The linear discriminant function that separates a part or the whole of the data set can be obtained as a solution to a LPP. Hence to obtain a linear discriminant function, the primary task is to have a linear programming formulation of the pattern classification problem.

For a given binary data set, there may be several linear discriminant functions that can be obtained as solutions to a LPP based formulation. Figure 1.2 shows a two-dimensional case. Figure 1.2a shows several discriminant functions separating the training samples. Figure 1.2b shows the optimal hyperplane that is equidistant from both classes, with maximum margins for both classes. The larger margin may enable it to correctly classify unknown patterns not used in training. Here, the hyperplane is in such a position that its distances d^+ and d^- with respect to the closest patterns of both the classes are the same, and it maximizes the sum $\left(d^+ + d^-\right)$. In order to obtain the optimal hyperplane, only the training vectors that are on the margin are relevant. A linearly inseparable two-dimensional pattern set can be discriminated by a combination of several such piecewise-linear (PWL) segments. Using hyperplanes of

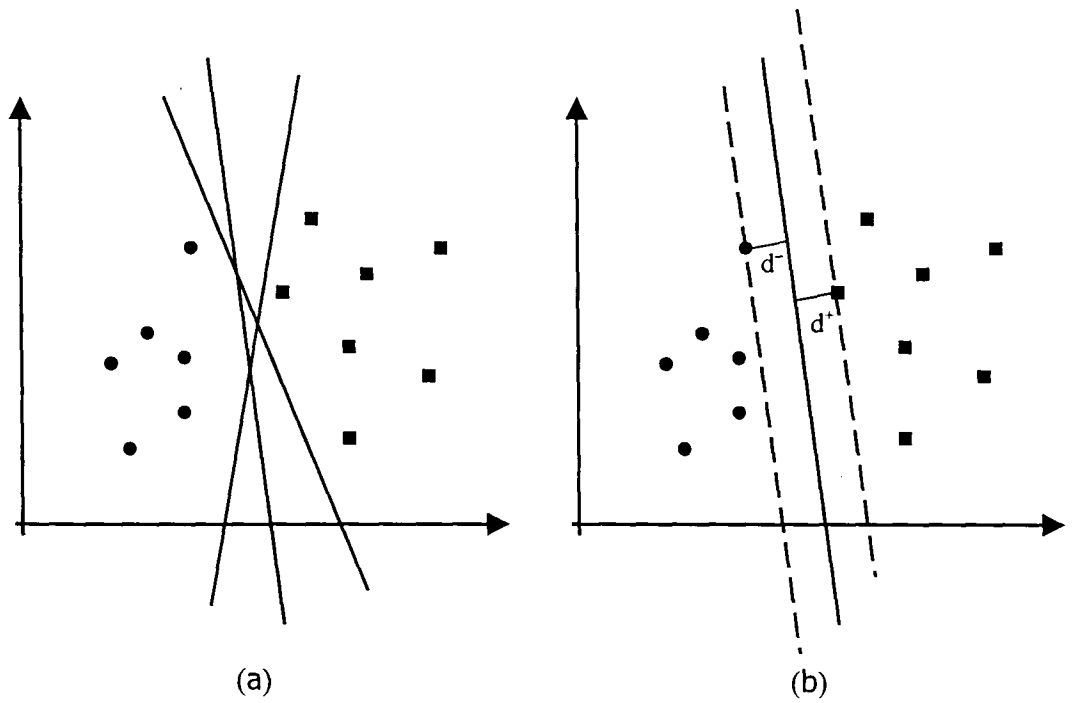


Fig. 1.2. Discriminating patterns of two classes. a) Existence of multiple discriminant functions for the linearly separable pattern set; b) Optimal discriminant function for the linearly separable pattern set.

appropriate dimension, the above notion can be extended to the multi-dimensional case, where the dimensionality of the input space is more than two.

In the sequel, Support Vector Machines (SVMs) that yield large margin classifiers have been used to obtain an optimal classifier. Nonlinear kernel based SVMs [Sch97, Vap98, Smo98a, Smo98b, Ama99, Vap2K, Cri2K, Sch2K2] have emerged as a new machine learning paradigm with applications in many areas such as pattern recognition, regression estimation etc. SVM solutions are theoretically elegant, computationally efficient, and seen to be very effective for large-scale problems.

Support Vector Machines grew out of ideas from Statistical Learning Theory [Vap98, Vap2K, Cri2K]. Here, the hypothesis that learns the training data is obtained by not just minimizing the error on the training data (empirical error). Large margin classifiers also minimize the structural risk, that expresses an upper bound on the generalization error, i.e. the probability of erroneous classification on unknown samples. If f denotes the classifier discriminant function chosen from any arbitrary set of functions S, Ω denotes a monotonic function of complexity parameter h, N denotes the size of the training set, and $R_{emp}(f)$ denotes the empirical error on the training set, then the actual risk $R(f)$ is bounded from above as

$$R(f) \leq R_{emp}(f) + \Omega\left(\frac{h}{N}\right). \tag{1.5}$$

The second term on the right-hand-side of (1.5) is the structural risk and is determined by the complexity or size of the set of classifier functions, S. The complexity can be described by the single parameter h known as the Vapnik-Chervonenkis (VC) dimension.

By using kernel functions, SVMs can be used to find discriminant functions for nonlinearly separable data. However, problems persist in the a priori choice of the kernel function and its various parameters. It is an open research area that needs to be explored. The idea of linear SVM classifiers has now been

extended to linearly inseparable problems and regression tasks [Vap98, Cri2K, Smo98b] as well. In a nutshell, the attributes that make SVMs attractive are:

i) They generalize well from relatively few data points.

ii) They provide a unique separating surface that maximizes the margin of separation.

iii) Using an adequately chosen kernel, they provide a way to obtain nonlinear classification boundaries by projecting data from the input space to a higher dimensional feature space.

iv) They provide model selection using conventional mathematical programming techniques, and are solvable by readily available software.

In this thesis, we have explored a number of methods that use linear programming (LP) as part of the training procedure. The problem of learning from labelled data is formulated as a LPP, and the optimal linear hyperplane is sought for better generalization; SVMs have been used for this purpose. The use of linear programming makes the problem formulation simpler, and the hyperplane based decision regions are easy to represent and visualize. For real-time solutions to the machine learning problem, efficient algorithms for linear programming [Ign94, Wis71, Las70, Bra2K] exist and the implementations can be parallelized [Cha97, Gol89, Alo90, Sri93].

Tree type constructive neural networks have been built by using linear hyperplanes at each node. No a priori assumptions need to be made regarding the number of neurons in the network or the kernel of the SVM classifier; the proposed approach realizes a piecewise-linear decision region. It was attempted to overcome the problems associated with derivative based methods for training multi-layer networks. The proposed tree type neural networks have been used in classification and control applications.

In machine learning applications, constructive networks have been used to learn synthetic data sets, and generalization performance on real benchmark data sets has been compared with a few other reported network construction algorithms. In control applications, tree type networks have been used as the critic in the action dependent heuristic dynamic programming (ADHDP) training strategy for the failure avoidance control of the pole-balancing problem. They have also been used for generating the switching signals for the speed control of a 3-phase induction motor. For real-time control applications, simpler LP formulations and parallelization of efficient LP algorithms offer the scope of faster learning [Sri93, Lus96, Kla2K].

In another part of this work, a technique to build growth networks to approximate functions in one dimension using linear hyperplanes has been

proposed. A new fast approach to function approximation that involves a single matrix inversion, has also been developed in this thesis.

The remainder of this thesis is organized as follows. In Chapter 2, a SVM based tree type neural network has been proposed for performing binary classification. This approach makes no a priori assumptions regarding the number of neurons in the network or the kernel of the SVM classifier, and effectively realizes a piecewise-linear decision region. An improved learning algorithm for binary classification has been proposed in Chapter 3, and its performance has been evaluated on benchmark data sets. In Chapter 4, we propose a technique for one-dimensional function approximation. The proposed approach constructs a multi-layer tree-structured neural network, in which the neurons have a piece-wise linear (PWL) activation function. Another approach to function approximation that involves a simple matrix inversion has been proposed and results compared with other algorithms. In Chapter 5, we discuss two applications of the binary classification network to control systems. In the first application, the network has been used as the critic in adaptive critic designs for the pole balancing problem, the objective being to train a controller that minimizes the occurrence of failures and provides smooth operation. The second application proposes the use of the tree type network to learn the inverter switchings for the speed control of a three-phase induction motor. Chapter 6 contains concluding remarks.

Chapter 2

Binary Classification using SVM based Tree type Neural Networks

2.1 Introduction

The problem of approximating an unknown function $f:\Re^n \to \Re$ from a set $S_t = \{(x,y)|x \in \Re^n, y \in \Re\}$ of input-output pairs can be considered as a pattern classification problem if the range of the function's output y is discretized to M ($M \geq 2$) values, and the output for each input sample is restricted to one of them, i.e. it involves a real to M-ary function mapping. A special case of the M-ary function mapping is known as the binary classification problem ($M = 2$), where the output is binary-valued. For a binary classification problem, there ideally exists a hypersurface of adequate complexity in the input space that completely separates samples of both classes.

Multi-layer networks are able to construct arbitrarily complex decision regions to accomplish a classification task or to approximate functions [Hor89], but there are no clear guidelines for choosing the number of neurons or the best network architecture. A network that is too small is unable to adequately learn a problem, while unduly large networks tend to overfit the data. Constructive methods, pruning techniques, and regularization schemes are different approaches to finding a network of the right size for a given task.

The Upstart algorithm [Fre90] constructs a hierarchical tree network with linear threshold units to implement any Boolean mapping using a simple recursive rule to build the net's structure by adding units as they are needed. A modified perceptron algorithm is used to learn the connection strengths. The "Pyramidal Delayed Perceptron" (PDP) proposed by Martinelli et al. [Mar90] uses a linear programming based approach. The PDP algorithm yields a feedforward network of pyramidal shape and determines the weights of the connections among neurons. "The Oil-Spot Algorithm" proposed by Mascioli et al. [Mas95] dynamically constructs a two layer neural network by involving successive binary samples. Muselli [Mus95] discusses a new technique called sequential window learning (SWL), that constructs two-layer perceptrons with binary inputs. Frank et al. [Fra98] used model trees, which are a type of decision tree with linear regression functions at the leaves. The "CARVE" algorithm proposed by Young et al. [You98] constructs a feedforward network with a single hidden layer of threshold units that can implement arbitrary classification tasks. Treadgold et al. [Tre99] examined the effect of regularization on generalization in constructive cascade algorithms. Parekh et al. [Par2K] presented two constructive learning algorithms, *Mpyramid-real* and *Mtiling-real*, as an extension to the *pyramid* and *tiling* algorithms for learning real to *M*-ary mappings. Neural Network Construction Using Cross Validation (N2C2S) proposed by Setiono [Set2K1] constructs feedforward neural network classifiers with a single hidden layer, and is trained using a conjugate gradient minimization technique. For a given

problem, most learning algorithms for multi-layer networks generally build networks either by adding more layers of processing units, or by adding more neurons with a fixed number of hidden layers. It is also desirable to have good generalization performance of the network. A procedure for improving the generalization in classification trees has been proposed in [Man2K2]. Here, the decision borders are placed in an optimal position inside the problem space, thereby unlearning noisy training samples and improving generalization. The widely referenced book by Duda et al. [Dud2K3] gives a very systematic account of major topics in pattern classification, and discusses a wealth of different techniques that can be applied to them.

Apart from the choice of neural network architecture, another aspect of importance is the choice of the training strategy. Section 1.4 discusses some of the problems that accompany gradient-based training of multi-layer neural networks with particular reference to the case of learning binary mappings. It also discusses alternative techniques of neural network training that use linear programming formulations of the learning problem to determine network weights.

Using a simple linear programming formulation of the binary classification task, the proposed approach develops a growth network by adding neurons one at a time. Each neuron in the tree represents a maximum-margin classifier that separates points belonging to the two classes. It obviates the need for an a priori

choice of the number of neurons, number of hidden layers, or the kernel function in a SVM. The final decision surfaces are piecewise-linear. The existence of efficient codes for linear programming facilitates high speed implementation.

The remainder of this chapter is organized as follows. Section 2.2 discusses the general binary classification problem. Section 2.3 describes how the binary classification problem can be formulated in a linear programming framework. It also elaborates on the proposed SVM based tree type neural network. Section 2.4 contains experimental results on the applications of the SVM based tree type neural network. Section 2.5 is devoted to concluding remarks.

2.2 Binary Classification Problem

Given a labelled data set $S_t = \{(x^i, y^i), i = 1,2,...,N; x^i \in \Re^n; y^i \in \{0,1\}\}$, the task is to obtain a set of weights $w \in \Re^n$, and bias $w_0 \in \Re$ such that

$$\sum_{d=0}^{n} w_d x_d^{(i)} \begin{cases} \geq \Delta, & \text{if } y^i = 1 \\ \leq -\Delta, & \text{if } y^i = 0 \end{cases}; \; x_0^{(i)} = 1; \; 1 \leq i \leq \text{card}(S_t) \tag{2.1}$$

where i denotes the i-th sample; $\text{card}(S_t) = N$ = cardinality of the data set, and Δ is a positive quantity that has been introduced for better separation of the data. In some cases, such as the SVM classifier formulation [Vap98, Vap2K, Cri2K], the output set is considered as $\{-1, 1\}$ instead of $\{0, 1\}$ in order to ensure

non-trivial value of any product. It is always possible to retrieve the $\{0, 1\}$ output values from the actual output by using a suitable logic function.

In case of a linearly separable 'training set', for a given Δ, a hyperplane $w^T x + w_0 = 0$ can be determined to satisfy the above inequalities. Without loss of generality we may choose Δ=1. Clearly if Δ≠1, both sides of the entire inequality can be divided by the value of Δ to obtain a new inequality with Δ=1. Once the hyperplane is found, the class of a new data point x is determined by taking the sign of $\sum_{d=0}^{n} w_d x_d$; x_0 is chosen to be equal to 1 and Δ does not picture in the testing phase. In case of linearly inseparable data, following the approach of the proposed growth network, a hyperplane can always be found that learns the training set.

For the above problem, the capability of tree structured neural networks for binary classification has been demonstrated in [Jay2K2b], where each neuron in the network is a perceptron implementing a maximum margin classifier for the data linearly separable by it. It may be noted that (2.1) has many possible solutions. An ideal choice is a hyperplane that is placed at an optimal distance from the correctly learnt samples of the two classes. At the same time, it must maximize their number. This is illustrated by a simple example shown in Fig. 2.1. This is a binary classification problem having two parallel rows of patterns of each class, and is clearly not linearly separable. We are interested in the hyperplane that makes a minimum number of errors in learning. As evident from

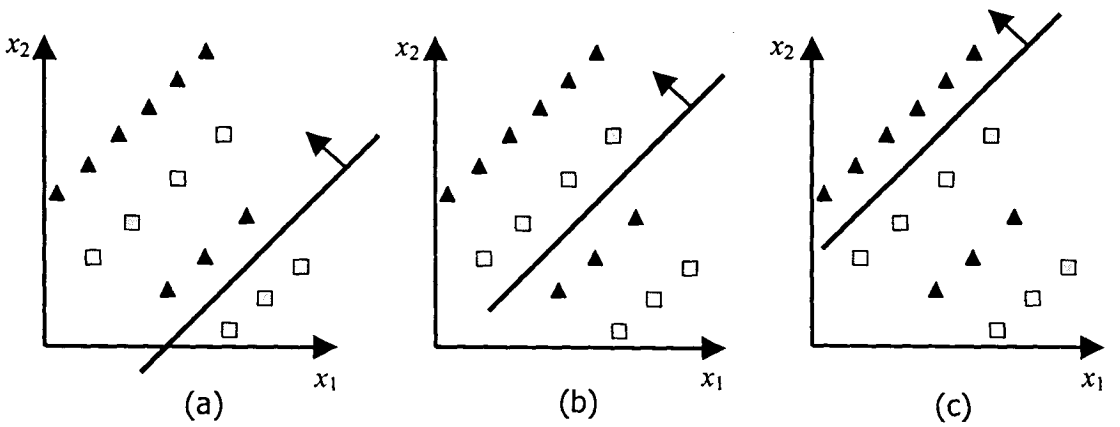


Fig. 2.1. A Binary Classification Problem and possible hyperplanes that provide a solution. a) a sub-optimal solution with 4 error points; b) a sub-optimal solution with 7 error points; c) optimal solution with 3 error points.

Fig. 2.1, the hyperplanes in Figs. 2.1a and 2.1b give sub-optimal solutions, with 4 and 7 error points respectively, while the hyperplane in Fig. 2.1c gives the optimal solution with 3 error points.

Thus, with the objective of learning binary mappings, a constructive growth network made up of perceptrons may be envisaged, where each perceptron tries to correctly learn as many patterns from its training set as possible, and the final perceptron discriminator is placed optimally with respect to the correctly learnt patterns using SVMs. Corrections on account of any incorrectly learnt pattern are achieved by adding child neurons. Section 2.3 describes the network growth and illustrates the approach through an example.

2.3 SVM based Tree type Neural Networks

2.3.1 Threshold Logic Activation function

The activation function of the neurons in the proposed network is of the threshold logic type. Its transfer characteristic is given by (2.2), and is sketched in Fig. 2.2.

$$\text{threshold}(net) = \begin{cases} 1, & \text{if } net > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

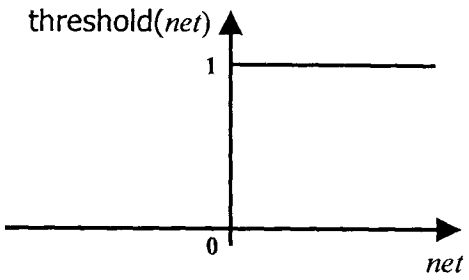


Fig. 2.2. Threshold Logic Activation function

2.3.2 Building the Neural Network

Given a input-output training data set

$$S_t = \{(x^i, y^i), i = 1,2,...,N; x^i \in \Re^n; y^i \in \{0,1\}\},$$

I_1: index set of data samples of Class 1 having output $y^i = 1$; card $(I_1) = m$,

I_0: index set of data samples of Class 0 having output $y^i = 0$; card $(I_0) = k$,

$I = I_1 \cup I_0$; card$(I) = N = m + k$; $I_1 \cap I_0 = \phi$, the objective is to build a network by adding perceptrons one at a time. A perceptron correctly learns its whole training set if it satisfies the inequalities in (2.1) for all the input patterns.

In (2.1), w_d can be of either sign, and since the Simplex method requires all variables to be non-negative, following Smith [Smi68], each w_d can be expressed in terms of non-negative variables $w_d^{(+)}$ and $w_d^{(-)}$ as

$$w_d = w_d^{(+)} - w_d^{(-)}. \tag{2.3}$$

Considering $x_0^i = 1$, introducing additional non-negative variables $s_i^{(+)}, s_i^{(-)}, (i = 1,2,\ldots,N)$ and following Martinelli et al. [Mar90], (2.1) gets reduced to the following equalities:

$$\sum_{d=0}^{n}\left(w_d^{(+)} - w_d^{(-)}\right)x_d^i - \left(s_i^{(+)} - s_i^{(-)}\right) = \Delta, \text{ if } y^i = 1; i \in I_1 \tag{2.4a}$$

$$\sum_{d=0}^{n}\left(w_d^{(+)} - w_d^{(-)}\right)x_d^i + \left(s_i^{(+)} - s_i^{(-)}\right) = -\Delta, \text{ if } y^i = 0;\ i \in I_0. \tag{2.4b}$$

The i-th constraint in (2.1) is satisfied if, in the corresponding equality in (2.4),

$$s_i^{(-)} = 0, \text{ for } i \in I. \tag{2.5}$$

Given a set of input-output data pairs, the obvious objective is to minimize the number of inequalities that are not satisfied. This can be formulated as the following linear programming problem (LPP).

$$\text{(LPP)} \qquad \text{Minimize} \quad \left(\sum_{i=1}^{m} s_i^{(-)} + \sum_{i=1}^{k} s_i^{(-)}\right) \tag{2.6}$$

subject to the following constraints

$$\sum_{d=0}^{n}\left(w_d^{(+)} - w_d^{(-)}\right)x_d^{(i)} = \Delta + \left(s_i^{(+)} - s_i^{(-)}\right), \text{ if } y^i = 1; i \in I_1 \tag{2.7a}$$

$$\sum_{d=0}^{n}\left(w_d^{(+)}-w_d^{(-)}\right)x_d^{(i)}=-\Delta-\left(s_i^{(+)}-s_i^{(-)}\right),\ \text{if}\ y^i=0; i\in I_0 \tag{2.7b}$$

where $\text{card}(I_1)=m;\ \text{card}(I_0)=k;$ and $x_0^i=1$.

The objective function used in (2.6) is similar to the one used in Phase I of the two-phase Simplex method (Kam97). Bennett et al. [Ben92] used an objective function that assigns relative weights to the slack variables as shown in (2.8) along with constraints (2.7a) and (2.7b).

$$\text{Minimize}\ \sum_{i=1}^{m}\frac{1}{m}s_i^{(-)}+\sum_{i=1}^{k}\frac{1}{k}s_i^{(-)} \tag{2.8}$$

The objective function in (2.6) is a greedy choice that assigns equal weightage to all the slack variables. The use of (2.8) has an advantage over (2.6) as it avoids the trivial solution, $w=\mathbf{0},\ w_0=0$, and ensures convergence [Ben92]. The objective functions in (2.6) and (2.8) have been used in [Jay2K2b] for finding the initial hyperplane to dichotomize the training set while learning a neuron of the tree-structured neural network.

LPPs can be solved to obtain a separating hyperplane that tries to classify as many samples as possible. Following [Mar90], the N samples can be divided into the four classes of Table 2.1, on the basis of the neuron's output. Classes C_1, C_2, C_3, and C_4 contain N_1, N_2, N_3, and N_4 samples, respectively. Note that $N_1 + N_2 + N_3 + N_4 = N$ = the total number of training samples.

Table 2.1: Classification of Samples based on Output

Class	Actual Output	Desired Output	No. of Samples
C_1	0	0	N_1
C_2	1	1	N_2
C_3	0	1	N_3
C_4	1	0	N_4

Depending on the mismatch between the actual output and the desired output for the samples presented to the parent neuron, two neurons A and B are added to the parent as shown in Fig. 2.3, to take care of samples belonging to classes C_3 and C_4, respectively. Neuron A contributes a positive input for samples belonging to class C_3, thereby making the total input to the parent neuron positive. Neuron B contributes a negative input for samples belonging to C_4, thereby making the total input to the parent neuron negative for those samples. The desired outputs for the two child neurons are given in Table 2.2.

Table 2.2: Required Output of Neuron A and Neuron B

Class	Output of A	Output of B
C_1	0	Don't care
C_2	Don't care	0
C_3	1	0
C_4	0	1

Let the outputs of neurons A and B for the i-th sample be denoted by x_a^i and x_b^i respectively. Then, the constraints that need to be satisfied are given by

$$\sum_{d=0}^{n} w_d x_d^{(i)} + w_a x_a^i + w_b x_b^i \geq \Delta; \text{ if } y^i = 1; i \in I_1, \text{ and} \tag{2.9a}$$

$$\sum_{d=0}^{n} w_d x_d^{(i)} + w_a x_a^i + w_b x_b^i \leq -\Delta; \text{if } y^i = 0; i \in I_0 \tag{2.9b}$$

where $\text{card}(I_1) = m; \text{card}(I_0) = k; x_0^i = 1$.

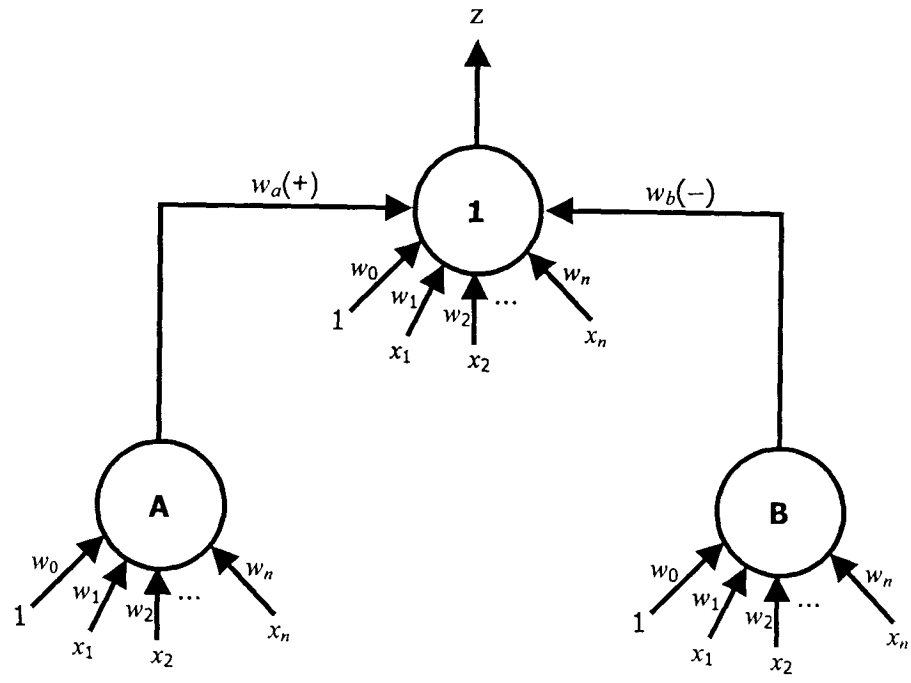


Fig. 2.3. The Parent Neuron and its two Child neurons, A and B

This augmented data set $\left\{\left(x_0^i\ x_1^i \ldots x_d^i\ x_a^i\ x_b^i\right)^T; i = 1,2,\ldots,N\right\}$ is always linearly separable, and is classified by using a maximum margin classifier type SVM, that is known to maximize the generalization ability. This is done by considering the parent neuron with the desired inputs from its child nodes. Its solution yields the input weights $w \in \Re^n$, and bias $w_0 \in \Re$ of the parent neuron, as well as weights of the interconnections from neuron A and B, viz. w_a and w_b. A network branch continues to grow until a terminal neuron has learnt its whole training set without any classification error. We now illustrate this approach through an example.

Example 1: A synthetic two-dimensional data set is considered in which samples belonging to Class 1 are located at points with the co-ordinates (1,1), (1,8), (4,5), (4,4), (6,5), (6,4), (10,8), and (10,1), while samples belonging to Class 0 are located at points with the co-ordinates (2,6), (2,3), (8,6), and (8,3). Δ=1 was chosen as the scalar factor for better separation. Figure 2.4(a) shows the initial hyperplane that is obtained, which is given by

$$x_1 - 3 = 0. \tag{2.10}$$

The hyperplane (2.10) classifies the samples at the parent neuron (neuron 1), following Table 2.1. The classifications and the desired correcting inputs from child neurons are given in Table 2.3.

Table 2.3: Classification Table for Neuron No. 1

Class	Samples	Desired Output of A	Desired Output of B
C_1	(2,3), (2,6)	0	Don't care
C_2	(4,4),(4,5),(6,4),(6,5),(10,1),(10,8)	Don't care	0
C_3	(1,1), (1,8)	1	0
C_4	(8,3), (8,6)	0	1

Due to the presence of some of the samples in class C_3, a neuron of type A (neuron no 2) needs to be added to the parent neuron so as to increase the net input for these samples at the parent neuron. Samples of class C_4 necessitate the addition of a neuron of type B (neuron no 3) to the parent neuron so as to reduce the net input for these samples. From Table 2.3, the training set of the neuron of type A (neuron no 2) consists of samples (2,3), (2,6), (8,3), (8,6) in Class 0 and (1,1), (1,8) in Class 1. Similarly, training set of the neuron of type B (neuron 3) consists of samples (4,4), (4,5), (6,4), (6,5), (10,1), (10,8), (1,1),

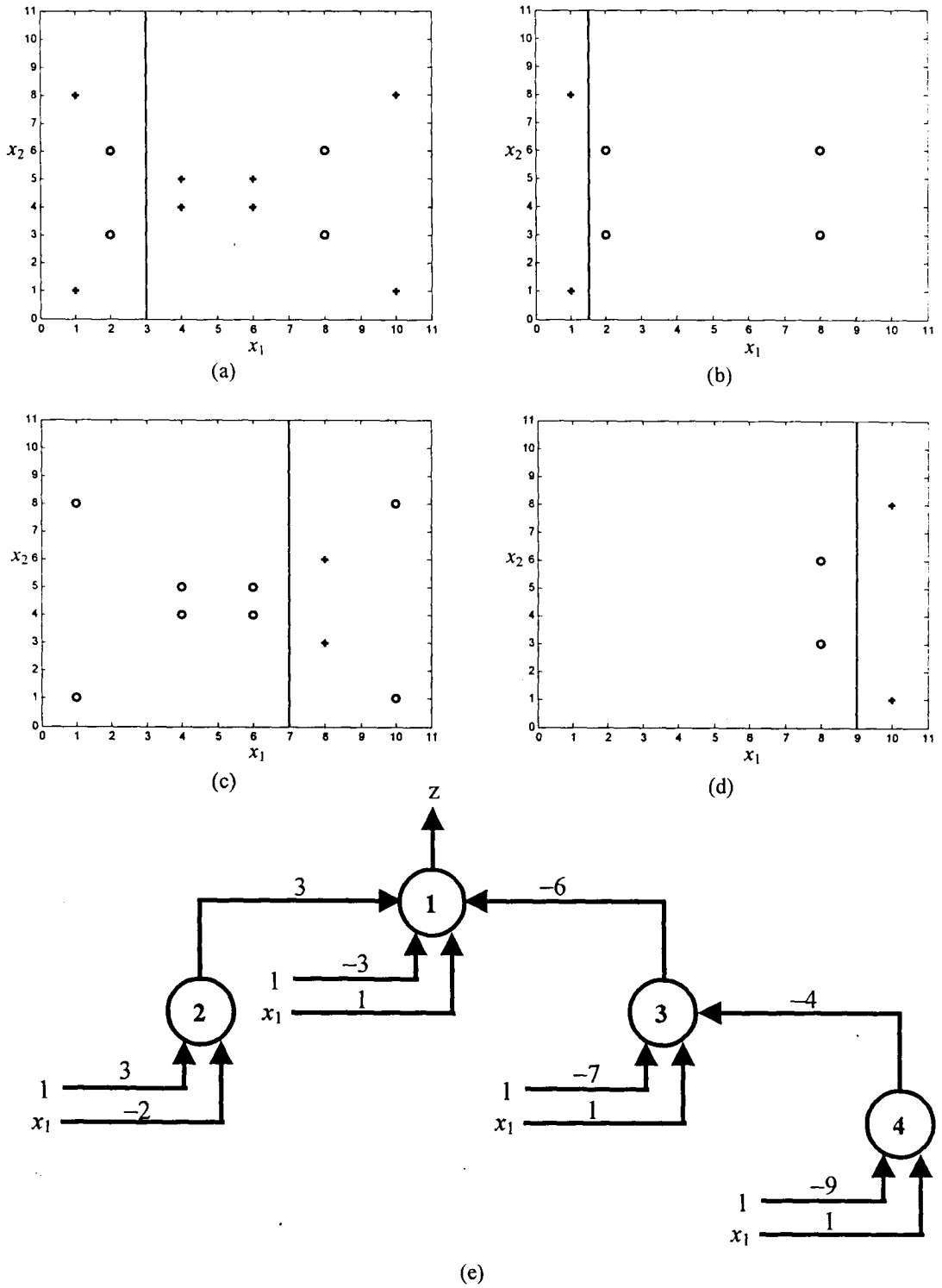


Fig. 2.4. Hyperplane Classifiers: (a) Neuron 1; (b) Neuron 2; (c) Neuron 3; (d) Neuron 4; (e) Entire classifier network.

(1,8) in Class 0 and (8,3), (8,6) in Class 1. Considering Δ=1, using (2.9) the solution of the maximum-margin classifier yields $w_a = 3$, and $w_b = -6$.

At neuron 2, the training set consists of samples {(2,3), (2,6), (8,3), (8,6)} in Class 0, and {(1,1), (1,8)} in Class 1. The classifying hyperplane, as shown in Fig. 2.4(b) is obtained as

$$2x_1 - 3 = 0. \tag{2.11}$$

This hyperplane correctly classifies all samples of the data set of neuron 2 as shown in Table 2.4. Hence no further growth takes place at this branch of the tree.

Table 2.4: Classification Table for Neuron No. 2

Class	Samples	Desired Output of A (not needed)	Desired Output of B (not needed)
C_1	(2,3), (2,6), (8,3), (8,6)		
C_2	(1,1), (1,8)		
C_3	{}		
C_4	{}		

For neuron 3, the training set consists of patterns {(4,4), (4,5), (6,4), (6,5), (10,1), (10,8), (1,1), (1,8)} in Class 0, and patterns {(8,3), (8,6)} in Class 1. The classifying hyperplane, which is shown in Fig. 2.4(c), is obtained as

$$x_1 - 7 = 0. \tag{2.12}$$

The newer outputs and desired correcting inputs from child neurons (if any) are shown in Table 2.5.

Table 2.5: Classification Table for Neuron No. 3

Class	Samples	Desired Output of A (not needed)	Desired Output of B
C_1	(4,4),(4,5),(6,4),(6,5),(1,1),(1,8)		Don't care
C_2	(8,3), (8,6)		0
C_3	{}		
C_4	(10,1), (10,8)		1

The presence of samples in Class C_4 necessitates the addition of a child neuron of type B (neuron 4) to neuron 3. The training set of neuron 4 consists of samples {(8,3), (8,6)} of Class 0, and {(10,1), (10,8)} of Class 1. Considering $\Delta=1$, and using (2.8), the solution of the maximum-margin classifier yields $w_b = -4$.

At neuron 4, the training set consists of samples {(8,3), (8,6)} in Class 0, and patterns {(10,1), (10,8)} in Class 1. The decision boundary as shown in Fig.2.4(d), is obtained as

$$x_1 - 9 = 0. \tag{2.13}$$

This hyperplane correctly classifies all samples of the data set of neuron 4 as shown in Table 2.6. Hence no further growth takes place of this branch of the tree.

Table 2.6: Classification Table for Neuron No. 4

Class	Samples	Desired Output of A (not needed)	Desired Output of B (not needed)
C_1	(8,3), (8,6)		
C_2	(10,1), (10,8)		
C_3	{}		
C_4	{}		

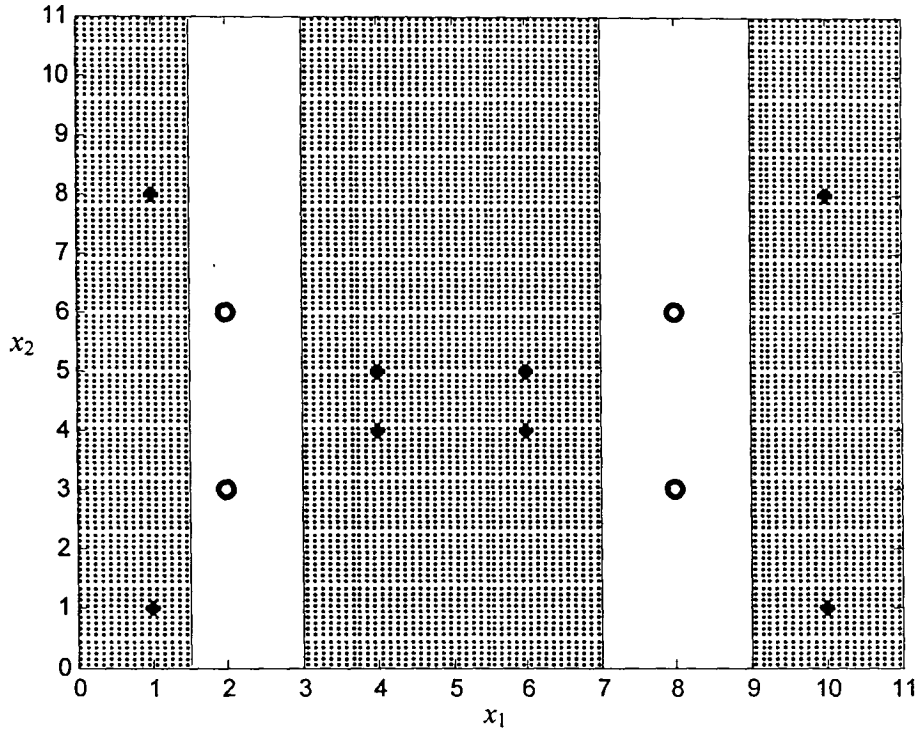


Fig. 2.5. Classification contours generated by the entire network.

Thus, the growth of the binary classification network is complete, and the entire network is shown in Fig. 2.4(e). The final classification contours generated by the entire network around the sample points are shown in Fig. 2.5.

Note: In Fig 2.4c, o and + signs are interchanged with respect to the other figures. This can be explained by combining Table 2.1 and Table 2.2 as below:

Table 2.7: Combination of Table 2.1 and 2.2

Class	Actual Output of parent	Desired Output of parent	Number of Samples	Desired Output of Type A neuron	Desired output of Type B neuron
C_1	0	0	N_1	0	Don't care
C_2	1	1	N_2	Don't care	0
C_3	0	1	N_3	1	0
C_4	1	0	N_4	0	1

It can be observed from Table 2.7 that for a child neuron of Type A, the desired binary outputs for training samples are identical to those of its parent while for a child neuron of Type B, the desired binary outputs for training samples are complementary to those of its parent. In the illustrative example, this can be noted for in neurons 1 and 3 in Figs 2.4a and 2.4c and neurons 3 and 4 in Figs.2.4c and 2.4d.

Steps for training the SVM based tree type neural network:

The overall algorithm may be summarized as follows:

Input: Training set, $S_t = \{(x^i, y^i), i = 1,2,...,N; x^i \in \Re^n; y^i \in \{0,1\}\}$

Δ = Scalar factor chosen for better separation.

The training set at the parent neuron decides the training sets of the child neurons. In case of benchmark data sets, the training sets are obtained by using a ten-fold cross validation methodology.

Step 1: Use the input attributes to formulate the equalities as in (2.7a), and (2.7b) in terms of the different components of w, and bias w_0 and suitable slack and surplus variables, all unrestricted in sign.

Step 2: Choose a suitable objective function in terms of the slack variables of the equalities. The objective function (2.6) was used to start with but that sometimes yielded a trivial solution. Therefore the objective function given by (2.8) is preferred. However, in the case of unbalanced data sets, where the number of samples of one class is much larger than the other, both (2.6) and

(2.8) were found to yield trivial solutions. In order to take care of such problems, two new objective functions are proposed in Chapter 3.

Step 3: Apply the Simplex algorithm until no further exchange of variables is possible. Obtain the connection weights from the final simplex table.

Step 4: Using these weights, compute the actual output of the perceptron for all the samples.

Step 5: Classify the samples into categories C_1, C_2, C_3, and C_4 as described in Table 2.1.

Step 6: If there are no samples in classes C_3 and C_4, go to step 9.

Step 7:

a) If any sample is in class C_3, a type-A neuron needs to be added. Develop a training set for neuron A by considering the desired output for samples from different classes according to Table 2.2.

b) If any sample is in class C_4, a type-B neuron needs to be added. Develop a training set for neuron B by considering the desired output for samples from different classes according to Table 2.2.

Step 8: If either neuron A or neuron B or both need to be added, augment the input dimension of the vertex neuron by considering the desired outputs of neuron A or neuron B or both, as obtained in step 7, as inputs to the parent neuron.

Step 9: Find the maximum margin classifier [Vap98, Vap2K, Cri2K] for the augmented input-desired output set if vertex neuron has child neurons A or B or

both (2.9), or for the original input-output relation (2.1) if the vertex neuron has no child.

Step 10: If there is no sample in classes C_3 or C_4, the solution yields the weight vector of the vertex neuron. Otherwise, the solution yields the input weight vector of the vertex neuron as well as the weights of the interconnections from neurons A and B.

Step 11: Repeat Steps 1 to 10 for type-A and type-B neurons.

END

2.4 Experimental Results

The SVM based tree type neural network was developed by using the MATLAB software (version 6.5 (R13)) [MAT], running on a 800 MHz Pentium III PC. The initial hyperplane obtained by solving the LPP was implemented by using the modified Simplex algorithm [Mar90], and the maximum margin classifier for obtaining the final classification boundary was implemented by using the .m-files from the Support Vector Machine Toolbox [Gun98] in the MATLAB 6.5 environment. The network was tested on artificial two-dimensional data sets as well as real benchmark data sets, and was also used for learning binary sequences, the results of which are summarized in Sections 2.4.1 to 2.4.3.

2.4.1 An Artificial Classification Problem

The following synthetic two-dimensional data set was considered. Samples belonging to Class 1 are located at {(3,3), (3,7), (5,5), (7,3), (7,7)} while samples belonging to Class 0 are located at {(1,1), (1,5), (1,10), (9,1), (9,5), (9,10)}. Figure 2.6 shows the SVM based tree type neural network having 6 nodes that has been generated by the algorithm. Figure 2.7 shows the points in two dimensions and the optimal decision boundaries composed of piecewise linear segments learnt by the network. Figure 2.8 shows the decision boundary obtained by a support vector classifier using a RBF kernel with $\sigma^2 = 1$, which was computed by using Schwaighofer's toolbox [Sch2K2]. This toolbox uses decomposition methods and working set selection strategies like those in SVM_*light* [Joa99], and can handle large data sets. Note that the Gaussian kernel yields a non-smooth decision boundary that is much more convoluted than the one found by our SVM based tree type classifier. From the theoretical perspective, Dong et al. [Don2K3] proposed a theoretical measure L of the classifiability of a classification problem and established an empirical relationship between the classifiability measure and boundary complexity S by the following theorem:

Theorem [Don2K3] *The classifiability measure is strongly and inversely related to the boundary complexity.*

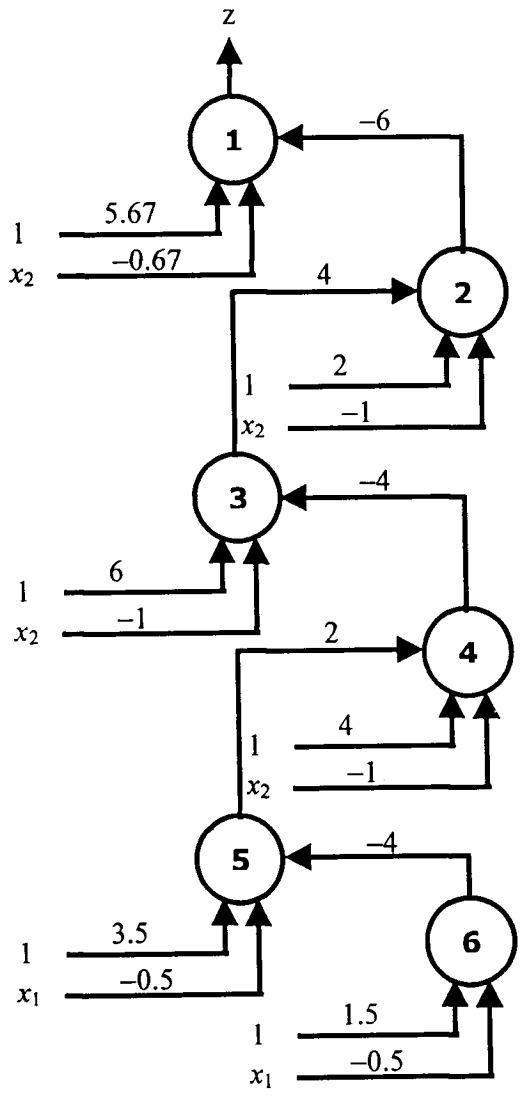


Fig. 2.6. SVM based tree type neural network for the Artificial Data set.

Considering boundary length as the boundary complexity parameter, the following empirical relationship was derived,

$$L + kS = 1 \tag{2.14}$$

where, k is a constant for a given data distribution. The above relationship shows that the classifiability measure relates inversely to the boundary complexity.

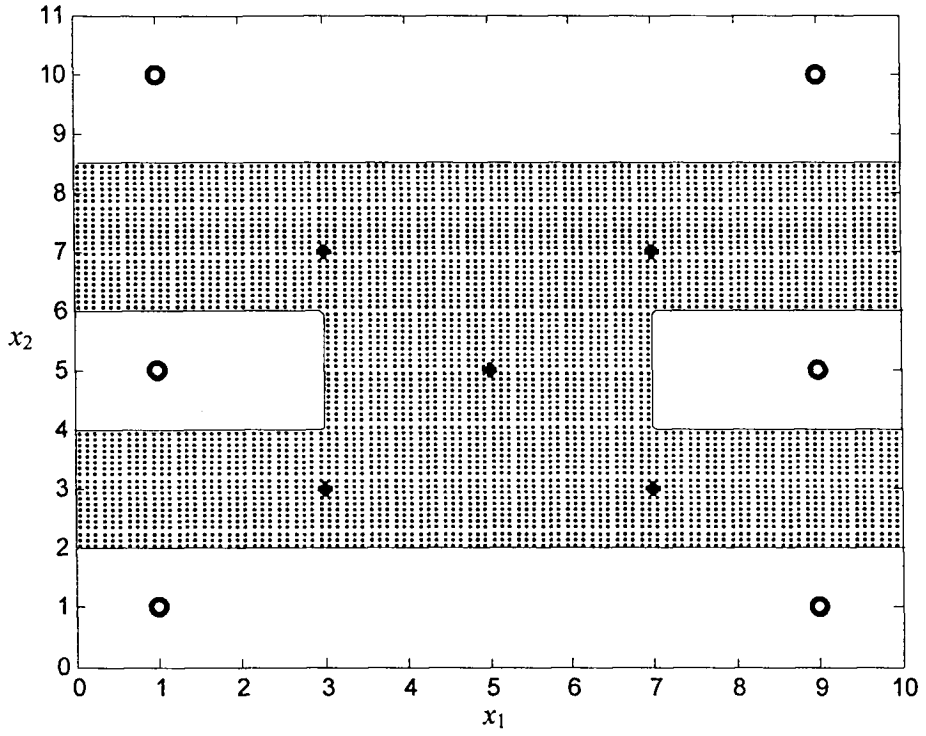


Fig. 2.7. Classification contours generated by the SVM based tree type neural network.

The relationship of the classifiability measure with the boundary complexity have been illustrated on four two-dimensional data sets [Don2K3] with sinusoidal decision boundary as shown in Table 2.8.

Table 2.8: Relationship between Classifiability Measure and Boundary Complexity

Type of data boundary	**Boundary Length,** S	**Classifiability,** L
Linear	1	0.9562
Sinusoid ($1\frac{1}{2}$ periods)	3.2313	0.8415
Sinusoid (3 periods)	6.1393	0.5981
Sinusoid ($4\frac{1}{2}$ periods)	9.1021	0.4344

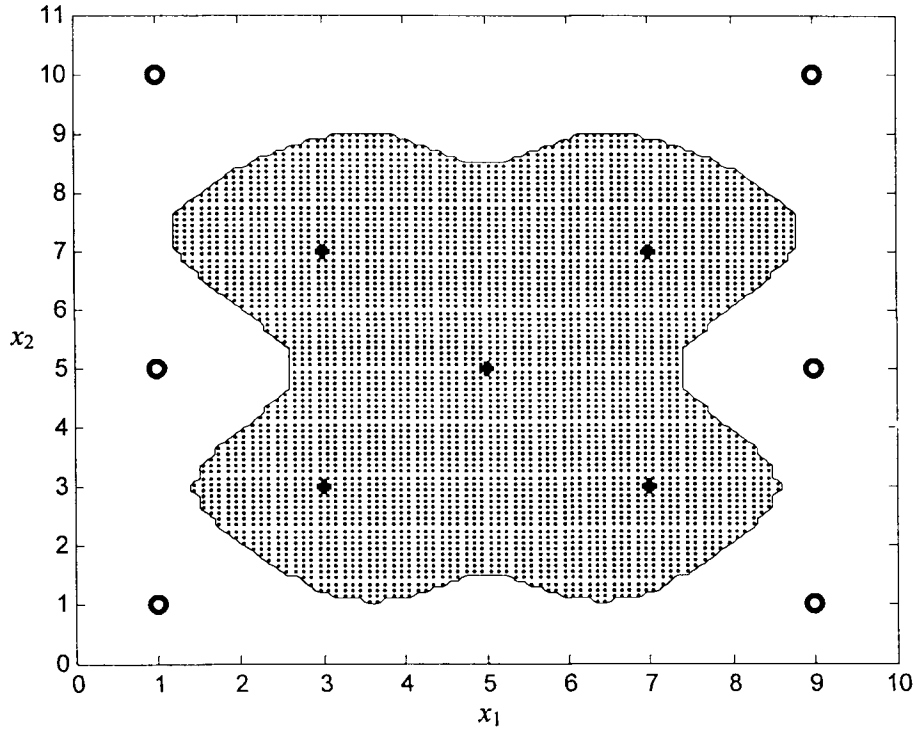


Fig. 2.8. Classification contours generated by Support Vector Classifier using a RBF kernel.

The data set with linear boundary had a higher classifiability while data sets with convoluted and increasing boundary length had lower classifiability. The decision boundary in Fig. 2.8, generated by the RBF-kernel based support vector classifier, is highly complex. On the other hand, the decision boundary in Fig. 2.7 generated by the SVM based tree type neural network is smooth and piecewise linear, with sharp non-interlacing boundaries between patterns of the two classes.

2.4.2 Real Data sets

Description.

The SVM based tree type neural network was next testd on the Breast Cancer-Winsconsin and the Heart statlog data sets. These are actual classification benchmark sets from the UCI Machine Learning database [Bla] and have continuous valued input attributes. The outputs of the data sets are binary.

A. Breast Cancer-Winsconsin data set

This breast cancer database originates from the University of Wisconsin Hospital, Madison. There are a total of 699 instances (458 benign, and 241 malignant) each having 9 continuous disease attributes. The missing attributes amount to about 0.3%.

B. Heart-statlog data set

This is a heart disease database with 13 continuous attributes. The number of instances are 270 (150 absence of heart disease, and 120 presence of heart disease). This is a complete data set with no missing attributes.

K-fold cross validation: Cross validation [Bre84, Dud2K3] is a very popular empirical technique for estimating the generalization error of any learning scheme and there are several versions. In each trial of K-fold cross validation, the training set is randomly split into K mutually exclusive subsets (folds) of appropriately equal size. The classifier is trained using (K–1) of the folds, and

tested on the fold left out. The mean of the test set error over K trials gives an estimate of the expected generalization error.

Both the data sets were preprocessed by the following guidelines before conducting the experiments.

i) Continuous attribute values were scaled to the range [0, 1] while binary attributes were encoded as 0/1.

ii) Some a priori data processing was needed for a few of the data sets, for some of the missing attribute values. A missing continuous attribute was replaced by the average of the available values. A missing discrete attribute value was assigned a value unknown; hence the corresponding input was considered as zero.

Results

The ten-fold cross-validation results are shown in Table 2.9. The learning accuracy on the training set was expectedly higher than the predictive accuracy on the test set.

On the same data sets, Table 2.9 also shows the comparison of our result with that of the N2C2S algorithm [Set2K1]. N2C2S uses a feed-forward neural network having a single hidden layer, the "tanh" activation function (Fig. 1.1b), and is trained using the conjugate gradient minimization algorithm. The results of our approach are comparable with that of N2C2S.

Table 2.9: Results of 10-fold cross-validation

	Our Result		N2C2S [Set2K1]	
Name of data set	**Training accuracy (%)**	**Testing accuracy (%)**	**Training accuracy (%)**	**Testing accuracy (%)**
Breast cancer-W	99.59 ± 0.27	95.32 ± 0.52	97.47 ± 0.06	96.58 ± 0.24
Heart-statlog	99.074 ± 0.64	77.67 ± 1.27	94.08 ± 1.35	77.56 ± 1.00

The training result by our approach is always better than other approaches as it tends to learn the complete training set. However such a learning strategy tends to suffer from overfitting. When this happens, the testing error is more. As this approach considers the error for each data point, the presence of outliers in the training data would lead to poorer generalization. On the other hand, N2C2S uses the validation set to stop training before overfitting occurs that might have resulted in better test set accuracy.

The proposed SVM based tree type neural network is capable of learning binary mappings and can potentially be used in digital systems. Any sequential data having binary valued outputs can be suitably modified (if required) to match the LP formulation (2.6 and 2.7). A scheme to learn binary sequences by SVM based tree type neural networks is discussed in the next section.

2.4.3 Learning Binary Sequences

A class of neural networks composed of multilayer perceptrons can be thought of where the output is computed as a nonlinear function of a window of past inputs and output samples

$$y(t) = f(u(t), u(t-1), u(t-2), \cdots, u(t-n), y(t-1), y(t-2), \cdots, y(t-m)), \tag{2.15}$$

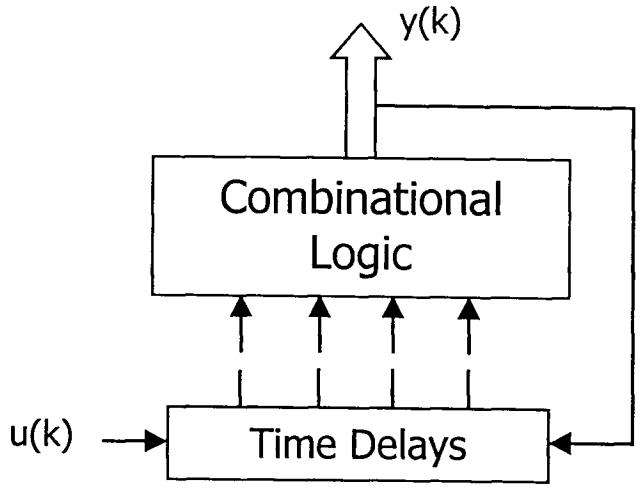


Fig. 2.9. Implementation of finite memory machine

where n and m are the size of the input and output windows, respectively, and $f(\cdot)$ is the nonlinear function. These models are capable of implementing arbitrary finite memory machines (FMM) as shown in Fig. 2.9. We now discuss how the tree type neural network can be used to binary sequences.

Example 1 [Jay2K2c]: Learn a 4-bit arbitrary binary sequence such that the current state generates the next immediate state at each step.

b_1	b_2	b_3	b_4
0	0	0	1
		↓	
1	1	1	0
		↓	
0	0	0	0
		↓	
1	0	1	1
		↓	
0	1	0	0

Here the present state is generated by the previous state and there are no external inputs. With this consideration, for the given sequence, the input-output bit patterns that should be available to a learning machine are given below.

INPUT				**OUTPUT**			
x_1	x_2	x_3	x_4	b_1	b_2	b_3	b_4
0	0	0	1	1	1	1	0
1	1	1	0	0	0	0	0
0	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0
0	1	0	0	0	0	0	1

Using $\Delta=1$, four tree structured neural networks were trained, one corresponding to each bit position. The delayed outputs generated by a network for a bit location affect its own output and also the outputs at other bit locations in the current state. Figure 2.10 shows the combination of 4 sub-networks, each of which is a tree type neural network for a bit location. The network is able to learn the 4-bit arbitrary binary sequence. It was found that starting from any state of the sequence generates the whole sequence repeatedly.

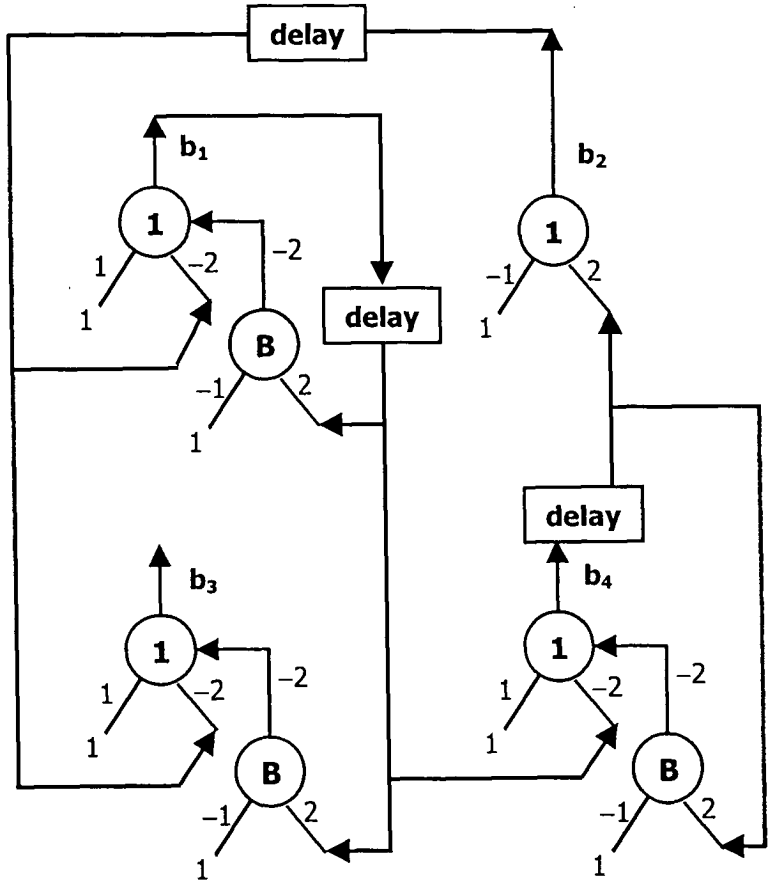


Fig. 2.10. Neural networks for generating the sequence [0001 → 1110 → 0000 → 1011 → 0100].

2.5 Conclusion

In this chapter, we proposed an algorithm for training a SVM based tree type neural network. The method proposes an elegant way of formulating a binary classification problem in the linear programming framework, and then obtaining the separating hyperplane for maximizing the margin of separation. This is achieved by using the maximum margin classifier type SVM that minimizes the L_2-norm of the weight vector. Each neuron in the network behaves

as a maximum margin classifier for the samples correctly learnt by it. The samples not separable at a node are taken care of by adding child neurons to the node in question. This approach obviates the need for an a priori choice of the number of neurons, number of hidden layers, learning parameters, or the kernel function in a SVM. The use of linear programming makes the formulation simpler. Efficient codes are available for solving LPPs. For a binary classification problem, a LP based approach generates piecewise smooth surfaces that are easy to implement. The LP based approach gives decision boundaries composed of piecewise linear segments providing sharp non-interlacing boundaries between the patterns of the two classes and increases the scope of better classifiability between the patterns.

The algorithm was tested on synthetic and real benchmark data sets. The two-dimensional illustrative example in Section 2.3.2 shows the working of the algorithm.

A limitation that is observed during the growth of the network is that the LP formulation using objective functions (2.6) or (2.8) with equality constraints (2.7a), and (2.7b), yields a trivial solution under some circumstances. This happens while learning training sets in which the number of samples of one class are relatively small as compared to the other. This problem is observed at some of the nodes lower down the tree in certain data sets. The trivial solution is highly biased towards the class that has a majority of the training patterns. This

results in the same training pattern occurring for the child neuron as in the parent neuron, leading to an infinite growth of the branch. If the growth of such a branch is terminated when the parent and the child have the same training set, the cost incurred is in terms of some patterns of the original training set not being learnt at all. Hence, our approach tends to learn the whole training set, though the training set accuracy has been slightly less than 100% for some real data sets. The problem of learning binary classification data by the LP based approach when the number of samples in the two classes is unbalanced, is discussed in Chapter 3.

Chapter 3

SVM N-tree: An Improved Learning Algorithm for Binary Classification

3.1 Introduction

In Chapter 2, we showed that given a completely labelled training data set $S_t = \{(x^i, y^i), i = 1,2,...,N; x^i \in \Re^n; y^i \in \{0,1\}\}$, the problem of binary classification involves obtaining a set of weights $w \in \Re^n$, and bias $w_0 \in \Re$ such that

$$\sum_{d=0}^{n} w_d x_d^{(i)} \begin{cases} \geq \Delta, \text{ if } y^i = 1 \\ \leq -\Delta, \text{ if } y^i = 0 \end{cases}; x_0^{(i)} = 1; 1 \leq i \leq \text{card}(S_t) \tag{3.1}$$

where i denotes the i-th sample; $\text{card}(S_t) = N$ = total number of samples in the data set, and Δ is a positive quantity introduced for better separation of the data. Chapter 2 also discusses the formulation of the binary classification problem as the following linear programming problem (LPP).

$$\text{(LPP)} \qquad \text{Minimize } \sum_{i=1}^{m} s_i^{(-)} + \sum_{i=1}^{k} s_i^{(-)} \tag{3.2}$$

subject to the following constraints

$$\sum_{d=0}^{n} \left(w_d^{(+)} - w_d^{(-)}\right) x_d^{(i)} = \Delta + \left(s_i^{(+)} - s_i^{(-)}\right), \text{ if } y^i = 1; i \in I_1 \tag{3.3a}$$

$$\sum_{d=0}^{n} \left(w_d^{(+)} - w_d^{(-)}\right) x_d^{(i)} = -\Delta - \left(s_i^{(+)} - s_i^{(-)}\right), \text{ if } y^i = 0; i \in I_0 \tag{3.3b}$$

for $\text{card}(I_1) = m$; $\text{card}(I_0) = k; x_0^i = 1..$

The objective function used in (3.2) is similar to the one used in Phase I of the two-phase Simplex method (Kam97). Bennett et al. [Ben92] used an objective function that assigns relative weights to the slack variables as shown in (3.4) along with the constraints (3.3a) and (3.3b).

$$\text{Minimize } \sum_{i=1}^{m} \frac{1}{m} s_i^{(-)} + \sum_{j=1}^{k} \frac{1}{k} s_j^{(-)} \tag{3.4}$$

The objective function in (3.2) is a greedy choice that assigns equal weights to all the slack variables. The use of (3.4) over (3.2) has an advantage as it ensures that the trivial solution $w = \mathbf{0}$, $w_0 = 0$, and ensures better convergence [Ben92]. Formulations using (3.2) and (3.4) have been used in Chapter 2, and reported in [Jay2K2b] for finding the initial hyperplane to dichotomize the training set.

Neurons at different levels of a tree-structured neural network have different training sets and the relative numbers of the two classes of data also vary at different nodes. As the network grows, some of the terminal nodes often have too few training samples of one class as compared to the other. For many data sets, the above-formulated LPPs were unable to dichotomize the training sets and yielded trivial solutions, e.g. a hyperplane that is highly biased towards the class whose samples are larger in number. For such training sets, the training set for the child neuron ends up being identical to that of the parent neuron. This leads to an infinite growth of the branch, and growth of such a branch needs to be forcibly terminated. The cost incurred is in terms of some patterns of the original training set not being learnt at all. Hence the training set

learning accuracy of the SVM based tree type neural network on some real benchmark data sets, as reported in Table 2.9 of Chapter 2, was slightly less than 100%.

LPP with Generalized Objective Function

The problem of learning a binary classification data set by the LP based approach when the number of samples of one class is much smaller than the other was overcome by using two new types of objective functions. Assigning weights $\lambda > 0$, and $\mu > 0$ respectively to the slack variables $s_i^{(-)}$ in (3.2), the LPP with the generalized objective function may be stated as below.

$$\text{(LPP)} \qquad \text{Minimize } \sum_{i=1}^{m} \lambda s_i^{(-)} + \sum_{i=1}^{k} \mu s_i^{(-)} \tag{3.5}$$

subject to the following constraints

$$\sum_{d=0}^{n} \left(w_d^{(+)} - w_d^{(-)}\right) x_d^{(i)} = \Delta + \left(s_i^{(+)} - s_i^{(-)}\right), \text{ if } y^i = 1; i \in I_1 \tag{3.6a}$$

$$\sum_{d=0}^{n} \left(w_d^{(+)} - w_d^{(-)}\right) x_d^{(i)} = -\Delta - \left(s_i^{(+)} - s_i^{(-)}\right), \text{ if } y^i = 0; i \in I_0 \tag{3.6b}$$

for card $(I_1) = m$; card $(I_0) = k; x_0^i = 1$.

For a given training set, changing the relative weightings λ and μ of the slack variables $s_i^{(-)}$ in the objective function (3.5) can lead to different linear programming solutions. Depending upon the number of samples m and k in the

two classes, we prescribe different choices for λ and μ that lead to four possible kinds of LPPs. These are summarized in Table 3.1.

Table 3.1: Types of LPP

Type	Value of λ and μ	Case
Type I	$\lambda = 1; \mu = 1$	Substantial values of m & k
Type II	$\lambda = \frac{1}{m}; \mu = \frac{1}{k}$	Substantial values of m & k
Type III	$\lambda = \mathrm{M}; \mu = \frac{1}{k}; \mathrm{M} \geq 1$	$m << k$
Type IV	$\lambda = \frac{1}{m}; \mu = \mathrm{M}; \mathrm{M} \geq 1$	$k << m$

LPPs of Type I and Type II are suitable for cases where there is no substantial numerical imbalance between the numbers of training samples of Classes 1 and 0. Type I and Type II LPPs were used in Chapter 2 for finding the initial hyperplane to dichotomize the training set while learning a neuron of the tree-structured neural network.

LPPs of Types III and IV have been introduced to overcome the learning problem that occurs at some of the terminal nodes, when the number of samples of one of the two classes is much smaller than the other. Type III LPPs are used when the number of samples of Class 1 is much less than that of Class 0. In such a case, λ and μ are chosen such that a higher value $\lambda = (\mathrm{M})$ is used, so as to assign greater importance to the slack variables $s_i^{(-)}$ corresponding to the patterns of Class 1. This increases the possibility of elimination of these variables

from the basis set during early runs of the Simplex algorithm. LPPs of Type IV also behave in the same way for the case where the training set samples of Class 0 are much fewer than the number of Class 1 samples.

For, $m << k$, possible minimum value of m, $m_{\min} = 1$. Using LPP of Type II,

$$\lambda_{\max} = \frac{1}{m_{\min}} = 1$$

For, $k << m$, possible minimum value of k, $k_{\min} = 1$. Using LPP of Type II,

$$\mu_{\max} = \frac{1}{k_{\min}} = 1$$

Even such values of $\lambda_{\max}$ and $\mu_{\max}$ tends to give trivial solution for unbalanced binary classification data. So in Type III $(m << k)$ and Type IV $(k << m)$, deliberately a higher value of λ and μ was used for the variables of the class having lower cardinality. A higher value (>1), say, $\mathbf{M} = 2$ was used that was found to be sufficient for variable exchange during simplex method to give non-trivial solution. Although $\mathbf{M} = 2$ is sufficient, choosing a very large value of $\mathbf{M}$ may not be desirable.

Note: When objective functions of Type III and Type IV are used for non-skewed data sets, a biased or trivial solution may be found.

The LPP solution gives a separating hyperplane that tries to correctly classify as many samples as possible. Incorporating the four types of LPP for learning a binary data, we propose a complete learning algorithm in the Section3.2.

3.2 SVM N-tree Learning Algorithm

3.2.1 Multiple Hyperplanes for a Training Data

For a fixed training data set, i.e. for the same set of constraints (3.6a), and (3.6b), there exists the possibility of different hyperplane solutions of the LPP due to the following reasons:

1) Use of different values of λ and μ in (3.5), as described in Table 3.1.

2) For a fixed value of λ and μ, the original training set, and the training set obtained by reordering the samples based on certain criteria may result in different hyperplane solutions, as they can have a different sequence of variable exchanges in the Simplex method.

3.2.2 Choice of the Hyperplane

The proposed learning algorithm uses both a training set S_t and a validation set S_v. During network growth, the training set is used to determine different hyperplanes depending on the type of LPP used, while the validation set acts as a guide regarding the choice of the best hyperplane. A choice has to be made from the possible hyperplane solutions of a training set keeping in mind

1) the number of correctly learnt samples;

2) dichotomization of the validation set; and

3) higher margin between the two classes of the correctly learnt samples.

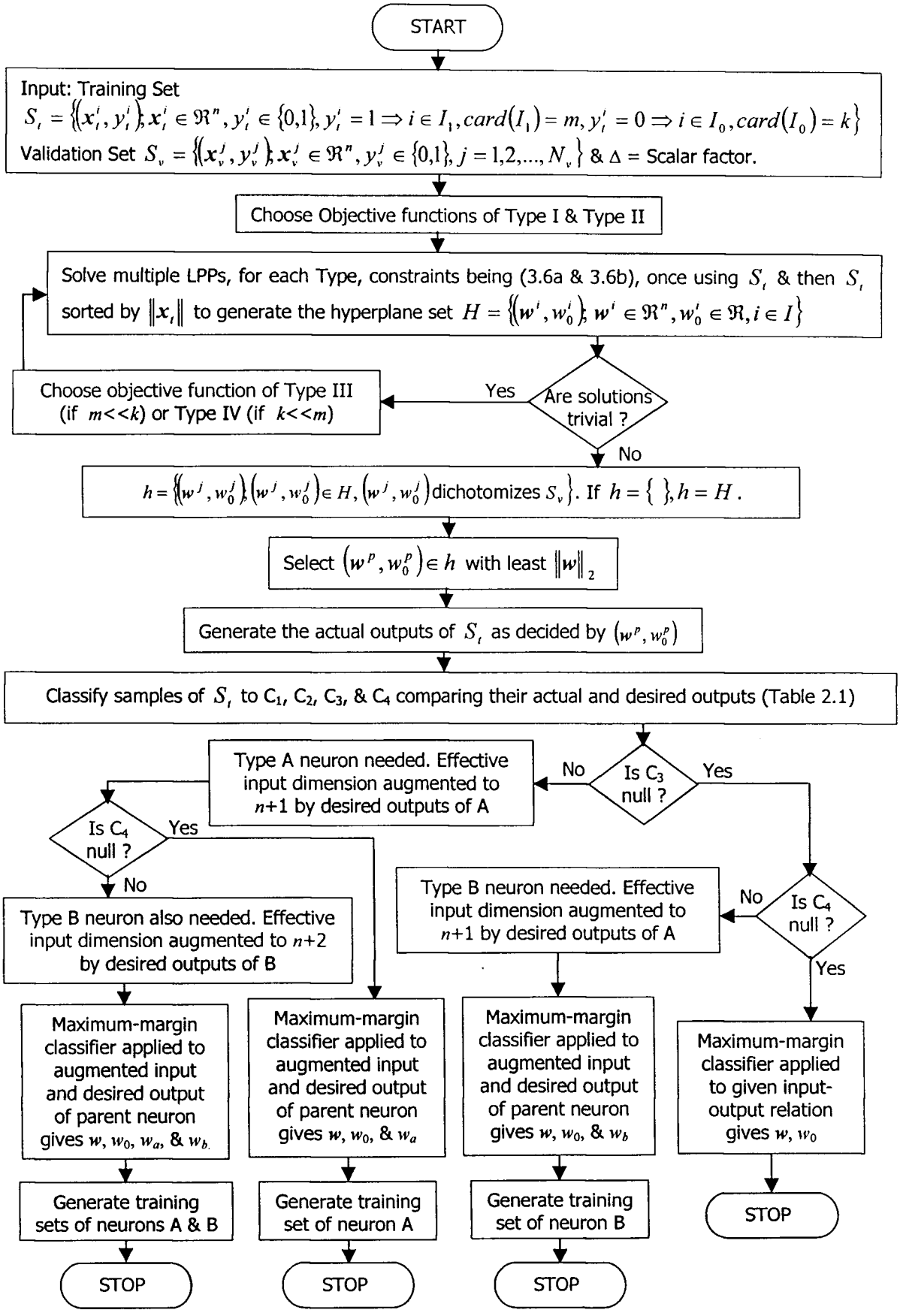


Fig. 3.1. Flowchart for learning at a neuron.

The flowchart in Fig. 3.1 describes the implementation of the learning algorithm at a neuron. The algorithm is applied at each neuron starting from the root and thus the complete network is grown.

3.3 Experimental Results

The SVM N-tree algorithm was implemented using MATLAB (version 6.5 (R13)) [MAT], and was run on a 800 MHz Pentium III PC. The initial hyperplane was obtained through the solution of the LPPs (3.5, 3.6a, and 3.6b), by using the modified Simplex method [Mar90]. The maximum margin classifier for obtaining the final classification boundary was implemented by using the .m-files from the Support Vector Machines Toolbox [Gun98] in MATLAB 6.5 environment. To speed up the working of the algorithm, instead of directly using the MATLAB function *quadprog* while solving the SVM-dual, we used decomposition methods and working set selection strategies such as those of SVM_*light* [Joa99]. The SVM N-tree was tested on benchmark data sets, the results of which are summarized in this section.

Real Data sets

Description:

Fifteen data sets were classified by using the SVM N-tree. These were benchmark data sets from the UCI Machine Learning database [Bla], and from the Machine Learning Research Group at the Department of Computer Science,

University of Waikato [Wai]. Table 3.2 provides details regarding the data sets, while the data sets are briefly described in the sequel. The outputs for all data sets are binary.

Table 3.2: Description of Data sets

Data Set	Size	Missing Values(%)	Input Attributes		
			Continuous	Binary	Nominal
Australian	690	0.6	6	4	4
Breast Cancer	286	0.3	0	3	6
Breast Cancer-W	699	0.3	9	0	0
German	1000	0.0	6	3	4
Glass(G2)	163	0.0	9	0	0
Heart-c	303	0.2	6	3	4
Heart-h	294	20.4	6	3	4
Heart-statlog	270	0.0	13	0	0
Hepatitis	155	5.6	6	13	0
Horse colic	368	23.8	7	2	13
Ionosphere	351	0.0	33	1	0
Labor	57	3.9	8	3	5
Pima-indians	768	0.0	8	0	0
Sonar	208	0.0	60	0	0
Vote	435	5.6	0	16	0

A. Australian Data set

This data set is the credit card approval data concerning credit card application. The total number of instances are 690 (383 Class 1, and 307 Class 2) having a mix of continuous, binary, and nominal attributes.

B. Breast Cancer Data set

This data set pertains to breast cancer data obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. The original number of instances are 286 (201 no-recurrence-events, and 85 recurrence-events) each having 9 attributes.

C. Breast Cancer-Winsconsin Data set

This breast cancer database was obtained from the University of Wisconsin Hospitals, Madison. There are a total of 699 instances (458 benign, and 241 malignant) each having 9 continuous disease attributes.

D. German Data set

This data set is the German credit data from Institut für Statistik und Okonometrie, Hamburg University. It has 13 attributes selected from amongst the original 20 attributes. There are a total of 1000 instances (700 good, and 300 bad).

E. Glass (G2) Data set

This data set pertains to the glass identification data set and its binary classification was motivated by criminological investigation. The original number of instances are 214 (window glass, and non-window glass) having 9 continuous values attributes. In our work we have used the window glass instances numbering 163 (87 float processed, and 76 non-float processed being the two classes).

F. Heart-c Data set

This heart disease database is from the Cleveland Clinic Foundation, each instance having 13 input attributes. There are 303 instances: 139 instances having vessel diameter narrowing greater than 50%, and 164 instances in which the vessel diameter narrowing was less than 50%.

G. Heart-h Data set

This heart disease database is from the Hungarian Institute of Cardiology, Budapest, and each instance having 13 input attributes. There are 294 instances: 106 instances having vessel diameter narrowing greater than 50%, and 188 instances in which the vessel diameter narrowing was less than 50%.

H. Heart-statlog Data set

This is a heart disease database with 13 continuous attributes. The number of instances are 270 (150 absence of heart disease, and 120 presence of heart disease).

I. Hepatitis Data set

This binary classification data set is from the hepatitis domain, with each instance having 19 input attributes (like age, sex, steroid, antiviral, fatigue, bilirubin, albumin etc) that determine the disease. There are 155 instances (32 die, and 123 live).

J. Horse colic Data set

This data set contains 368 horse colic instances, and the 22 input attributes are used to predict whether or not the lesion is surgical. There are 232 lesion, and 136 non-lesion cases.

K. Ionosphere Data set

These data classify the radar returns as "good", or "bad" signifying if there is or isn't any evidence of structure in the ionosphere as observed. There are a total of 351 patterns (224 "good" patterns, and 127 "bad" patterns).

L. Labor Data set

This is a two-class identification problem from the final settlements of labor negotiations in Canadian industry, with each instance having 16 attributes. There are 57 instances (37 good negotiation, and 20 bad negotiation).

M. Pima Indians Data set

This is the Pima Indians diabetes database, classified on the basis of 8 continuous attributes. There are 768 instances (268 tested positive for diabetes, and 500 tested negative for diabetes).

N. Sonar Data set

This is a two-class identification problem of underground targets (rock, or mines) on the basis of 60 attributes. There are a total of 208 patterns (111 mines, and 97 rock patterns).

O. Votes Data set

This binary classification data is from the 1984 United States congressional voting records database having 435 instances (267 democrats, and 168 republicans). Each instance consists of 16 binary valued attributes.

The performance of our proposed algorithm was evaluated by using the ten-fold cross validation technique, that is a very popular empirical technique for estimating the generalization error of a learning scheme as described in Section 2.4.2. In our case, from the (K–1) folds used for training, a single fold has been separated out, that can be considered as the validation set, and which is used to

guide the network growth by choosing one hyperplane from the different hyperplanes obtained by solving different types of LPPs. All the data sets were preprocessed as described in Section 2.4.2.

Results

Table 3.3 shows the mean and variance of the number of perceptrons needed, and the accuracy values on the training set, validation set, and the test set for each of the data sets on application of the ten-fold cross-validation methodology.

Table 3.3: 10-fold Cross validation results on UCI Machine Learning data sets

Data Set	Number of Perceptrons	Training Set Accuracy (%)	Validation Set Accuracy (%)	Test Set Accuracy (%)
Australian	33.2 ± 3.39	100 ± 0.0	82.9 ± 4.36	81.45 ± 3.53
Breast Cancer	43.3 ± 4.056	100 ± 0.0	69.64 ± 7.69	68.6 ± 4.74
Breast Cancer-W	11.7 ± 3.26	100 ± 0.0	95.86 ± 4.17	95.85 ± 2.89
German	96.7 ± 9.04	100 ± 0.0	69.4 ± 4.57	68.8 ± 3.55
Glass(G2)	14.1 ± 3.18	100 ± 0.0	77.87 ± 4.52	76.03 ± 4.76
Heart-c	14.3 ± 1.56	100 ± 0.0	80.86 ± 6.96	81.85 ± 7.88
Heart-h	18.6 ± 4.35	100 ± 0.0	81.24 ± 7.42	80.93 ± 5.91
Heart-statlog	12.7 ± 2.31	100 ± 0.0	81.11 ± 8.97	78.89 ± 5.80
Hepatitis	4.4 ± 1.5	100 ± 0.0	81.83 ± 6.98	82.54 ± 6.92
Horse colic	9.5 ± 1.43	100 ± 0.0	80.99 ± 3.97	79.37 ± 5.91
Ionosphere	3.0 ± 0.0	100 ± 0.0	89.18 ± 3.47	88.61 ± 4.01
Labor	1.3 ± 0.48	100 ± 0.0	96.33 ± 7.77	87.67 ± 8.61
Pima-indians	94.7 ± 6.66	100 ± 0.0	69.8 ± 3.91	69.8 ± 4.01
Sonar	1 ± 0.0	100 ± 0.0	100 ± 0.0	75.48 ± 5.28
Votes	5 ± 1.56	100 ± 0.0	95.17 ± 2.75	95.16 ± 2.30

The training nature of SVM N-tree ensures 100% accuracy on the training set for all the data sets. However such a learning strategy suffers from overfitting, leading to an increase in the test set error. As the approach considers the error at each data point, the presence of outliers in the training set can lead to poor generalization.

Note: The Sonar data set is of size 208 and has 60 input attributes. High dimensional data sets often have a much lower "essential dimension", with many of the features being irrelevant or redundant. Due to the high dimensionality of the data, there are more free variables in the weight vector. Consequently overfitting of the training set may have taken place and one neuron was sufficient to learn the whole training set. This could be the reason for the high difference between the training set and test set accuracies.

The performance of the SVM N-tree was compared with some other reported methods. We compared our result with that of Setiono's N2C2S algorithm [Set2K1]. The results of N2C2S are based on highly tuned values of the number of hidden units of a feedforward neural network using the 'tanh' activation function, decided by the performance on the cross-validation set. Simple runs of the algorithm during the tuning of the number of hidden units generate much poorer results and the final network topology that gives best results based on the performance of the cross-validation set have been chosen. Here the performance also depends on the randomly initialized weights. The use of the validation set to stop training before overfitting occurs might have resulted

in better test set accuracy in some of the cases. We also compared our results with those obtained by M5' algorithm [Fra98] that generates binary decision trees by using a linear regression function at the leaf nodes. Frank et al. [Fra98] compared their result with that of C5.0, a modification to the widely used C4.5 [Dud2K3] decision tree method for classification proposed by Quinlan. Results of the application of C5.0 on the same data sets obtained from [Fra98] have been quoted in Table 3.4 for comparison. A value in the table having a bold circle (•) indicates that the test set accuracy of SVM N-tree is higher than that of the other indicated method.

The proposed approach obviates the need for a priori choice of the number of neurons, number of hidden layers, any learning parameter, or the kernel function in a SVM. Linearly inseparable data are classified by using optimally placed piecewise-linear (PWL) boundaries that are easier to interpret.

Figure 3.2 pictorially depicts the mean test set accuracies and error bars of the ten-fold cross-validation exercise, and gives a ready comparison among the methods. Valuewise test results of the proposed method are comparable to other algorithms and better in some of the cases. In fact, other algorithms also show similar behaviour. The proposed approach also ensures the highest accuracy on the training set, that may be required for certain applications.

Table 3.5 compares the total neuron count obtained by using SVM N-tree with that of the N2C2S. Total number of neurons include input, output, and

hidden neurons. As can be seen, the networks trained by SVM N-tree generally needed far fewer neurons as compared to N2C2S.

Table 3.4: Test Set Comparison with Other Approaches

Data Set	SVM N-tree	N2C2S	C5.0	M5'
Australian	81.45 ± 3.53	84.83 ± 0.71	85.3 ± 0.5	85.8 ± 0.9
Breast Cancer	68.6 ± 4.74	67.8 ± 2.17 •	73.3 ± 1.6	69.6 ± 2.3
Breast Cancer-W	95.85 ± 2.89	96.58 ± 0.24	94.5 ± 0.3 •	95.3 ± 0.3 •
German	68.8 ± 3.55	70.1 ± 1.65	71.2 ± 1.0	72.9 ± 0.7
Glass(G2)	76.03 ± 4.76	77.91 ± 2.5	78.7 ± 2.1	81.8 ± 2.27
Heart-c	81.85 ± 7.88	82.47 ± 1.95	76.8 ± 1.4 •	80.9 ± 1.4 •
Heart-h	80.93 ± 5.91	81.63 ± 1.6	79.8 ± 0.9 •	79.0 ± 0.8 •
Heart-statlog	78.89 ± 5.80	77.56 ± 1.00 •	78.7 ± 1.4 •	82.2 ± 1.0
Hepatitis	82.54 ± 6.92	81.94 ± 3.34 •	79.3 ± 1.2 •	81.9 ± 2.2 •
Horse colic	79.37 ± 5.91	78.97 ± 1.29 •	85.3 ± 0.6	84.6 ± 0.7
Ionosphere	88.61 ± 4.01	89.52 ± 2.26	88.9 ± 1.6	89.7 ± 1.2
Labor	87.67 ± 8.61	91.9 ± 4.1	78.1 ± 4.8 •	79.7 ± 4.6 •
Pima-indians	69.8 ± 4.01	76.04 ± 0.86	74.5 ± 1.2	76.2 ± 0.8
Sonar	75.48 ± 5.28	76.51 ± 1.56	74.7 ± 2.8 •	78.5 ± 3.4
Votes	95.16 ± 2.30	96.09 ± 0.48	96.3 ± 0.6	96.2 ± 0.3

Table 3.5: Neuron Count Comparison between SVM N-tree and N2C2S

Data Set	SVM N-tree	N2C2S
	Number of Perceptrons	Total Number of Neurons
Australian	33.2 ± 3.39	54.22 ± 1.38
Breast Cancer	43.3 ± 4.056	59.06 ± 0.84
Breast Cancer-W	11.7 ± 3.26	15.4 ± 0.51
German	96.7 ± 9.04	73.54 ± 1.43
Glass(G2)	14.1 ± 3.18	19.06 ± 0.84
Heart-c	14.3 ± 1.56	30.6 ± 1.13
Heart-h	18.6 ± 4.35	32.94 ± 1.14
Heart-statlog	12.7 ± 2.31	21.52 ± 0.99
Hepatitis	4.4 ± 1.5	37.66 ± 1.16
Horse colic	9.5 ± 1.43	68.88 ± 0.38
Ionosphere	3.0 ± 0.0	41.2 ± 0.87
Labor	1.3 ± 0.48	34.46 ± 0.4
Pima-indians	94.7 ± 6.66	16.88 ± 0.99
Sonar	1 ± 0.0	68.18 ± 0.48
Votes	5 ± 1.56	39.42 ± 0.61

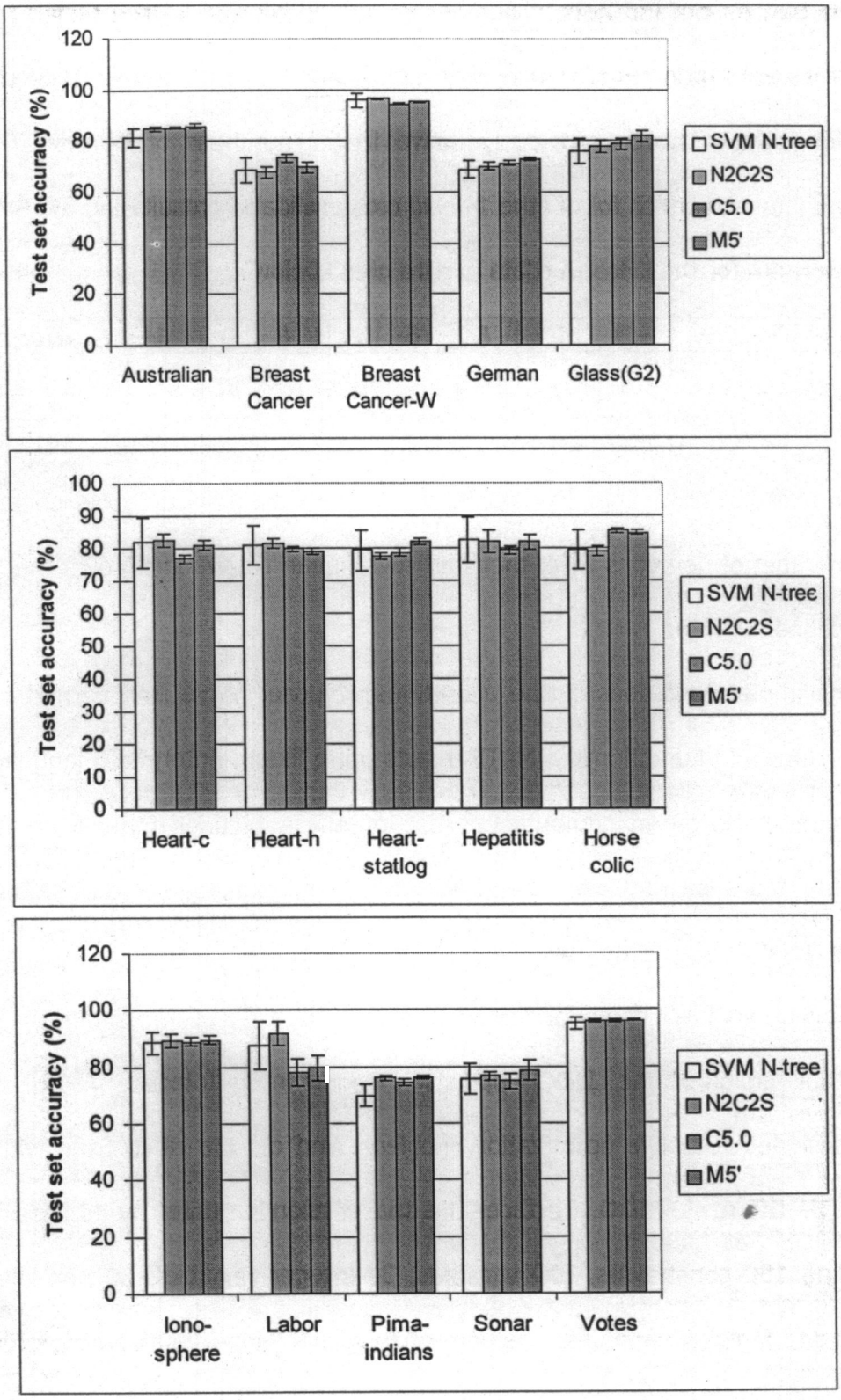


Fig. 3.2 Comparison with Other Approaches.

The perceptron and neuron number are the mean and variance over the 10-fold cross validation runs. Hence their values are fractions. For each of the runs (folds), the networks obviously have integer number of neurons. The neuron numbers for each fold of the 10-fold cross-validation result using the SVM N-tree algorithm for the Heart-h data set are given below:

The above result shows that the networks corresponding to each fold contain integral number of neurons while their statistics (mean and variance) over the 10 folds fractions.

The implementations of the algorithms proposed have been carried out using MATLAB 6.5. Functionally, MATLAB acts more like an interpreted language that renders it slower in computation. Use of efficient codes for solving the optimization tasks like linear and quadratic programming instead of the MATLAB functions would increase the overall algorithmic speed.

Simulations have been conducted on a 3.2 GHz, 1 GB RAM, Pentium IV PC using a trial version of the LINDO Application Programming Interface (API) [Lin], that is designed to solve optimization problems and can be called from within MATLAB via the **mxLINDO** interface. The trial version is limited by its capacity of handling 150 constraints, 300 variables, 30 integer variables, 30 non-linear formulas and 5 global variables. The performance of SVM N-tree algorithm coded

in MATLAB and using the MATLAB-LINDO interface (using a barrier type solver) are shown in Table 3.6

Table 3.6: Run Time for 10-fold Cross validation on SVM N-tree using MATLAB and MATLAB-LINDO interface

Data set	SVM N-tree (MATLAB only)	SVM N-tree (MATLAB-LINDO interface)
G2	547 secs	348.3 secs
Hepatitis	287.1 secs	96.2 secs
Labor	90 secs	64.4 secs

Results correspond to 10-fold cross validation experiment on the G2 ($N=163; n=9$), Hepatitis ($N=155; n=19$) and Labor ($N=57; n=16$) data sets (N is the size of the data set, n is the input dimension) obtained from the UCI Machine Learning repository.

The above results suggest that the use of better LP codes can lead to considerable improvement in the run time.

3.4 Pruning a network trained by SVM N-tree

Pruning techniques are one form of adaptive structure neural networks, that are used to incrementally modify a network model to perform a given task. A complete overview of some important pruning techniques proposed so far has been given in Section 1.2. The general principle of these techniques is to begin with a larger trained network, and then remove redundant network connections and/or hidden units, based on some performance criterion. Different pruning approaches not only result in a reduction in model complexity, but also reduce the overfitting that occurs in large networks.

Procedure

The cross-validation performance of the SVM N-tree algorithm has been given in Section 3.3. We chose to observe the effect of pruning during cross-validation of the network constructed by SVM N-tree. The pruning was performed according to the following steps.

Step 1: Generate all the tree networks during the cross-validation runs and store the networks corresponding to each sub-set (fold).

Step 2: At a sub-set (fold), tabulate the validation set accuracy of the whole network using the stored network, as each neuron is added one by one from the top.

Step 3: Starting from the top, retain the network upto the node where the validation set accuracy is maximum.

Step 4: At each sub-set (fold), compute the training set, validation set, and the test set accuracies, with the number of neurons.

Step 5: Compute the mean of the training set, validation set, and test set accuracies, and the number of neurons over all the sub-sets (folds).

Results

Pruning of the network trained by SVM N-tree following the above procedure was carried out over 7 chosen data sets used in Section 3.3. Table 3.7 gives ten-fold cross-validation results of pruning a network trained by SVM N-tree.

Table 3.7: 10-fold Cross-validation results on UCI Machine Learning data sets after pruning a network trained by SVM N-tree

Data Set	Mean Number of Perceptrons	Training Set Accuracy (%)	Validation Set Accuracy (%)	Test Set Accuracy (%)
Breast Cancer-W	3.3	97.15 ± 1.46	96.99 ± 2.73	95.99 ± 2.67
Heart-c	3.1	86.84 ± 3.4	85.49 ± 6.03	83.51 ± 5.98
Heart-h	3.7	85.67 ± 2.56	86.38 ± 6.85	81.6 ± 8.16
Heart-statlog	3.4	88.61 ± 5.68	84.81± 6.63	80 ± 5
Hepatitis	1.8	93.06 ± 5.67	86.41 ± 7.14	83.16 ± 8.85
Horse colic	1.7	88.58 ± 1.84	86.7 ± 4.26	85.06 ± 5.3
Ionosphere	1.5	97.50 ± 1.77	90.04 ± 3.03	89.18 ± 3.2

As expected, the training set accuracy has reduced after pruning. The minimal tree is chosen by stopping learning at the node where the validation set accuracy is maximum. Consequently, the validation set accuracy has increased for all the data sets.

Table 3.8 shows the comparison of test set accuracies of N2C2S [Set2K1], M5', and C5.0 [Fra98], with that of the pruned network trained by the SVM N-tree algorithm.

Table 3.8: Comparison of Test Set accuracies of Other Approaches with the pruned network trained by SVM N-tree

Data Set	SVM N-tree	N2C2S	C5.0	M5'
Breast Cancer-W	95.99 ± 2.67	96.58 ± 0.24	94.5 ± 0.3 •	95.3 ± 0.3 •
Heart-c	83.51 ± 5.98	82.47 ± 1.95 •	76.8 ± 1.4 •	80.9 ± 1.4 •
Heart-h	81.6 ± 8.16	81.63 ± 1.6	79.8 ± 0.9 •	79.0 ± 0.8 •
Heart-statlog	80 ± 5	77.56 ± 1.00 •	78.7 ± 1.4 •	82.2 ± 1.0
Hepatitis	83.16 ± 8.85	81.94 ± 3.34 •	79.3 ± 1.2 •	81.9 ± 2.2 •
Horse colic	85.06 ± 5.3	78.97 ± 1.29 •	85.3 ± 0.6	84.6 ± 0.7 •
Ionosphere	89.18 ± 3.2	89.52 ± 2.26	88.9 ± 1.6 •	89.7 ± 1.2

• indicates that the test set accuracy of the pruned network trained by SVM N-tree is higher than that of other indicated methods.

Pruning by retaining the minimal network upto the node where the validation set accuracy is maximum, results in a reduction on the number of perceptrons in the network.

3.5 Conclusion

In this chapter, we proposed SVM N-tree, a new approach for building a SVM based tree type neural network. The method proposes an elegant way of formulating a binary classification problem in the linear programming framework, and then obtaining the maximum margin hyperplane at each node of the tree. Mis-classified samples at a node are taken care of by adding child neurons to the node in question. This approach obviates the need for an a priori choice of the number of neurons, number of hidden layers, or the kernel function in a SVM.

In the LP formulation of the binary classification problem, a generalized objective function was proposed [Deb2K3]. Multiple objective functions were used to obtain better solutions for data sets in which the number of samples of one class was much smaller than the other.

The ten-fold cross-validation performance of the network trained by SVM N-tree was assessed on real benchmark data sets and was found to compare favorably with that of other reported algorithms. A procedure for pruning the SVM N-tree network was devised, that was based on the validation set

performance. Though the pruned network has reduced training set accuracy as compared to the complete network, the use of a validation set based criterion for pruning results in improvement in its predictive capability.

To conclude, the SVM N-tree algorithm guarantees complete learning of any binary classification data, and can be readily used to learn a set of system-generated labelled data having binary valued outputs.

The motivation for solving the binary classification problem by using a LP-based formulation stems from the easy understandability of its representation and simplicity of its solution. Efficient methods for high speed implementation of linear programming methods are available. In view of the advantages of LP-based formulations of the binary classification problem, we investigated whether the strategy can be extended to the task of function approximation. It is also desirable to obtain the hypothesis of a function approximator using a minimum number of mathematical operations. This motivates building constructive networks for function approximation using LP formulations, and devising techniques to obtain an approximator with a minimal computational overhead. We discuss such approaches in Chapter 4.

Chapter 4

Neural Networks for Function Approximation

4.1 Introduction

The objective of any function approximation problem is to learn a continuous nonlinear mapping $f:\Re^n \to \Re$ (or a good approximation of it). Specific information about f obtained from experimentation is provided by way of a sample set $S_t = \{(x^i, y^i), x^i \in \Re^n, y^i \in \Re, i \in I, \text{card}(I) = N\}$, that contains samples of f at a finite number of points in $\Re^n$. Classification is a special case of function approximation in which the range of the function's output y is restricted to one of M ($M \geq 2$) values. A particular case of classification is binary classification in which $M = 2$.

A function approximator may be represented parametrically as $\hat{f}(x,\Theta)$, where $\Theta \in \Re^p$ is a vector of parameters used in the definition of the approximator's mapping. If $\Gamma^p \subset \Re^p$ denotes the set of all values that the parameters of the approximator may take, the approximator $\hat{f}$ belongs to a class of functions $G = \{\hat{f}(x,\Theta) : \Theta \in \Gamma^p, p \geq 0\}$.

Definition: A function $f:D \to \Re$ may be uniformly approximated on $D \subseteq \Re^n$ by functions of class G if for each $\varepsilon > 0$, there exists some $\hat{f} \in G$ such that, $\sup_{x \in D} |\hat{f}(x,\Theta) - f(x)| < \varepsilon$.

In this definition, the choice of an appropriate $\hat{f}(x,\Theta)$ can depend on ε; hence for a fixed $\varepsilon > 0$, some $\hat{f}(x,\Theta) \in G$ may result in $\sup_{x \in D} \left| \hat{f}(x,\Theta) - f(x) \right| < \varepsilon$ while others may not. The number of parameters $p \geq 0$ that defines the approximator $\hat{f}(x,\Theta)$ for achieving a particular $\varepsilon > 0$ is also not specified in the above definition. Generally, more number of parameters p ensures smaller values of ε.

Definition: A mathematical structure defining a class of functions G_1 is said to be a universal approximator for functions of class G_2 if each $f \in G_2$ may be uniformly approximated by G_1.

The above statement means that an approximating structure can act as approximators for a particular class of functions. Depending on the characteristics of the approximator, several types of approximation are possible.

4.2 Approximator Types

Some conventional approximation techniques that are named according to the basic building blocks used for approximation are the following:

4.2.1 Step Approximation

A simple method of approximation is to use step type functions [Bar64] to uniformly approximate continuous functions in one dimension. A function

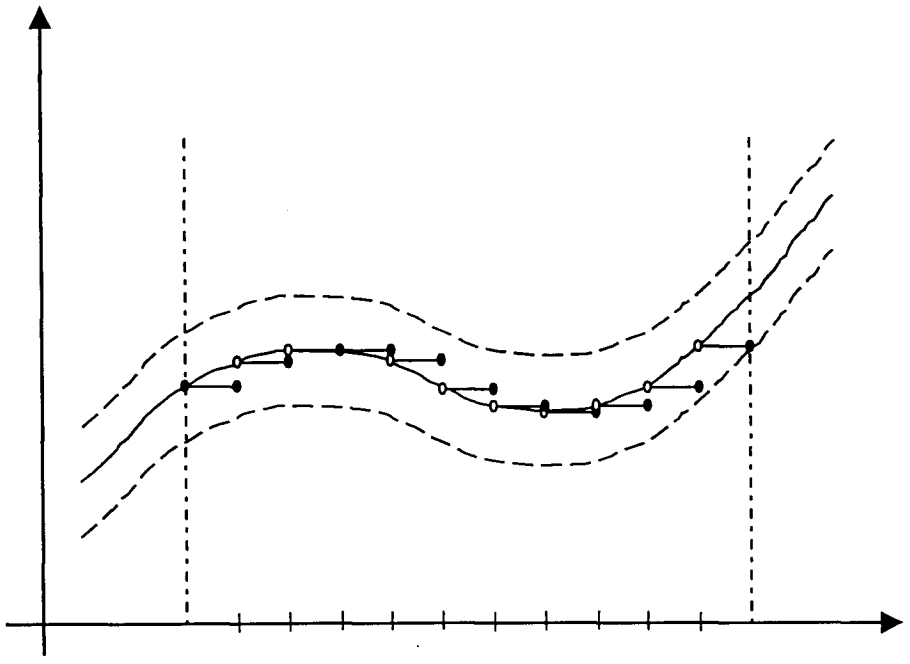


Fig. 4.1. Approximating a continuous function with a step function

$\hat{f}(x): D \to \Re$ for $D \subset \Re$ may be called as a **step type function** if it takes only a finite number of distinct values, with each value assigned over one or more disjoint intervals.

Figure 4.1 shows an example of step approximation to a continuous function within the predefined domain of the independent variable.

4.2.2 Piecewise Linear Approximation

Succeeding the step function in the hierarchy of approximation structures is the ramp function with fixed gradient: positive, negative, or zero. A piecewise linear function contains several ramp functions. A function $\hat{f}(x): D \to \Re$ for $D \subset \Re$

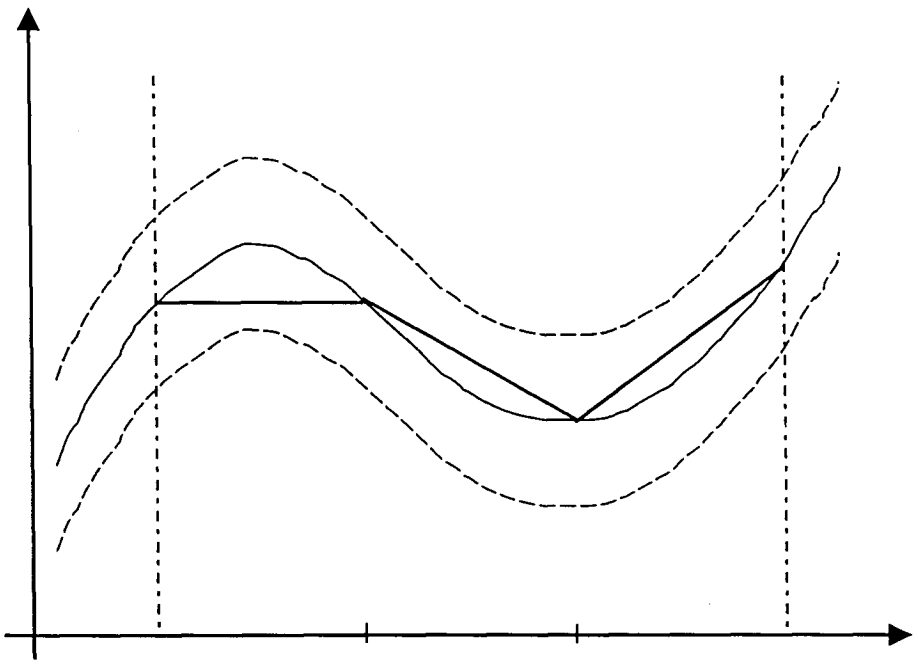


Fig. 4.2. Approximating a continuous function with a piecewise linear function

may be called as a **piecewise linear function** [Bar64] on D if D can be divided into a finite number of disjoint sub-intervals, denoted $I_1, I_2, \cdots, I_m$ such that $\hat{f}$ is linear on each $I_k, k = 1,2,\cdots,m$.

Figure 4.2 shows an example of the piecewise linear approximation of a continuous function within a predefined domain of the independent variable.

4.2.3 Approximation using Basis functions

One technique of function approximation is to choose a class of approximators whose elements can be expressed in terms of a finite basis. Provided there exists a finite basis $\{g_j(.)\}_{j=1}^{m}$, the estimate $\hat{f}$ is expressed in

terms of the basis functions. An intuitive approach is to express the estimate $\hat{f}$ as a linear combination of the basis functions as given below.

$$\hat{f}(x) = b_0 + \sum_{j=1}^{m} b_j g_j(x) \tag{4.1}$$

where b_0 is the constant term, and $b_1, b_2, \cdots, b_m$ are the weightings associated with the basis functions. Three major techniques are often used to choose the basis function, viz.

a) choosing a large fixed basis to approximate any desired function,
b) tuning the parameters of a smaller adaptive basis to obtain an optimal approximation, and
c) adding new basis functions as data points arrive.

By adopting the technique (a), the basis function coefficients, viz. b_i can be obtained by least-squares minimization techniques. However, due to the "curse of dimensionality", this approach has the disadvantage that with an increase in the dimensionality of the data, the required number of basis functions increases exponentially. Method (b) involves the use of the derivative based gradient-descent algorithms, which often require prohibitive computation times, are prone to convergence to sub-optimal solutions, and suffer from forgetting problems. Approach (c) works irrespective of the dimensionality of the input space, but has prohibitively large memory requirements as the number of data points increases. Most of the different approximation techniques using basis functions use one

method or the other, but an ideal approach may be a mix of the above techniques.

It has been well established that multi-layer feed-forward neural networks with a variety of activation functions can act as universal function approximators [Hor89, Bos98]. Hornik et al. [Hor89] used the Stone-Weirstrass theorem to prove that a two layer feed-forward network can approximate a continuous real function on a compact subset of $\Re^n$ arbitrarily closely in terms of the L_∞-norm, while Cybenko [Cyb89] showed that any absolutely integrable function can be uniformly approximated by a neural network having only one hidden layer and employing continuous nonlinearities. In actual application, neural networks often combine all the above mentioned techniques to choose basis functions in their training schemes. The most popular back-propagation algorithm and its variants are examples of the second technique, while radial basis function approaches can incorporate all three techniques. CART [Bre84] uses an expansion into indicator functions. Breiman [Bre93] showed a method of fitting hinge functions to data for regression problems. A technique for approximating a function by a single hidden layer network was proposed by Hush et al. [Hus98], in which basis functions, implemented as piecewise-linear sigmoidal nodes, are added using the well-known method of fitting the residual. Esposito et al. [Esp2K] proposed a method for approximating continuous and discontinuous mappings by a growing neural network having Radial Basis

Functions (RBF) as hidden nodes, in which the variances of the nodes are learnt by an evolutionary optimization strategy.

An important consideration in a neural network application is the choice of network structure. Most adaptation laws presume a neural network with a fixed structure, the choice of which has to be done a priori. In practice, the optimal number of neurons and the architecture cannot be decided in advance. This can often lead to underfitting or overfitting of the training data. Therefore, it is necessary to choose adaptive structure neural networks depending on the application. Sanger [San91] proposed a tree-structured adaptive network for function approximation with the hope that for most of the data, only a few dimensions of the input may be necessary to compute the desired output function. Additional dimensions are incorporated only when needed. Choi et al. [Cho92] introduced a new method of constructing a three layer neural network with sigmoidal activation functions, that approximates any continuous function of one variable. Choi et al. [Cho94] proposed a constructive neural network that is devised by introducing a space tessellation, i.e. a covering of the Euclidean space by nonoverlapping hyperpolyhedral convex cells. It is capable of approximating arbitrary continuous functions with a piecewise linear or nonlinear local interpolation. A novel constructive algorithm, named as Iterative Incremental Function Approximation (IIFA), was proposed by Draelos et al. [Dra96], that is capable of approximating one and two-dimensional functions. Eppler et al.

[Epp99] proposed a piecewise linear network (PLN) for function approximation that divides the input space into several linear regions. It is a single hidden layer network, where each hidden neuron represents a linear region, and evaluates the distance from the input vector to the center point of a linear region, that is represented by the weight vector of the hidden neuron.

Apart from the choice of neural network architecture, another aspect of importance when developing a neural network is the choice of the training strategy. Section 1.4 discusses the problems associated with conventional derivative-based neural network weight updating procedures, that employ iterative schemes in which the incremental change in weight is dependent on the output error. It also discusses alternative techniques of neural network training, that use linear programming formulations of the learning problem to determine network weights. The simplicity of the LP formulation and its guaranteed convergence have increased the potential of LP based approaches for training neural networks.

The algorithm presented in this chapter builds a neural network one neuron at a time. Neurons with linear threshold transfer characteristics are added until the approximation error is below a specified tolerance $\pm\Delta$ at all sample points. The weights of each neuron are determined by solving a simple linear programming problem. In comparison to an approach such as back-

propagation, the user does not have to choose the number of neurons or layers, the learning rate, momentum term, or the initial weight values. The approach thus seeks to explore an alternative to the associated difficulties of training multilayer neural networks.

The chapter is organized as follows. Section 4.3 discusses the function approximation problem, and how it can be posed in a linear programming framework. Section 4.4 describes the algorithm developed to learn a neural network for the function approximation problem, as well as a technique to accelerate learning by removing redundant samples. Section 4.5 contains experimental results. Section 4.6 introduces a new approach termed as Potential Proximal Support Vector Regression (PPSVR) for multidimensional function approximation, that is computationally simpler compared to conventional Support Vector Regression. Section 4.7 is devoted to concluding remarks.

4.3 LP Formulation of Function Approximation Problem

Given a set of input-output data pairs $S_t = \{(x^i, y^i) | x^i, y^i \in \Re; i = 1,2,\cdots,N\}$, where $y^i = f(x^i)$, find an approximating function $\hat{f}$, such that the absolute value of the residual error computed at each of the data points is bounded from above by Δ. Expressed mathematically, if $\hat{y} = \hat{f}(x)$ learns such a data set, the desired satisfiability conditions are $|y^i - \hat{f}(x^i)| \leq \Delta$, i.e. $-\Delta \leq y^i - \hat{f}(x^i) \leq \Delta$. In this work,

we investigate the possibility of learning a piecewise-linear (PWL) type approximation. Without loss of generality, we assume that the function to be learnt is always positive, i.e. the values of y^i are positive. It is always possible to add a suitable offset to any function that does not satisfy this requirement and to then apply the proposed method [Jay2K2a].

It is initially assumed that a single linear segment can fit the given data. Such a segment corresponds to a single neuron. The equation of such a line segment is of the form

$$y = w_0 + w_1 x . \tag{4.2}$$

In order for this segment to approximate the given data set within the above-mentioned specified bound of Δ, it is required that

$$w_0 + w_1 x^i \geq y^i - \Delta, \;\; i = 1, 2, \cdots, N \tag{4.3a}$$

$$w_0 + w_1 x^i \leq y^i + \Delta, \;\; i = 1, 2, \cdots, N . \tag{4.3b}$$

Introducing non-negative slack and surplus variables $s_{i1}^{(+)}$, $s_{i1}^{(-)}$, $s_{i2}^{(+)}$, and $s_{i2}^{(-)}$, inequalities (4.3a-4.3b) can be rewritten as the following equalities.

$$\begin{aligned} w_0 + w_1 x^i &= y^i - \Delta + s_{i1}^{(+)} - s_{i1}^{(-)} \\ w_0 + w_1 x^i &= y^i + \Delta - s_{i2}^{(+)} + s_{i2}^{(-)}; \;\; s_{ij} \geq 0, \;\; i = 1, 2, \cdots, N, \;\; j = 1, 2. \end{aligned} \tag{4.4}$$

Inequalities (4.3a-4.3b) are satisfied if

$$s_{i1}^{(-)} = s_{i2}^{(-)} = 0; \quad i = 1, 2, \cdots, N. \tag{4.5}$$

Therefore, given a set of input-output data pairs, the obvious objective is to minimize the number of inequalities that are not satisfied. Expressing w_0 and w_1 in terms of non-negative variables as $w_k = \left(w_k^{(+)} - w_k^{(-)}\right), \; k = 0, 1$ as in (2.3), we can formulate (4.4) and (4.5) as the following linear programming problem (LPP).

$$\text{Minimize} \quad \sum_{i=1}^{N} s_{i1}^{(-)} + s_{i2}^{(-)} \tag{4.6}$$

subject to the following constraints

$$\begin{aligned} (w_0^{(+)} - w_0^{(-)}) + (w_1^{(+)} - w_1^{(-)})x^i &= y^i - \Delta + s_{i1}^{(+)} - s_{i1}^{(-)} \\ (w_0^{(+)} - w_0^{(-)}) + (w_1^{(+)} - w_1^{(-)})x^i &= y^i + \Delta - s_{i2}^{(+)} + s_{i2}^{(-)}, \quad i = 1, 2, \cdots, N. \end{aligned} \tag{4.7}$$

As mentioned earlier in this section, if a single linear segment provides a satisfactory approximation, then the value of the objective function given by (4.6) would be zero.

4.4 Building the Neural Network

4.4.1 Variants of Linear and Piecewise Linear Activation functions

The objective is to build the function approximating neural network using piecewise linear segments. Using MATLAB [MAT] terminology, different possible

neural network activation functions having linear or partly linear characteristics are described in Sections 4.4.1.1 to 4.4.1.4.

4.4.1.1 "purelin"

This is a purely linear activation function with unity slope and passing through the origin. Its transfer characteristic, given by (4.8) is shown in Fig. 4.3(a).

$$\text{purelin}(net) = net\,. \tag{4.8}$$

4.4.1.2 "satlins"

This is the symmetric saturating linear activation function. It is a modification of "purelin" with outputs saturating at ±1, and having a transfer characteristics given by (4.9). Figure 4.3(b) shows a sketch of the function.

$$\text{satlins}(net) = \begin{cases} -1\ , \text{if } net \le -1 \\ net, \text{if } -1 < net < 1\ . \\ 1\ , \text{if } net \ge 1. \end{cases} \tag{4.9}$$

4.4.1.3 "poslin"

This is the positive linear transfer function, also known as the "linear-threshold"-type activation function. It is a modification of "purelin" with non-zero output for positive values of the input argument and zero output elsewhere. Its transfer characteristic is given by (4.10) and is shown sketched in Fig. 4.3(c).

$$\text{poslin}(net) = \max(0, net). \tag{4.10}$$

4.4.1.4 "satlin"

This is the saturating linear activation function. It is a modification of "satlins" with a non-zero value for positive values of the input argument, and having a

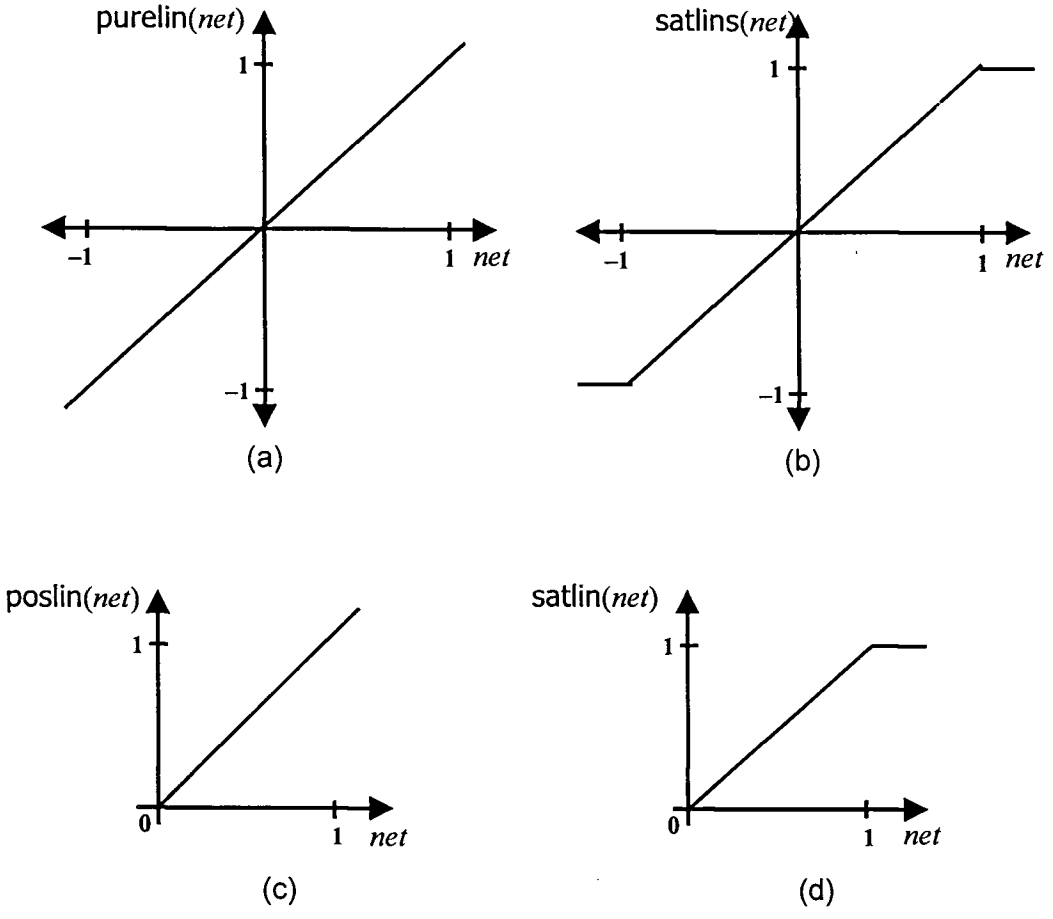


Fig. 4.3 Linear Activation functions: (a) purelin, (b) satlins, (c) poslin, and (d) satlin.

zero value elsewhere. Its transfer characteristic is given by (4.11). Figure 4.3(d) shows a plot of the characteristic.

$$\text{satlin}(net) = \begin{cases} 0 & \text{, if } net \le 0 \\ net, & \text{if } 0 < net \le 1 \\ 1 & \text{, if } net > 1 \end{cases} . \tag{4.11}$$

4.4.2 Network Construction

The neural network is grown one neuron at a time, with each neuron corresponding to a piecewise linear segment. It is important to note, however,

that the neurons are not combined in a simple linear manner. In this work, use is made of neurons with poslin-type characteristics, as shown in Fig. 4.3(c). The net input to the neuron is given by

$$net = w_0 + w_1 x, \tag{4.12}$$

and the output is given by

$$z = \max(0, net). \tag{4.13}$$

Inspired by its biological analogue, this type of neuron, whose output is a rate coded firing frequency and confined only to the positive real axis, has been used and analyzed widely in the literature [Ben95, Hah98, Wer2K1].

The LPP (4.6-4.7) is solved to obtain the best linear fit to the given data. This also yields the weights w_0 and w_1 of the first neuron. The output of the neuron yields the approximated value of the function. In the solution to the LPP, if the value of the objective function (4.6) is zero, then a single linear segment would be sufficient to achieve the desired approximation and no more neurons (PWL segments) are needed. On the other hand, if the approximation error exceeds $\pm\Delta$ at any sample, then the value of the objective function will be non-zero and some of the variables $s_{i1}^{(-)}, s_{i2}^{(-)}$ $(i = 1,2,\cdots,N)$ will have positive values. The proposed algorithm adds PWL segments to correct for the approximation error at such points. Each PWL segment corresponds to a neuron in the network. However, the segments, as represented by the neurons, do not add up in a simple linear manner, i.e. the overall characteristic is not obtained by a simple

summation of the individual segments. We now discuss the case when a single linear segment does not yield a satisfactory approximation.

Let z^i denote the output of the first neuron corresponding to the i-th input sample, viz. x^i, as given by (4.10). The approximation error corresponding to x^i is given by

$$e^i = y^i - z^i, \tag{4.14}$$

where y^i is the specified value of the function at the sample x^i. The samples $x^i(i = 1,2,\cdots,N)$ are classified into four classes C_1, C_2, C_3, and C_4 based on the rules given in Table 4.1. Classes C_1, C_2, C_3, and C_4 contain N_1, N_2, N_3, and N_4 samples, respectively. Note that $N_1 + N_2 + N_3 + N_4 = N$.

Table 4.1: Classification of the Samples Based on the Error

Class	Error	Number of Samples
C_1	$0 < e^i \leq \Delta$	N_1
C_2	$-\Delta \leq e^i \leq 0$	N_2
C_3	$e^i > \Delta$	N_3
C_4	$e^i < -\Delta$	N_4

Depending on the error, two child neurons A and B are added to the parent neuron as shown in Fig. 4.4, to take care of inputs belonging to classes C_3 and C_4, respectively. For a pattern x^i belonging to class C_3, the error is positive, i.e. $y^i > z^i$. Neuron A contributes a positive input for such samples, thereby increasing the net input, which increases z^i, thus reducing e^i. Similarly, neuron B adds a negative input for samples belonging to class C_4. For samples in this

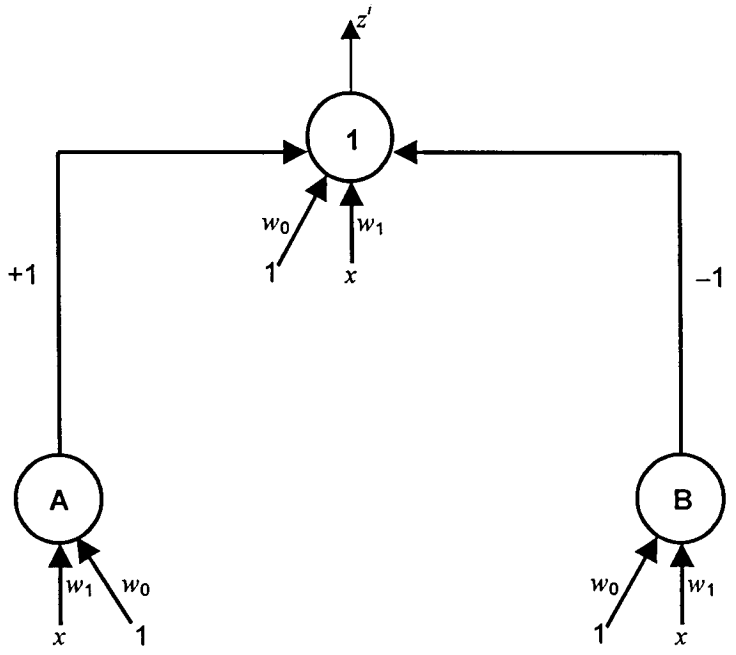


Fig. 4.4. Vertex Neuron 1 and its child neurons A and B.

class, the error e^i is negative, i.e. $y^i < z^i$. Again, the input from B reduces the output z^i of the parent neuron, thus reducing e^i. However, one needs to ensure that in this process, other patterns are not disturbed. This implies that the output of neuron A should be zero for samples belonging to classes C_1, C_2, and C_4, while neuron B should give a zero output for samples belonging to classes C_1, C_2, and C_3. The required outputs for neurons A and B to the parent neuron are summarized in Table 4.2. While a negative input is required from neuron B, this is not directly possible since the transfer characteristics of the neurons precludes negative outputs. Therefore, the weight connecting neuron B to its parent is chosen to be negative, and has been made equal to –1. The outputs of neuron B are thus also required to be non-negative as indicated in Table 4.2.

Table 4.2: Required outputs of A and B neurons connected to the vertex neuron

Class	A	B
C_1	0	0
C_2	0	0
C_3	$y^i - (w_0 + w_1 x^i)$	0
C_4	0	$(w_0 + w_1 x^i) - y^i$

The inequalities to be satisfied by A type neurons of the second layer are given by

$$\begin{aligned} w_0 + w_1 x^i &\le -\Delta, && x^i \in C_1, C_2, C_4 \\ w_0 + w_1 x^i &\le y_A^i + \Delta, && x^i \in C_3 \\ w_0 + w_1 x^i &\ge y_A^i, && x^i \in C_3 \end{aligned} \quad , \tag{4.15}$$

where

$$y_A^i = y^i - (w_0 + w_1 x^i), \; x^i \in C_3 . \tag{4.16}$$

Similarly, for neuron B, the inequalities to be satisfied are

$$\begin{aligned} w_0 + w_1 x^i &\le -\Delta, && x^i \in C_1, C_2, C_3 \\ w_0 + w_1 x^i &\ge y_B^i, && x^i \in C_4 \\ w_0 + w_1 x^i &\le y_B^i + \Delta, && x^i \in C_4 \end{aligned} \quad , \tag{4.17}$$

where

$$y_B^i = (w_0 + w_1 x^i) - y^i, \; x^i \in C_4 . \tag{4.18}$$

The weights of neuron A are determined by solving a similar LPP; the objective function is given by (4.6), with constraints (4.15) replacing those in (4.2) and (4.3). The weights of neuron B are obtained by solving the LPP defined by (4.6) subject to the constraints in (4.17).

It is possible that the approximation error is still not within the specified bounds for some of the input samples. Again, we segregate the input samples into classes based on the approximation error as summarized in Table 4.1. If the sets C_3 and C_4 are not null for either A or B, then we again add neurons in order to correct the error. Note that neurons A and B, which are connected to the parent neuron, are in the second layer. We refer to the child neurons being connected to neurons A and B as *third layer* neurons.

The required output of neurons in the third and subsequent layers is specified in Table 4.3. The change from Table 4.2 arises because once the $\pm\Delta$ condition is applied to the vertex neuron, and the training set for second layer neurons is determined following Table 4.2, the training set for neurons in the third and subsequent layers has to be fixed after considering the residue due to samples belonging to classes C_1 and C_2 as well. Note that the contribution of B-type neurons in the third and subsequent layers is negative for samples belonging to classes C_2 and C_4. This is ensured by connecting such neurons to their parents through a negative weight equal to –1.

Table 4.3: Desired output of A and B type neurons in layers 3 and beyond

Class	A	B
C_1	$y^i_{parent} - z^i_{parent}$	0
C_2	0	$z^i_{parent} - y^i_{parent}$
C_3	$y^i_{parent} - z^i_{parent}$	0
C_4	0	$z^i_{parent} - y^i_{parent}$

Here, y^i_{parent} and z^i_{parent} refer to the required and actual outputs of the parent neuron (of A, or B as the case may be) for the i-th input sample. In Table 4.3, though the desired output for samples belonging to the class C_1, C_3 and C_2, C_4 have identical representations, they have different magnitudes.

Considering a child neuron of type A, for the i-th sample at the parent, the magnitudes of the desired output at the child are as follows:

i) if $i \in C_1$, then $y^i_{parent} - z^i_{parent} \leq \Delta$, and

ii) if $i \in C_3$, then $y^i_{parent} - z^i_{parent} > \Delta$.

Similarly considering a child neuron of type B, for the i-th sample at the parent, the magnitudes of the desired output at the child are as follows:

i) if $i \in C_2$, then $z^i_{parent} - y^i_{parent} \leq \Delta$, and

ii) if $i \in C_4$, then $z^i_{parent} - y^i_{parent} > \Delta$.

The set of inequalities to be satisfied by a type-A neuron in the third and subsequent layers is given by

$$\begin{aligned} w_0 + w_1 x^i &\leq -\Delta, && x^i \in C_2, C_4 \\ w_0 + w_1 x^i &\leq y^i_A + \Delta, && x^i \in C_1, C_3, \\ w_0 + w_1 x^i &\geq y^i_A, && x^i \in C_1, C_3 \end{aligned} \tag{4.19}$$

where

$$y^i_A = y^i_{parent} - z^i_{parent}. \tag{4.20}$$

Similarly, for type-B neurons in the third and subsequent layers, the following constraints apply

$$
\begin{aligned}
&w_0 + w_1 x^i \le -\Delta, && x^i \in C_1, C_3 \\
&w_0 + w_1 x^i \le y_B^i + \Delta, && x^i \in C_2, C_4, \\
&w_0 + w_1 x^i \ge y_B^i, && x^i \in C_2, C_4
\end{aligned}
\tag{4.21}
$$

where

$$
y_B^i = z_{parent}^i - y_{parent}^i . \tag{4.22}
$$

The network is further grown by solving the LPPs defined by the objective function (4.6) subject to the appropriate constraints, viz. (4.19) for type-A neurons and (4.21) for type-B neurons. The growth of a branch ceases when all the input samples are satisfied at a particular node.

Example: Consider the sample set of (x, y)-tuples given by {(1,2.8415), (1.6,2.9996), (3,2.1411), (4.7,1), (6.5,2.2151), (8,2.9893), (9,2.4121), (10,1.4559)} to be learnt within bounds of $\Delta = \pm 0.05$. The data set was learnt, following section 4.4.2, by a constructive network having 22 linear threshold-type neurons. The training set at neuron 5 in layer 3 was as follows:

x	1	1.6	3	4.7	6.5	8	9	10
y	0	0	0	0.31468	0	0	0	0

Following the proposed approach, the above data set was approximated by the following linear segment

$$z = -0.77413 + 0.24137x$$

Using the training set, the output (z) of a neuron of the linear threshold-type (Fig. 4.3c), its corresponding error (e), and the desired output (y) are given in the following tables.

x	1	1.6	3	4.7	6.5	8	9	10
y	0	0	0	0.31468	0	0	0	0
z	0	0	0	0.36031	0.7947	1.1568	1.3982	1.6395
$e=y-z$	0	0	0	−0.04563	−0.7947	−1.1568	−1.3982	−1.6395
Class	C_1	C_1	C_1	C_2	C_4	C_4	C_4	C_4

The residue at the sample points can be corrected by adding a child neuron of type-B. Though the residue for the sample belonging to class C_2 is within the bound of $\pm \Delta$, from the layer 3 onwards such residues appear in the training set for its child along with the residue for samples belonging C_4. Hence the training set of the type-B child neuron is given by

x	1	1.6	3	4.7	6.5	8	9	10
y	0	0	0	0.04563	0.7947	1.1568	1.3982	1.6395

If any child neuron of type-A needs to be added, from layer 3 onwards, the same procedure decides its training set. For these we consider non-zero residues of samples belonging to classes C_1 and C_3 at the parent.

Thus in Table 4.3, the residues for samples belonging to class C_1 and C_2 are within bounds of $\pm \Delta$, though they have been represented identically to samples of classes C_3 and C_4.

4.4.3 Avoiding Trivial Solutions

Using the objective function in (4.6) may sometimes yield a trivial solution independent of the inputs. Such a null solution is obtained, for example, when the system of equalities is degenerate [Ben92]. To overcome this problem we use the following objective function

$$\text{Minimize} \quad \frac{1}{m}\sum_{i=1}^{m} s_i^{(-)} + \frac{1}{k}\sum_{j=1}^{k} s_j^{(-)}, \tag{4.23}$$

where m and k are the numbers of less than type and greater than type inequalities, respectively. The objective function in (4.23) is similar to (2.8) of the LP formulation of the binary classification problem that is based on the one used in [Ben92] for pattern classification.

4.4.4 Removing Redundant Samples

As the network grows, the required output for neurons at many of the input points becomes zero. This can be used to reduce the number of inequalities and hence the size of the LPP at a given node. Consider the case shown in Fig. 4.5(a). A non-zero output is required at datapoints 5, and 10, while the desired output is zero at all other datapoints. We see that hyperplanes H_1 and H_2 are sufficient to obtain the required non-zero output at datapoint 5 while ensuring a zero output at other datapoints. Hyperplanes H_1 and H_2 are drawn by using only the datapoints with required zero output (i.e. datapoints 4 and 6), which are just adjacent to the datapoint with a desired non-zero output (i.e. at datapoint 5). Recall from Fig. 4.3(c) and (4.10) that for a linear threshold neuron to have a zero output, the input needs to be any non-positive value. A similar situation arises at other points, e.g. at datapoint 10. Thus, the sample set for a neuron can be reduced, in some cases significantly, as shown in Fig. 4.5(b). This

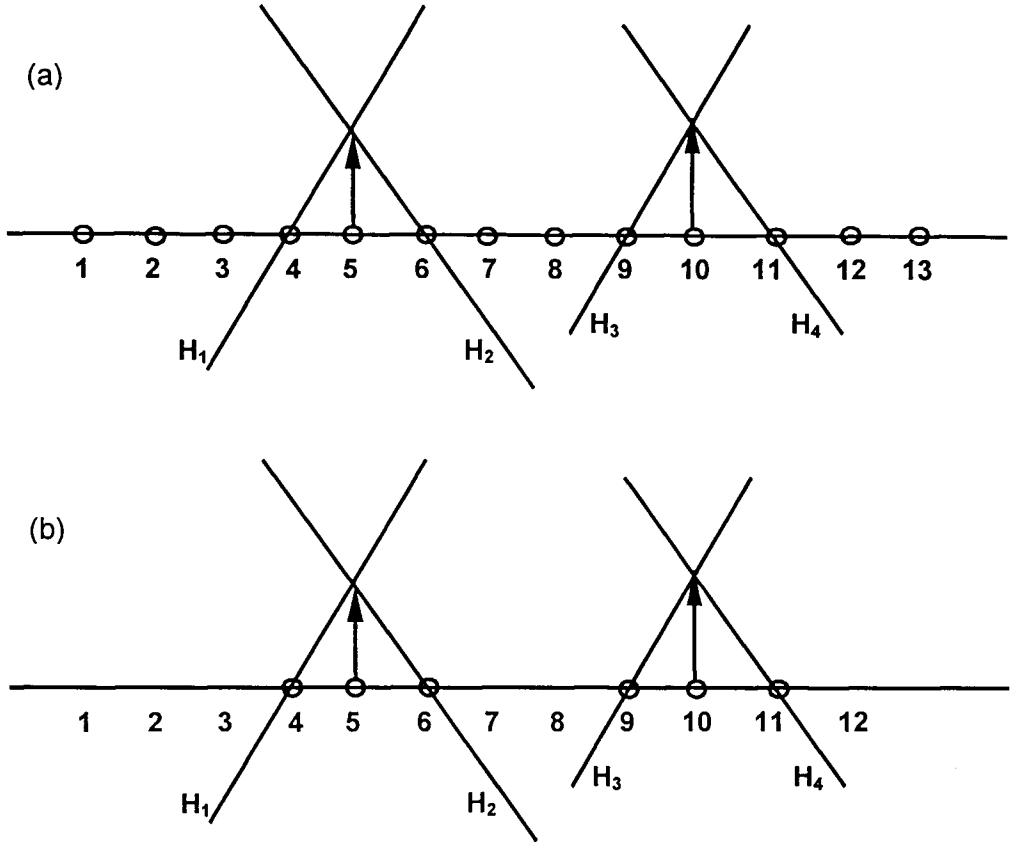


Fig. 4.5. Removal of Redundant Samples.

reduces the number of inequalities, and hence the size of the LPP and the computation time.

4.5 Experimental Results

The algorithm was implemented in MATLAB running on a 800 MHz Pentium-III computer. Eight samples of the function $y = 2 + \sin(x)$ were chosen for the learning task. The sample set of (x, y)-tuples is given by {(1,2.8415), (1.6,2.9996), (3,2.1411), (4.7,1), (6.5,2.2151), (8,2.9893), (9,2.4121), (10,1.4559)}.

For a chosen Δ, the algorithm builds a tree structured network, with neurons being added one-by-one until the network output at all sampling points lies within a margin ±Δ of the specified value. The performance of the algorithm was compared with that of a multi-layer backpropagation network (MLBN). The latter was also simulated in MATLAB by using the Neural Network Toolbox [MAT]. An important choice that needs to be made is with regard to the activation function of the neurons. Since the proposed network uses PWL neurons, the logical choice of the activation function for the MLBN should be one of "poslin", "satlin", or "satlins" functions, which are available in MATLAB. These are shown in Fig. 4.3. Of these, the "poslin" activation function is identical to the PWL characteristic used in the proposed algorithm.

In the case of the MLBN, the sum-squared error (SSE) was used to determine the convergence criterion. For a given Δ used with the proposed algorithm, the MLBN training was ceased when the SSE fell below the limit $SSE_{lim} = N\,\Delta^2$, where N denotes the number of sampling points. This is a conservative criterion, assuming that the error at each sampling point is allowed to be Δ. The MLBN outputs must be in the range of –1 to 1 (0 to 1) for a "satlins" ("poslin", or "satlin") activation function. Therefore, the samples were appropriately shifted by a constant value using the same values of x as were used for the proposed algorithm.

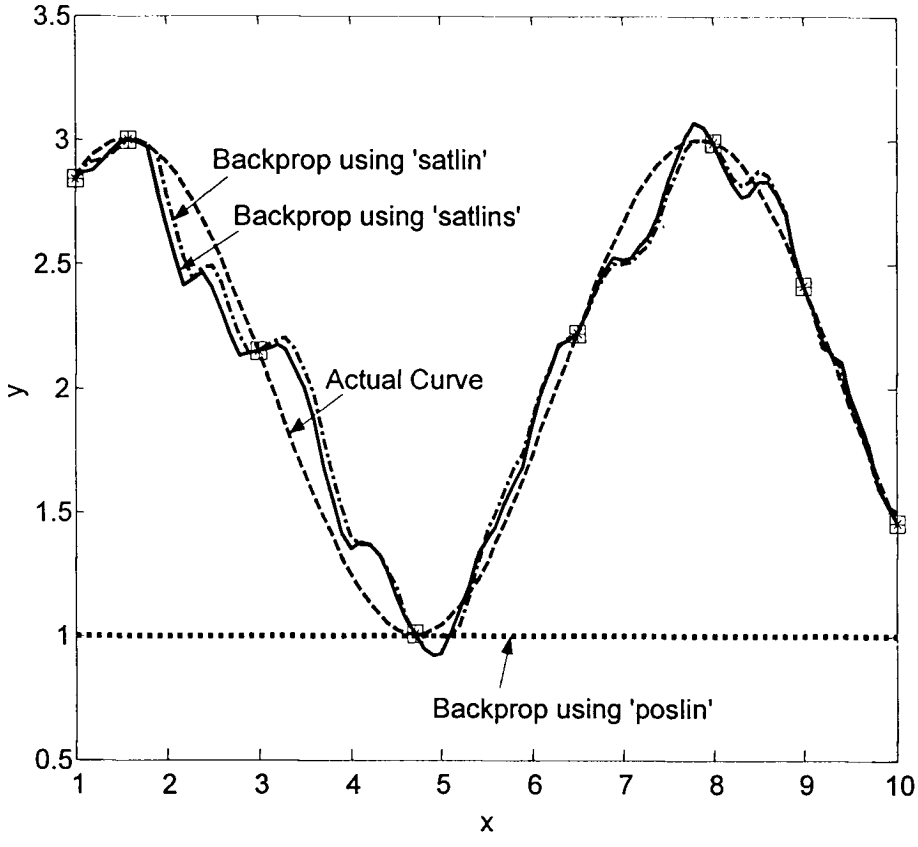


Fig. 4.6. Approximation of a sampling points on y=sin(x) (shifted), by the NLBN with 45 neurons in one hidden layer. The three approximating curves correspond to the multilayer networks using the "poslin", "satlin" and "satlins" activation functions

In order to compare the best result obtained by MLBN, it is necessary to determine the optimal number of hidden layer neurons, i.e. the MLBN for which the SSE is a minimum. The convergence criterion was chosen to be: $SSE < SSE_{lim}=0.0008$. The SSE was found to be a minimum for a network with the "satlins" activation function, having 45 hidden neurons, which took 1.75 seconds for learning. Of course, this does not imply that good solutions cannot be found for other choices of the number of hidden neurons, but reinforces the

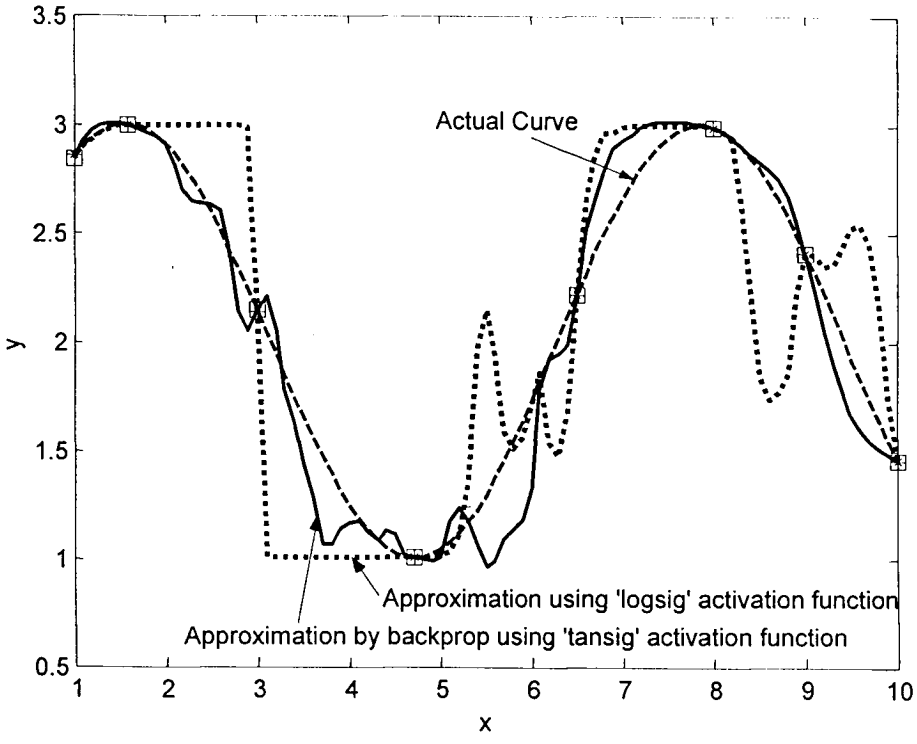


Fig. 4.7. Results for MLBNs using the "tansig" and "logsig" activation functions for Δ=0.01 with 8 sampling points. The networks require 32 and 49 neurons, respectively, in one hidden layer. Sample points are indicated by squares.

empiricism inherent in any practical application of the backpropagation algorithm. The performance of the backpropagation algorithm also depends on the initialization of weights, and can vary from one trial to another. The objective of the tree-structured architecture was to learn the samples within a tolerance band of $\pm\Delta$ (= 0.01).

Other possible activation functions include the logistic or sigmoid (Fig. 1.1a), and the tanh (Fig. 1.1b) function. These are termed in MATLAB as "logsig" and "tansig", respectively. Figures 4.6 and 4.7 show the results obtained by MLBNs using the corresponding activation functions. The results of Figs. 4.6 and 4.7 show that MLBNs tend to add spurious extremal points, i.e. minima, maxima,

and saddle points. This is because conventional backpropagation does not yield a piecewise smooth interpolation. Errors at individual samples cannot be bounded easily - pathological cases where the SSE is small, but error at some samples is large, are possible. Figure 4.8 shows a plot of the log(SSE) versus the number of hidden units, for three MLBNs with different activation functions. The figure shows how crucial the choice of the number of hidden neurons can be. Note that small changes in this number around local minima can lead to very large changes in SSE. The proposed algorithm learnt the given samples in 1.36s by using a network of 22 neurons with a SSE of 0.0002. Figure 4.9 shows the performance of the proposed algorithm for Δ=0.01. Figure 4.10 shows the neural network that was built by the proposed algorithm for Δ=0.4. In order to study how the approximating network scales with the specified tolerance, the sampling points were approximated over a range of values of Δ. The results are summarized in Table 4.4.

Table 4.4: Effect of Scaling Δ on network growth

Δ	No. of Neurons	Error at sample							
		1	2	3	4	5	6	7	8
1e-3	22	0.001	0	0	0	–0.001	0	0	0
0.01	22	0.01	0	0	0	–0.01	0	0	0
0.05	22	0.05	0	0	0	–0.05	0	0	0
0.1	22	0.1	0	0	0	–0.1	0	0	0
0.2	22	0.2	0	0	0	–0.131	0	0.2	0
0.3	17	0.2907	0	–0.3	0	–0.034	0	0.3	0
0.4	9	0.1942	0.4	–0.347	0	0.0045	0	0.4	0

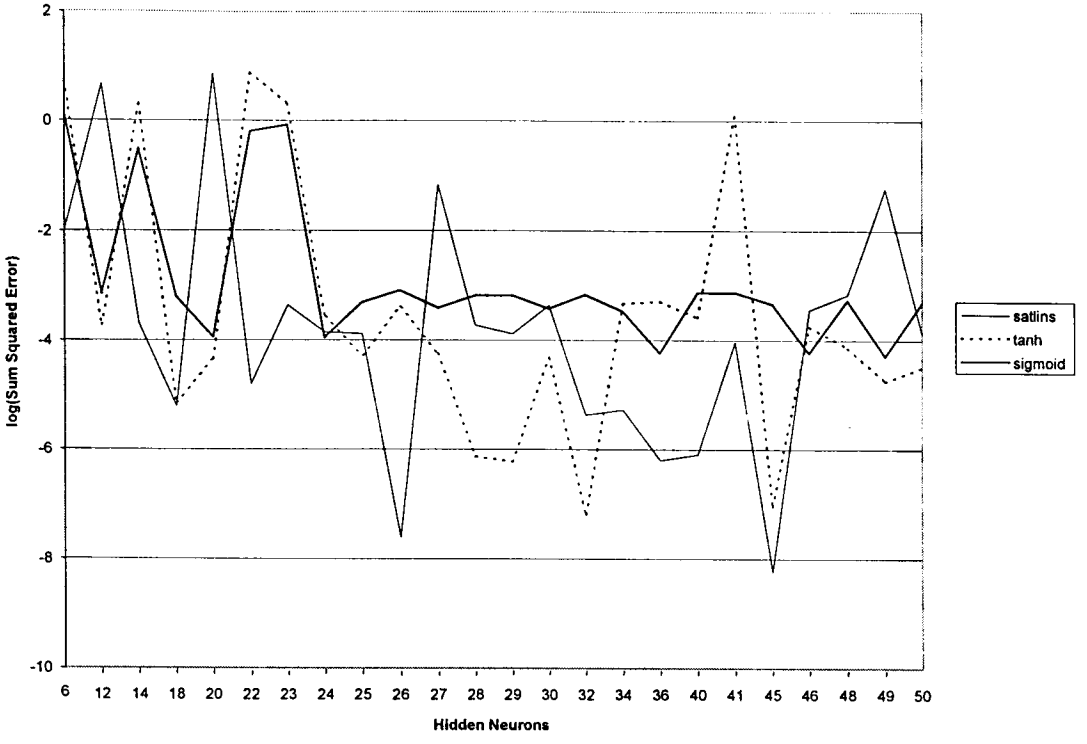


Fig. 4.8 Plot of log (SSE) versus the number of hidden neurons

The proposed algorithm generates an approximation that depends on the sample points as well as the tolerance Δ. A smaller tolerance generally yields an approximation with a larger number of PWL segments, or equivalently, a neural network with more neurons. However, the relationship between Δ and the network size is not a linear one; note that there is no increase in size when Δ is reduced from 0.1 to 10^{-3}. Also, note that the error at the training points is within the specified tolerance, for each sample and for each value of Δ. The number of neurons is larger than the number of segments constituting the approximated curve, but this is because they are not combined in a simple linear manner. At first glance, it appears logical to extend the proposed approach to functions of multiple variables.

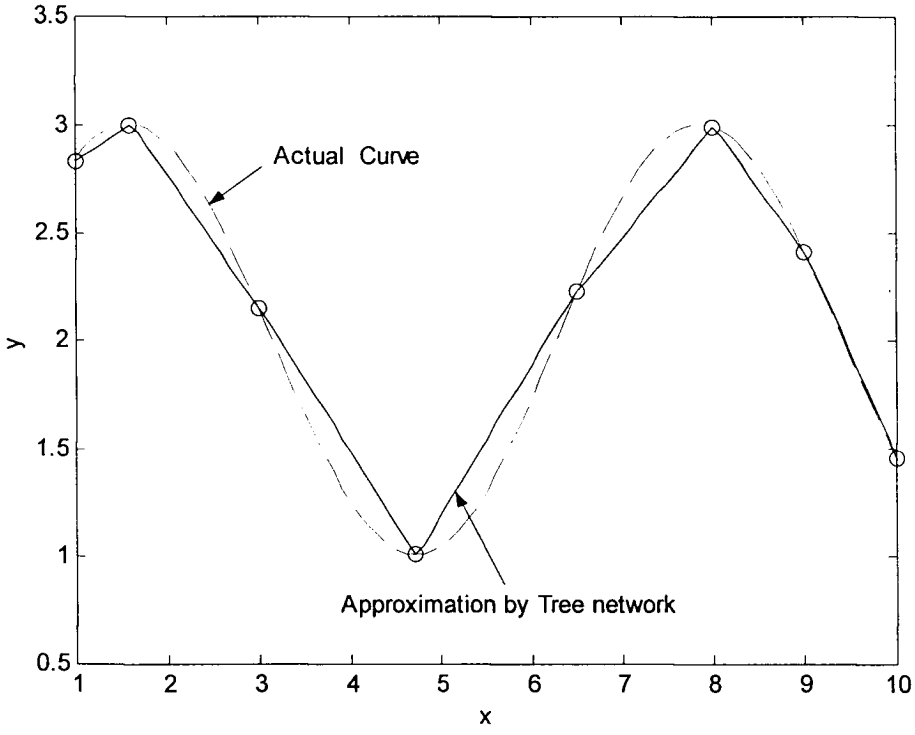


Fig. 4.9. Approximation obtained by a 22 node network at the 8 points sampled from y=2+sin (x), with Δ=0.01, by using the proposed algorithm.

4.6 Function Approximation using Potential Proximal Support Vector Regression (PPSVR)

The approach discussed in Sections 4.3-4.5 of this chapter suggests a new direction for building constructive networks for approximating functions of many variables. However, when extending it to higher dimensions it has been observed that cross interference of the hyperplanes as represented by the neuron disturbs the correctly learnt samples at the parent layer. Therefore alternative strategies were sought to overcome this problem.

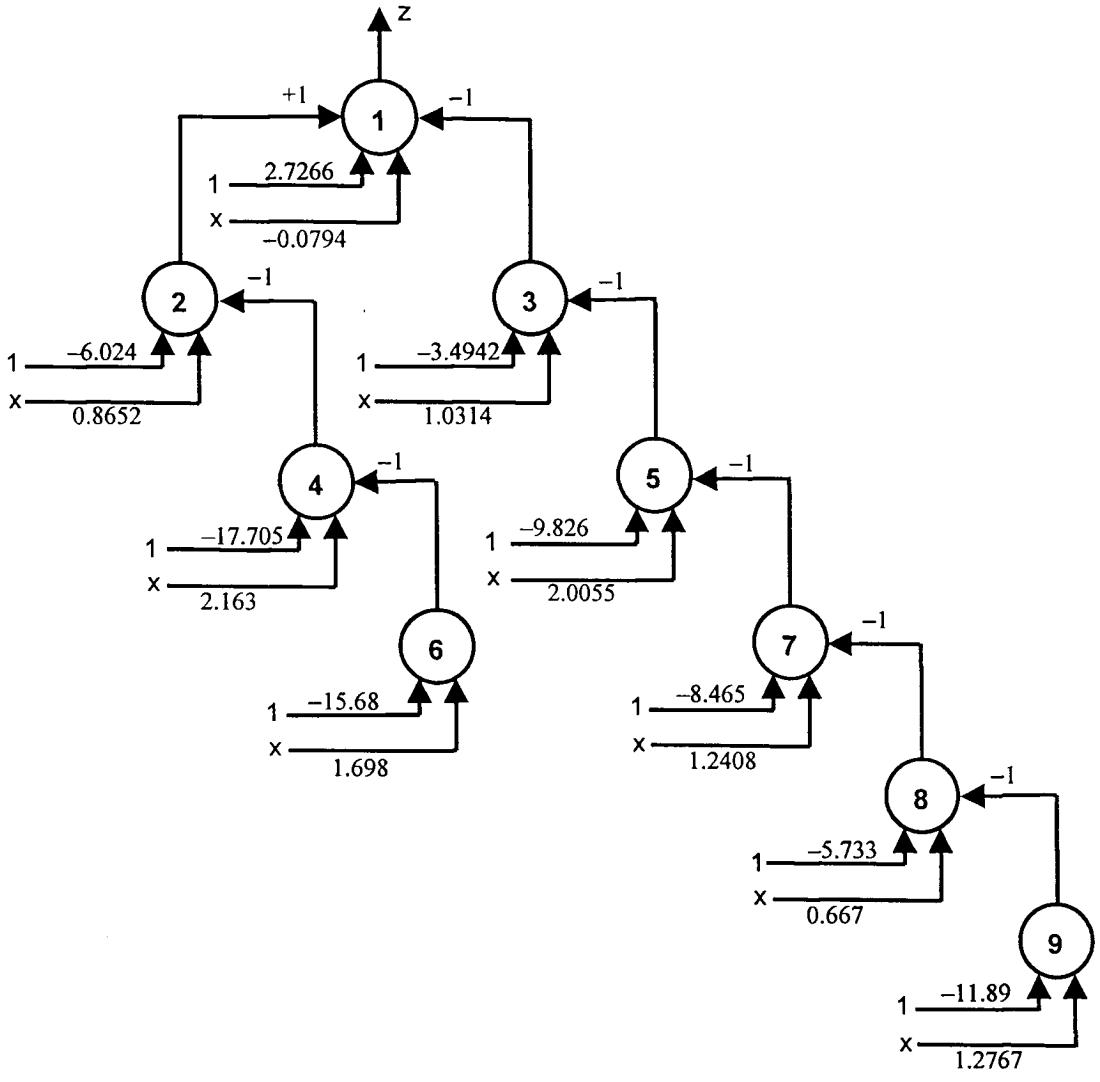


Fig. 4.10. Network built by the proposed algorithm for approximating y=2+sin (x). Δ=0.4. Numbers within circles indicate neuron numbers.

Support Vector Machines have been successfully used in pattern recognition and function estimation problems. Section 1.5 gives an introduction, overview and philosophy of Support Vector Machine Classifiers. Support Vector Regression [Vap98, Vap2K, Cri2K] is an extension of Support Vector Machines for regression problems. Conventional Support Vector Machine formulations, for

classification as well for regression, optimize an objective function subject to certain inequality constraints. The computational cost of solving a conventional support vector classifier or regressor is large because of the need to solve a quadratic programming problem. Section 4.6.1 describes a class of support vector regressors (LS-SVR), where the solution leads to solving a set of linear equations. Section 4.6.2 describes our approach to regression, the Potential Proximal Support Vector Regression (PPSVR). Section 4.6.3 gives experimental verification of the application of PPSVR on artificial and real data sets.

4.6.1 Least Squares Support Vector Regression (LS-SVR)

Saunders et al. [Sau98] and Suykens et al. [Suy99] have introduced a class of support vector machines, called Least Squares Support Vector Machines (LS-SVM) that have equality type constraints in the formulation. This leads to the solution being obtained from a set of linear equations, instead of quadratic programming for classical SVMs. Suykens et al. [Suy2K1] has also applied LS-SVMs to the N-stage optimal control problem.

Given training data $S_t = \{(x^i, y^i); x^i \in \Re^n; y^i \in \Re; i \in I; \text{card}(I) = N\}$. The nonlinear mapping $\phi : \Re^n \to \Re^{n_h}$ maps the input data into a so-called high dimensional feature space $w \in \Re^{n_h}, w_0 \in \Re$. In SVMs, the aim is to reduce the empirical risk

$$R_{emp}(w, w_0) = \frac{1}{N}\sum_{i=1}^{N} \left| y^i - \left\langle w \cdot \phi(x^i) \right\rangle - w_0 \right|_\varepsilon \tag{4.24}$$

where $|a|_\varepsilon$ denotes a loss function that penalizes the loss a depending on the parameter ε. Some commonly used loss functions are linear ε-insensitive loss function, quadratic ε-insensitive loss function, Huber's loss function etc [Vap98, Vap2K, Cri2K].

The risk minimization problem given by (4.24) can be expressed as the following form of regression problem:

$$\underset{w, w_0, \xi}{\text{Minimize}} \quad \frac{1}{2}\langle w \cdot w \rangle + \gamma \frac{1}{2}\sum_{i=1}^{N} \xi_i^2 \tag{4.25}$$

subject to the equality constraints

$$y^i = \left\langle w \cdot \phi(x^i) \right\rangle + w_0 + \xi_i, \qquad i = 1,2,\cdots,N. \tag{4.26}$$

For the above problem, the Lagrangian may be defined as

$$L_{LS}(w, w_0, \xi, \alpha) = \frac{1}{2}\langle w \cdot w \rangle + \gamma \frac{1}{2}\sum_{i=1}^{N} \xi_i^2 - \sum_{i=1}^{N} \alpha_i \left(\left\langle w \cdot \phi(x^i) \right\rangle + w_0 + \xi_i - y^i \right), \tag{4.27}$$

where α_is are the Lagrangian multipliers that can be positive or negative, as follows from the equality constraints of the Kuhn-Tucker conditions.

$$\nabla_w L_{LS} = 0 \;\Rightarrow\; w = \sum_{i=1}^{N} \alpha_i \phi(x^i). \tag{4.28a}$$

$$\frac{\partial L_{LS}}{\partial w_0} = 0 \;\Rightarrow\; \sum_{i=1}^{N} \alpha_i = 0. \tag{4.28b}$$

$$\frac{\partial L_{LS}}{\partial \xi_i} = 0 \Rightarrow \alpha_i = \gamma \xi_i, \quad i = 1,2,\cdots,N. \tag{4.28c}$$

$$\frac{\partial L_{LS}}{\partial \alpha_i} = 0 \Rightarrow \langle w \cdot \phi(x^i) \rangle + w_0 + \xi_i - y^i = 0, \quad i = 1,2,\cdots,N. \tag{4.28d}$$

Substituting w and ξ_i from (4.28a) and (4.28c) in (4.28d) we can write (4.28b) and (4.28d) compactly as

$$\begin{bmatrix} 0 & \vec{1}^T \\ \vec{1} & K + \gamma^{-1} I \end{bmatrix} \begin{bmatrix} w_0 \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix} \tag{4.29}$$

where, $y = [y_1\ y_2 \cdots y_N]^T$, $\vec{1} = [1\,1 \cdots 1]_{1\times N}]^T$, $\alpha = [\alpha_1\ \alpha_2 \cdots \alpha_N]^T$, and $K_{ij} = K(x^i, x^j) = \langle \phi(x^i) \cdot \phi(x^j) \rangle$, $i, j = 1,2,\cdots,N$. Some commonly used kernel functions are shown in Table 4.5.

Table 4.5: Some Kernel functions

Kernel Name	Kernel Function	Parameters
Linear kernel	$K(x^i, x^j) = \langle x^i \cdot x^j \rangle$	
Multilayer Perceptron kernel	$K(x^i, x^j) = \tanh(s\langle x^i \cdot x^j \rangle + t^2)$	t: bias s: scale parameter
Polynomial kernel	$K(x^i, x^j) = (\langle x^i \cdot x^j \rangle + t)^d$	t: intercept d: degree of the polynomial
Gaussian kernel	$K(x^i, x^j) = e^{-\frac{\langle (x^i - x^j)(x^i - x^j) \rangle}{\sigma^2}}$	σ^2: variance

As is evident from (4.28c), for the least squares case, the Lagrangian variables α_i are proportional to the errors at the data, contrary to classical SVMs where non-zero α_i's correspond to the support vectors. Hence, sparseness is lost in the least squares case. For a chosen kernel matrix K and the regularization

parameter γ, (4.29) represents a set of linear equations that can easily be solved for w_0 and α. Here lies the advantage of using LS-SVM where solving a set of linear equations is needed, instead of a quadratic programming problem for classical SVMs from the Kuhn-Tucker conditions. The resulting LS-SVM model for function approximation becomes

$$y(x) = \sum_{i=1}^{N} \alpha_i K(x^i, x) + w_0 . \tag{4.30}$$

Tuning of SVM hyperparameters is essential for minimizing the generalization error of the SVM. Cristianini et al. [Cri98], Chapelle et al. [Cha2K2] and Keerthi [Kee2K2] discuss some techniques for tuning SVM hyperparameters. In LS-SVMs, a judicious choice of the hyperparameters, the regularization parameter γ, and kernel parameters are needed to yield a good function estimate.

In this work, the performance of our proposed approach to regression, termed as PPSVR, has been compared with LS-SVR.

4.6.2 Potential Proximal Support Vector Regression (PPSVR)

Conventional Support Vector Machine formulations, for classification as well as for regression, optimize a quadratic objective function subject to certain inequality constraints. However, they suffer from the lack of invariance under linear transformation. The Potential Support Vector Machine (PSVM) [Hoc2K4] approach has been proposed that uses the covariance matrix of the input in the

objective function and leads to separating hyperplanes that are invariant under a linear transformation of the data.

In this work, we propose a novel approach to regression, termed Potential Proximal Support Vector Regression (PPSVR) for multi-input regression problems, that is computationally far simpler than conventional Support Vector Regression approaches, and requires a single matrix inversion to obtain the separating hyperplane. PPSVR is motivated by PSVM [Hoc2K4] and is in the spirit of least squares SVMs [Sau98, Suy99, Fun2K1].

Given training data $S_t = \{(x^i, y^i),\ x^i \in \Re^n; y^i \in \Re; i \in I; \text{card}(I) = N\}$, the task of a function approximator is to obtain $w \in \Re^n, w_0 \in \Re$, such that

$$E = \sum_{i=1}^{N} \left| \left(\langle w \cdot x^i \rangle + w_0 \right) - y^i \right|. \tag{4.31}$$

is minimized.

Let $X = \begin{bmatrix} x^1 & x^2 & \dots & x^N \end{bmatrix}^T$, $e = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}_{1 \times N}^T$, $y = \begin{bmatrix} y^1 & y^2 & \dots & y^N \end{bmatrix}^T$. Then the regression task can be compactly represented as

$$\text{Minimize} \quad E = \| Xw + ew_0 - y \|_1. \tag{4.32}$$

The residual error at the i-th point is given by

$$r_i = f_{w,w_0}(x^i) - y^i = \langle w.x^i \rangle + w_0 - y^i. \tag{4.33}$$

Considering a quadratic loss function, the mean squared empirical error over all the data points is given by

$$R_{emp}\left[f_{w,w_0}\right]=\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}r_i^2\,. \tag{4.34}$$

The hyperplane defined by w, w_0 minimizes the total residual error. Therefore, we have

$$\nabla_w R_{emp}\left[f_{w,w_0}\right]=\frac{1}{N}X^T\left[Xw+ew_0-y\right]=\mathbf{0} \tag{4.35}$$

and

$$\frac{\partial R_{emp}\left[f_{w,w_0}\right]}{\partial w_0}=\frac{1}{N}\sum_i r_i=w_0+\frac{1}{N}\sum_i\left(\left\langle w.x^i\right\rangle-y^i\right)=0\,. \tag{4.36}$$

Using an objective function that minimizes the possible error q_i at the i-th sample point, and that also minimizes the L_2-norm of the weight vector, the following optimization problem can be formulated.

$$\underset{q,w,w_0}{\text{Minimize}} \quad \frac{\gamma}{2}\left(q^Tq\right)+\frac{1}{2}\left(\left(Xw+ew_0\right)^T\left(Xw+ew_0\right)\right) \tag{4.37}$$

$$\text{such that} \quad \left[X \quad e\right]^T\left(Xw+ew_0-y+q\right)=0\,. \tag{4.38}$$

Here, (4.38) is a modified form of the equality constraints (4.28d) and is motivated by the idea of scale invariance [Hoc2K4].

For the above problem, the Lagrangian may be defined as

$$\begin{aligned}L(w,w_0,q,\beta)=&\frac{\gamma}{2}\|q\|^2+\frac{1}{2}\left(\left(Xw+ew_0\right)^T\left(Xw+ew_0\right)\right)\\&-\beta^T\left[\left(X \quad e\right)^T\left(Xw+ew_0-y+q\right)\right]\end{aligned} \tag{4.39}$$

where $\beta\in\Re^{n+1}$ is the Lagrangian vector.

The Kuhn-Tucker optimality conditions are given by

$$\nabla_w L = 0 \Rightarrow X^T(Xw + ew_0) - X^T[X \quad e]\beta = 0. \tag{4.40}$$

$$\frac{\partial L}{\partial w_0} = 0 \Rightarrow e^T(Xw + ew_0) - e^T[X \quad e]\beta = 0. \tag{4.41}$$

$$\Rightarrow [X^T \quad e^T][X \quad e]\begin{bmatrix} w \\ w_0 \end{bmatrix} - [X^T \quad e^T][X \quad e]\beta = 0. \tag{4.42}$$

Let $H = [X \quad e]$ and let $u = \begin{bmatrix} w \\ w_0 \end{bmatrix}$. Then (4.42) can be written as

$$H^T Hu - H^T H\beta = 0 \Rightarrow u = \beta \tag{4.43}$$

$$\nabla_q L = 0 \Rightarrow \gamma q - [X \quad e]\beta = 0 \Rightarrow q = \frac{H\beta}{\gamma}. \tag{4.44}$$

$$\nabla_\beta L = 0 \Rightarrow [X \quad e]^T\left[[X \quad e]\begin{bmatrix} w \\ w_0 \end{bmatrix} - y + q\right] = 0. \tag{4.45}$$

Substituting $H = [X \quad e]; u = \begin{bmatrix} w \\ w_0 \end{bmatrix}$ in (4.45); we obtain

$$H^T[Hu - y + q] = 0. \tag{4.46}$$

Substituting q from (4.44),

$$H^T Hu + \frac{H^T H\beta}{\gamma} = H^T y. \tag{4.47}$$

Since $u = \beta$, we have

$$H^T H\left(1 + \frac{1}{\gamma}\right)\beta = H^T y. \tag{4.48}$$

$$\beta = \left(1 + \frac{1}{\gamma}\right)^{-1} \left(H^T H\right)^{-1} H^T y. \tag{4.49}$$

Although $H^T H$ is always positive semidefinite, it is possible that it may not be well conditioned in some situations. We introduce a regularization term, cI, to take care of problems due to possible ill-conditioning of $H^T H$, where I is an identity matrix of appropriate order. We thus have

$$\beta = \left(1 + \frac{1}{\gamma}\right)^{-1} \left(H^T H + cI\right)^{-1} H^T y. \tag{4.50}$$

Since $u = \beta$, we have $\beta = \begin{bmatrix} w \\ w_0 \end{bmatrix}$, where w is a $n \times 1$ vector and w_0 is a scalar. Given a new $x^i \in \Re^n$, the regressor estimate is given by $w^T x^i + w_0$.

For the non-linear case, there exists a mapping function $\phi : \Re^n \rightarrow \Re^{n_h}$, that maps from the n-dimensional input space to the higher n_h-dimensional feature space. In this case, the regressor hyperplane is obtained by incorporating the following modifications,

$$H = \left[\phi^T\left(X^T\right) \quad e\right]. \tag{4.51}$$

The vector of Lagrangian parameters is given by

$$\beta = \left(1 + \frac{1}{\gamma}\right)^{-1} \left[H^T H + cI\right]^{-1} H^T y. \tag{4.52}$$

The performance of PPSVR depends on the values of parameters such as c and γ, and is worthy of further investigation. The non-linear case requires the a priori choice of the mapping function ϕ to the higher dimensional feature space. A simple choice of mapping to the higher dimensional space could be obtained by concatenating the n-dimensional input by its square terms as below

$$\begin{bmatrix} x_1^i & x_2^i & \cdots & x_n^i \end{bmatrix}^T \xrightarrow{\phi} \begin{bmatrix} x_1^i & x_2^i & \cdots & x_n^i & (x_1^i)^2 & (x_2^i)^2 & \cdots & (x_n^i)^2 \end{bmatrix}^T. \tag{4.53}$$

Example: To learn the function $y = 1 + x^2 \sin(x)$, the (x, y)-tuples of the sample set is given by {(–3, –0.2701), (–2.2, –2.9131), (–1.4, –0.9315), (–0.6, 0.7967), (0.2, 1.0079), (1.0, 1.8415), (1.8, 4.1553), (2.6, 4.4848)}. The input and output set can be separated from this sample set as below:

$$X = \begin{bmatrix} -3 & -2.2 & -1.4 & -0.6 & 0.2 & 1.0 & 1.8 & 2.6 \end{bmatrix}^T_{1\times 8}$$

$$y = \begin{bmatrix} -0.2701 & -2.9131 & -0.9315 & 0.7967 & 1.0079 & 1.8415 & 4.1553 & 4.4848 \end{bmatrix}^T_{1\times 8}$$

Using polynomial terms upto 5-th order to devise the mapping function, ϕ as $\phi(x) = \begin{bmatrix} x & x^2 & x^3 & x^4 & x^5 \end{bmatrix}^T$ to map to a higher dimensional feature space, the new input-output set, $\{\phi(X^T), y\}$ for the function approximation problem becomes,

$$\phi(X^T)=\begin{bmatrix} -3 & -2.2 & -1.4 & -0.6 & 0.2 & 1 & 1.8 & 2.6 \\ 9 & 4.84 & 1.96 & 0.36 & 0.04 & 1 & 3.24 & 6.76 \\ -27 & -10.648 & -2.744 & -0.216 & 0.008 & 1 & 5.832 & 17.576 \\ 81 & 23.426 & 3.8416 & 0.1296 & 0.0016 & 1 & 10.498 & 45.698 \\ -243 & -51.536 & -5.3786 & -0.07776 & 0.00032 & 1 & 18.896 & 118.81 \end{bmatrix}_{5\times 8}.$$

$$y=\begin{bmatrix} -0.2701 & -2.9131 & -0.9315 & 0.7967 & 1.0079 & 1.8415 & 4.1553 & 4.4848 \end{bmatrix}^T_{1\times 8}.$$

Although the use of a mapping function ϕ can improve the regression ability of the SVM, it also increases the dimensionality of the data. Therefore the choice of an appropriate ϕ needs to be investigated. In the considered example, the matrix H is given by

$$H=\begin{bmatrix} \phi^T(X^T) & e \end{bmatrix}=\begin{bmatrix} -3 & 9 & -27 & 81 & -243 & 1 \\ -2.2 & 4.84 & -10.648 & 23.426 & -51.536 & 1 \\ -1.4 & 1.96 & -2.744 & 3.8416 & -5.3786 & 1 \\ -0.6 & 0.36 & -0.216 & 0.1296 & -0.07776 & 1 \\ 0.2 & 0.04 & 0.008 & 0.0016 & 0.00032 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1.8 & 3.24 & 5.832 & 10.498 & 18.896 & 1 \\ 2.6 & 6.76 & 17.576 & 45.698 & 118.81 & 1 \end{bmatrix}_{8\times 6}.$$

Hence the matrix to be inverted in the PPSVR solution is given by

$$H^TH=\begin{bmatrix} 27.2 & -16.192 & 165.59 & -161.28 & 1193.9 & -1.6 \\ -16.192 & 165.59 & -161.28 & 1193.9 & -1581.6 & 27.2 \\ 165.59 & -161.28 & 1193.9 & -1581.6 & 9324 & -16.192 \\ -161.28 & 1193.9 & -1581.6 & 9324 & -15282 & 165.59 \\ 1193.9 & -1581.6 & 9324 & -15284 & 76209 & -161.28 \\ -1.6 & 27.2 & -16.192 & 165.59 & -161.28 & 8 \end{bmatrix}_{6\times 6}.$$

The example shows, that if the effective dimension of the feature space is n_h, then PPSVR involves inverting a matrix of dimension $(n_h+1)\times(n_h+1)$. Conventionally, support vector regression involves solving a quadratic programming problem with a kernel size of the order $N\times N$, where N is the size of the data set. Applying the polynomial kernel from Table 4.5 to the above example, the kernel matrix in the dual formulation for a conventional support vector machine becomes

$$K=\begin{bmatrix} 100 & 57.76 & 27.04 & 7.84 & 0.16 & 4 & 19.36 & 46.24 \\ 57.76 & 34.106 & 16.646 & 5.3824 & 0.3136 & 1.44 & 8.7616 & 22.278 \\ 27.04 & 16.646 & 8.7616 & 3.3856 & 0.5184 & 0.16 & 2.3104 & 6.9696 \\ 7.84 & 5.3824 & 3.3856 & 1.8496 & 0.7744 & 0.16 & 0.0064 & 0.3136 \\ 0.16 & 0.3136 & 0.5184 & 0.7744 & 1.0816 & 1.44 & 1.8496 & 2.3104 \\ 4 & 1.44 & 0.16 & 0.16 & 1.44 & 4 & 7.84 & 12.96 \\ 19.36 & 8.7616 & 2.3104 & 0.0064 & 1.8496 & 7.84 & 17.978 & 32.262 \\ 46.24 & 22.278 & 6.9696 & 0.3136 & 2.3104 & 12.96 & 32.262 & 60.218 \end{bmatrix}_{8\times 8}.$$

For larger data sets, the K matrix becomes prohibitively large and solving the quadratic programming problem becomes computationally expensive. For example, LS-SVR [Suy99] solves a system of linear equations of size $(N+1)\times(N+1)$ by using SOR or Conjugate gradient, and the memory and computational requirements can be prohibitive for large data sets. For example, in the case of the Compactiv data set [Del], the size of the matrix is 8192×8192.

Thus, PPSVR promises to offer a higher computational efficiency as it involves inversion of a matrix whose size depends on the effective dimension of the feature vectors, and is independent of the size of the data set. This can lead

to significant computational savings for large data sets. When a linear kernel is used, least squares methods normally require solving a system whose size is the number of data points. In some cases, such as Proximal SVMs (PSVMs) [Fun2K1], the Sherman Morrison Woodbury (SMW) formula [Gol96] has been used to solve a smaller system. The SMW formula states that

$$\left(\frac{I}{C}+HH^T\right)^{-1}=\left[I-H\left(\frac{I}{C}+H^TH\right)^{-1}H^T\right]C. \tag{4.54}$$

The use of the SMW formula entails the inversion of a matrix of the size $n\times n$, and the three matrix multiplications. One of the matrix multiplications involves a larger matrix, since N is usually much larger than n. In the case of linear regression using PPSVR, we note that the matrix to be inverted is of size $n\times n$, where n is the input dimension. However, only two matrix multiplications are required, since the term $\left(1+\frac{1}{\gamma}\right)^{-1}$ is a scalar.

In the case of nonlinear regression, conventional support vector regression involves solving a quadratic programming problem with a kernel matrix of size $N\times N$, where N is the size of the data set. Least squares methods cannot use the SMW formula [Gol96], and the size of the system to be solved is usually $N\times N$. For example, in the case of the Compactiv data set [Del], the size of the system to be solved by conventional SVM or a typical least squares approach is at least 8192×8192.

If the effective dimension of the feature space is n_h, then PPSVR involves inverting a matrix of dimension $(n_h+1)\times(n_h+1)$. In the function approximation problem example, the size of the data set is $N=8$ while the input dimension is $n=1$. Using a mapping function, $\phi(x)=\begin{bmatrix}x & x^2 & x^3 & x^4 & x^5\end{bmatrix}^T$, the matrix to be inverted H^TH to obtain the regression parameters have a dimension of 6×6. By choosing the mapping ϕ carefully, n_h can always be chosen to be smaller than N. However, a poorly chosen mapping may lead to $n_h > N$. Even in such a situation, one can resort to the use of the SMW formula and use it for determining the Lagrange multipliers. In such a case, PPSVR would require the inversion of a matrix of size $N\times N$. To put it in a nutshell, the PPSVR regressor can always be found by inverting a matrix of order $\min(n_h, N)$.

In Section 4.6.3, we show that PPSVR is substantially faster than LS-SVM on large data sets. This is primarily on account of the smaller system that PPSVR needs to solve.

4.6.3 Applications of PPSVR

PPSVR was implemented using the software MATLAB (version 6.5 (R13)) [MAT], and tested on synthetic, as well as real benchmark data sets. While the synthetic data sets were run on a 800 MHz Pentium III PC, the benchmark data sets data sets were executed on SUNFIRE 6800, Ultra SPARC III CU Processor, 8

processor machine, 8 GB RAM, 900 MHz systems running the Unix operating system.

4.6.3.1 Application of PPSVR on Synthetic data sets

PPSVR was applied to learn a synthetically generated data set. Eight samples of the function $y = 1 + x^2 \sin(x)$ were chosen for the learning task. The sample set of (x, y)-tuples is given by {(–3, –0.2701), (–2.2, –2.9131), (–1.4, –0.9315), (–0.6, 0.7967), (0.2, 1.0079), (1.0, 1.8415), (1.8, 4.1553), (2.6, 4.4848)}. The mapping function used for applying PPSVR to learn the synthetic sample set was $\phi(x) = \begin{bmatrix} x & x^2 & x^3 & x^4 & x^5 \end{bmatrix}^T$. The optimal value of the regularization parameter γ was found to be $\gamma = 1071$. Choosing $c = 0.01$, PPSVR parameters were obtained as $w = \begin{bmatrix} 0.26303 & 0.0835 & 0.69211 & -0.015342 & -0.0772 \end{bmatrix}^T$, $w_0 = 0.95003$. Figure4.11 shows the original and the approximated curves obtained by using PPSVR to learn the 1-D synthetically generated data set.

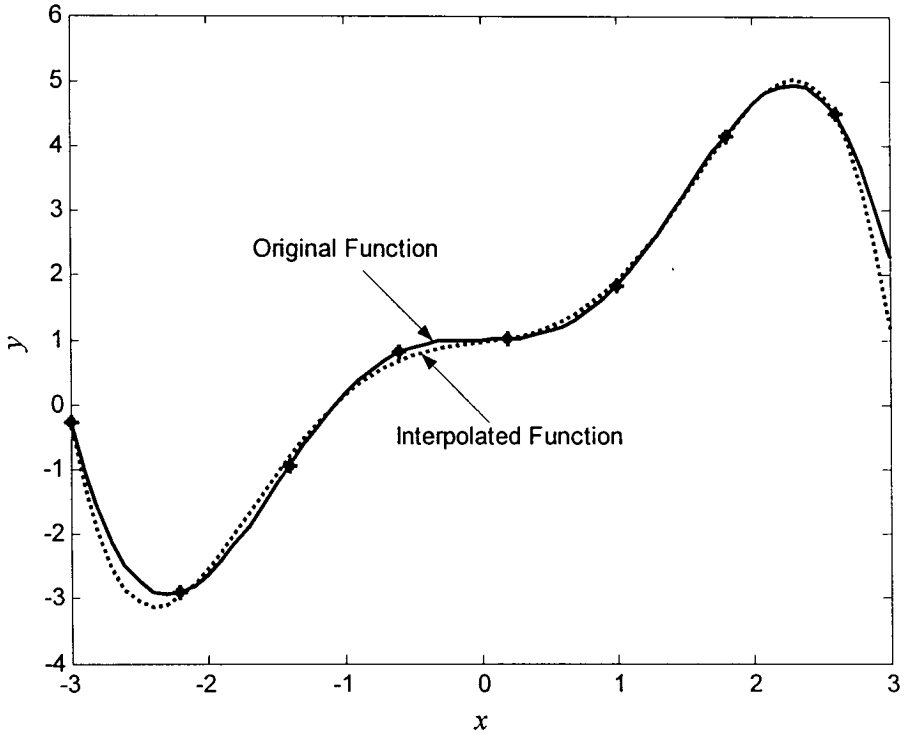


Fig. 4.11. A simple example illustrating PPSVR. Eight samples have been drawn from the function, $y = 1 + x^2 \sin(x)$ which is shown as a solid line. The samples are indicated by small "*"s on the curve. The PPSVR regressor obtained by using the mapping $\phi(x) = \begin{bmatrix} x & x^2 & x^3 & x^4 & x^5 \end{bmatrix}^T$ and by using all 8 data samples is shown as a dotted line.

The PPSVR algorithm was also tested on a two-dimensional synthetically generated data set. One hundred training samples of the function, $y = x_1 \sin(x_2)$ were uniformly drawn from the two-dimensional interval of $[-3 \quad 3] \times [-3 \quad 3]$. The mapping function ϕ contained initial terms from the expansion of $e^{x_1 + x_2}$. The actual mapping function used to represent the input to some higher dimensional feature space was given by

$$\phi(x) =$$
$$\phi(x_1, x_2) = \begin{bmatrix} x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 & x_1^3 & x_1^2 x_2 & x_1 x_2^2 & x_2^3 & x_1^4 & x_1^3 x_2 & x_1^2 x_2^2 & x_1 x_2^3 & x_2^4 \end{bmatrix}^T$$

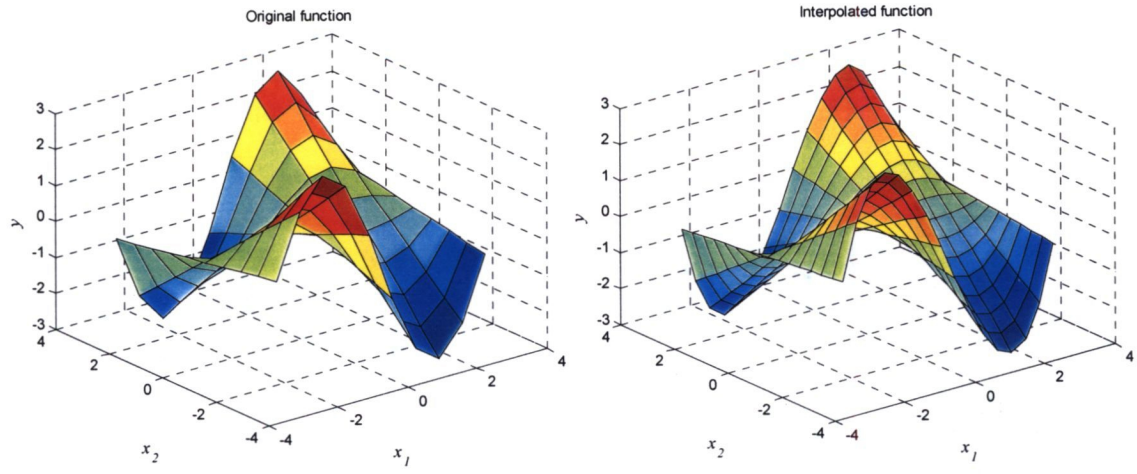
$\gamma = 10000$ was found to give a good regression estimate. Choosing $c = 0.01$, the PPSVR parameters were obtained as

$$w = \begin{bmatrix} -2.022\times10^{-7} \\ 3.4674\times10^{-4} \\ -2.1296\times10^{-6} \\ 0.84729 \\ 4.7663\times10^{-10} \\ 1.9828\times10^{-8} \\ -1.6891\times10^{-4} \\ 3.6861\times10^{-8} \\ 2.5104\times10^{-8} \\ 2.3425\times10^{-7} \\ -0.090637 \\ -5.507\times10^{-11} \\ 5.5819\times10^{-6} \\ -3.4149\times10^{-12} \end{bmatrix} \text{ and } w_0 = 2.222\times10^{-6}.$$

As observed, some of the regressor coefficients have small values and hence the corresponding terms may be neglected to simplify computation. Figure 4.12 shows the original curve and the PPSVR approximation. The interpolated curve was generated by the regressor to compute values at 256 uniformly distributed points.

4.6.3.2 Application of PPSVR on Real data

The performance of PPSVR was tested on some real benchmark data sets like the Housing data set, the Compactiv (CPU) data set, and the Census House



(a) (b)

Fig. 4.12. Application of PPSVR to learn a two-dimensional function. (a) Original function; (b) Interpolated function. 100 training samples have been drawn uniformly over the interval, $[-3 \quad 3]\times[-3 \quad 3]$ from the function, $y = x_1 \sin(x_2)$. The PPSVR regressor was obtained by using the mapping $\phi(\boldsymbol{x}) = \begin{bmatrix} x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 & x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^4 & x_1^3x_2 & x_1^2x_2^2 & x_1x_2^3 & x_2^4 \end{bmatrix}^T$ The interpolated function was obtained over 256 uniformly distributed points over the same interval.

data set. The Boston Housing set [Bla], that is widely used in regression problems contains 506 data points with 14 numerical attributes. The Compactiv data set [Del] contains 8192 data samples with 22 numerical valued attributes. The Census-House data set, also available at [Del], derived from the US census bureau data, that is utilized for predicting median home prices, contains 137 features. However, some features have constant values. These features have been removed, resulting in 121 features. A random subset of 5000 points from the complete data set was used.

Using these data sets, the performance of PPSVR was compared with that of LS-SVR [Sau98, Suy99]. Their comparative performances in terms of their

training and testing errors after ten-fold cross-validation runs are tabulated in Tables 4.6-4.8. Optimal values of the hyperparameters were obtained by using a tuning set consisting of 10% randomly chosen samples from the original data set. As a measure of the performance criterion, in a data set of N samples, if y_i denotes the desired output of a sample, and $\hat{y}_i$ denotes its estimated value, the percentage error over the whole set was calculated as

$$\% \ error = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|^2}{\sum_{i=1}^{N} y_i^2} \times 100 . \tag{4.55}$$

In all the cases of PPSVR, we chose $c = 0.01$.

From the results displayed in tables 4.6-4.8 it is observed that the performance of PPSVR and LS-SVR are nearly identical for the linear case. However, when a nonlinear mapping is used in PPSVR, or when polynomial or RBF kernels are used in LS-SVR, the performance of LS-SVR improves. In fact, though the learning time of LS-SVR is much higher than that of PPSVR, comparatively better function estimation is provided by LS-SVR using the RBF kernel.

Table 4.6: 10-fold Cross validation performance on the Housing data set (506×14)

Algorithm	γ	Kernel parameters	Training Error (%)	Testing Error (%)	Run Time (s)
PPSVR (Input only)	3235.3	-	19.15 ± 0.39	19.62 ± 3.34	3.1645
PPSVR (Input & its square terms)	3036.8	-	15.59 ± 0.41	16.36 ± 3.21	2.8584
LS-SVR (Linear)	0.8	-	19.16 ± 0.4	19.5 ± 3.31	68.1669
LS-SVR (Polynomial kernel)	0.03	$t=1; d=2$	11.35 ± 0.30	13.47 ± 3.77	102.923
LS-SVR (RBF kernel)	46	$\sigma^2=14$	6.67 ± 0.26	11.6 ± 2.35	9.4246

Table 4.7: 10-fold Cross validation performance on the Compactiv data set (8192×22)

Algorithm	γ	Kernel parameters	Training Error (%)	Testing Error (%)	Run Time (s)
PPSVR (Input only)	459.46	-	11.14 ± 0.13	11.23 ± 1.15	323.00
PPSVR (Input & its square terms)	1810	-	4.076±0.006	4.125 ± 0.06	324.608
LS-SVR (Linear)	180.79	-	11.14 ± 0.07	11.23 ± 0.70	21715.975
LS-SVR (Polynomial kernel)	295.34	$t=1; d=2$	3.167 ± 0.01	3.54 ± 0.5	30477.06
LS-SVR (RBF kernel)	298	$\sigma^2=1.3$	2.28 ± 0.06	2.87 ± 0.11	6371.78

Table 4.8: 10-fold Cross validation performance on the Census House data set (5000×121)

Algorithm	γ	Kernel parameters	Training Error (%)	Testing Error (%)	Run Time (s)
PPSVR (Input only)	1033	-	7.88 ± 0.108	8.4 ± 1.037	131.375
PPSVR (Input & its square terms)	1169	-	6.72 ± 0.103	7.95 ± 1.33	146.405
LS-SVR (Linear)	65	-	7.94 ± 0.094	8.64 ± 0.84	8757.22
LS-SVR (Polynomial kernel)	0.003	$t = 1; d = 2$	3.65 ± 0.039	9.36 ± 1.44	12147.29
LS-SVR (RBF kernel)	100	$\sigma^2 = 1970$	5.99 ± 0.082	8.765 ± 1.59	1841.87

4.7 Conclusion

As a continuation of the theme of this thesis, in this chapter, we have proposed a simple but effective LP based technique for approximating a function of a single variable by using a tree-structured neural network. The network uses linear threshold neurons, and new neurons are added until the residual errors at all sample points are brought down to within a specified tolerance of $\pm\Delta$. The working of the method has been illustrated with an example. The proposed algorithm compares favorably with a multi-layer network that was trained using the back-propagation algorithm. A major advantage with the proposed approach is that there is no need to choose the number of hidden layers, hidden neurons, learning rate, or the initial weight values. Since all weights are determined by solving a simple linear programming problem, problems of local minima are avoided. On a chosen example, the proposed algorithm used fewer neurons to

approximate the given samples to within a specified tolerance, and required less time to complete the learning. The experimental results emphasize the empiricism inherent in the use of back-propagation, and the sensitivity of performance to the number of hidden neurons. In addition, the approximations obtained by MLBNs tend to add spurious extremal points, and it is difficult to limit the error at individual samples.

Owing to the nature of the linear threshold neurons, the algorithm cannot be directly used for approximating a function having negative values. For functions that do not satisfy this requirement, an offset needs to be added prior to applying the algorithm. The number of neurons used appears to be larger than the minimum needed, and there is scope for improving the algorithm in this regard.

The approach opens a direction for building constructive networks for approximating functions of multivariables. However while extending it to higher dimensions it has been observed that cross interference of the hyperplanes as represented by the neurons disturbs the correctly learnt samples at the parent layer. Therefore alternative strategies were sought to overcome this problem.

A new approach to function approximation called Potential Proximal Support Vector Regression (PPSVR) has been proposed, that is computationally

simpler compared to conventional Support Vector Regression, and requires a single matrix inversion to obtain the regressor as compared to conventional SVMs, that require solving a constrained quadratic programming problem. The efficacy of the method has been demonstrated by its application on real benchmark data sets and by comparison with other algorithms. One drawback of PPSVR is the requirement of an explicit choice of the mapping function ϕ.

In the next chapter, we attempt some control applications. A control application is basically a learning problem. In the next chapter we apply SVM based tree type neural networks for control applications. To learn any data set having real valued outputs we have used the well-established regression tool, Least Squares Support Vector Regression (LS-SVR) [Sau98, Suy99] that has good generalization performance.

Chapter 5

Applications to Control

In Chapters 1-4 of this thesis, linear programming formulations have been used to develop constructive networks for binary classification and function approximation. LP based approaches provide simple formulations, and offer advantages such as parallelism and fast computation, that are important for real-time implementation. This chapter contains two sections that deal with the application of some of the proposed networks for control engineering applications. Section 5.1 investigates the use of the SVM based tree type network as a critic in Adaptive Critic Designs for control. Finally, Section 5.2 proposes a new approach to the direct torque control of an induction motor using the networks. Section 5.3 contains concluding remarks.

5.1 SVM based tree type neural network as a critic in Adaptive Critic Designs

5.1.1 Introduction

Open-loop applications such as signal processing or system identification are significantly different from closed-loop control applications. In the latter situation, interactions between plant dynamics and controller parameters come into play, introducing additional complexity into the system. Control action is closely related to function approximation, and neural networks are known to possess a good

function approximation property [Cyb89, Hor89]. Recent developments in the application of nonlinear models based on artificial neural networks hold much promise in the field of model-based control, where input-output training data is available, and supervised learning schemes can be used for learning the data. However, it is desirable to apply more advanced learning methods and intelligent features in control applications that may eliminate the need for analytic modeling of a plant. One approach in this direction is the use of reinforcement learning, where the available information is not as direct, immediate or informative as in supervised learning, and serves more to evaluate the system, as shown in Fig.5.1. This is particularly useful for control problems that require selecting actions whose consequences emerge over uncertain periods, for which input-output training data are not readily available. A particularly intriguing reinforcement learning topology is the adaptive critic [Wer90, Pro95, Pro97a, Pro97b] methodology, that is based on dynamic programming.

Support Vector machines, known for their generalization capability and discussed in Section 1.5, are good at pattern recognition and estimation tasks. Conventional SVM formulations optimize a quadratic objective function subject to certain linear inequality constraints. Another class of SVMs, termed as Least Squares Support Vector Machines (LS-SVM) have been proposed [Sau98, Suy99], in which the constraints are of the equality type. As discussed in Section 4.6.1, in LS-SVMs, the solution hyperplane is obtained by solving a set of linear equations.

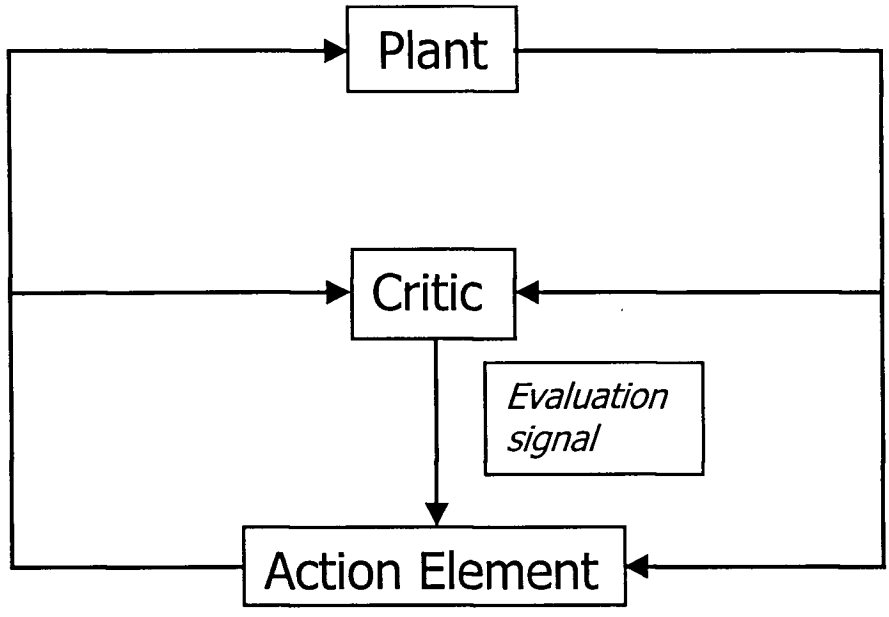


Fig. 5.1. Reinforcement Learning Scheme.

Using a chosen kernel, this technique gives a good estimation by placing the kernel functions at the training points. In this work, a LS-SVM regressor has been trained to act as the controller.

Section 5.1.2 briefly introduces the different adaptive critic designs. Section 5.1.3 describes the control strategy adopted in our work. Section 5.1.4 discusses experimental results.

5.1.2 Adaptive Critic Designs

Adaptive Critic designs (ACDs) have evoked growing interest in nonlinear control, because of their capability to approximate optimal control over time in nonlinear environments [Wer90]. Any real-life optimization problem can be cast as a minimization or maximization task, and an optimal solution can be achieved if

dynamic programming is used. However, dynamic programming methods are computation-intensive, due to initiation of the search process from the goal, and the dependence at any stage on the computation of the cost-to-go to the next stage, for different available controls. They also need highly efficient data structures and programming capabilities to carry out the method stage-by-stage. The "curse of dimensionality" has limited their use as a tool for finding solutions to complex optimization problems. Attempts have been made to overcome this curse by using heuristic strategies to approximate the cost function. Adaptive-critic design offers one such approach, where one attempts to build a "critic" that learns the cost function in dynamic programming. Multi-layer feed-forward neural networks have been known to be good function approximators because of their capability to represent complex decision regions. Therefore, neural networks have generally been useful for building function approximation structures in adaptive critic designs, though any other structure can suffice.

The three basic methods of ACDs are heuristic dynamic programming (HDP), dual heuristic programming (DHP), and globalized dual heuristic programming (GDHP). The basic blocks of an ACD are the action element and the critic element (evaluation module), but in some of the structures a model (identification) network is also used for identifying the dynamics of the plant. If the critic element has the action/control signals as part of its input, those designs are referred to as action dependent ACDs (ADACDs). Prokhorov et al. [Pro95,

Pro97b] and Prokhorov [Pro97a] present an elaborate discussion of different adaptive critic designs.

5.1.2.1 Heuristic Dynamic Programming (HDP) and Action Dependent Heuristic Dynamic Programming (ADHDP)

In a discrete-time nonlinear dynamical system,

$$x(t+1) = P[x(t), u(t), t] \tag{5.1}$$

where $x(t) \in \Re^n$ denotes the system state, and $u(t) \in \Re^m$ denotes the control action, the state of the system changes at every instant with the application of a control input. This change is guided by the dynamical nature of the system, which is captured by the function P. As the system's state changes from one instant to the next it is associated with a cumulative cost to drive the system from that instant to the goal. The performance index $J(.)$ for Heuristic Dynamic Programming at any instant is given by

$$J[x(t), t] = \sum_{k=t}^{\infty} \gamma^{k-t} U[x(k), k], \tag{5.2}$$

where $U(.)$ is the utility function or local cost function, and γ is the discount factor with $0 < \gamma < 1$. The above functional, J, is the cost-to-go of the state $x(t)$ for the infinite horizon problem. If the control actions are also part of the utility function, it is termed as the Action Dependent Heuristic Dynamic Programming, whose performance index is given by

$$J[x(t), t] = \sum_{k=t}^{\infty} \gamma^{k-t} U[x(k), u(k), k]. \tag{5.3}$$

Figure 5.2 shows a Critic for the HDP/ADHDP scheme. In HDP, $R(t)$ is a vector of the states $x(t)$ of the plant, while for ADHDP, it is a concatenation of the states

Fig. 5.2. Critic for a HDP.

$x(t)$ and the control vector $u(t)$. Minimization of $J(t)$ is the desired objective. Equations (5.2) or (5.3) may be written more succinctly as

$$J[t]=\sum_{k=t}^{\infty}\gamma^{k-t}U[k]. \tag{5.4}$$

Expanding the above, and using the terminology for the cost-to-go at (t+1), we have

$$\begin{aligned} J[t] &= U[t]+\gamma U[t+1]+\gamma^2 U[t+2]+... \\ &= U[t]+\gamma J[t+1]. \end{aligned} \tag{5.5}$$

For any chosen instant 't', the objective is to find an appropriate control sequence $u(k), k=t,t+1,...$ so that the function J is minimized. The task of the critic network is to learn the function $J[t]$. As a general training strategy, the critic network has to be trained by minimizing the following error measure over time

$$\|E_1\|=\sum_t E_1^2(t), \tag{5.6}$$

where $E_1(t)=J(t)-U(t)-\gamma J(t+1)$.

A neural network may be trained to act as the critic, so that at any instant it can output the cost function for the immediate future. Denoting the critic's parameters as W_c in an ADHDP, the input-output relationship of the critic network may be written as

$$J[t] = J[x(t), u(t), W_c]. \tag{5.7}$$

5.1.2.2 Dual Heuristic Programming (DHP) and Action Dependent Dual Heuristic Programming (ADDHP)

In Dual Heuristic Programming (DHP) and Action Dependent Dual Heuristic Programming (ADDHP), the critic network estimates the derivatives of J with respect to the vector $R(t)$, where $R(t)$ is the vector of the states $x(t)$ of the plant (DHP), or a concatenation of the states $x(t)$ and the control vector $u(t)$

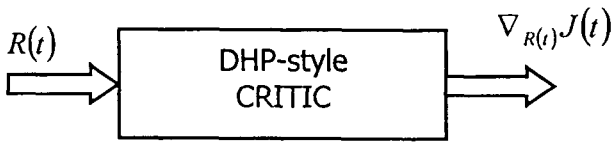


Fig. 5.3. Critic for a DHP.

(ADDHP), as shown in Fig. 5.3. The critic element trains to minimize the following time-dependent error measure

$$\|E_2\| = \sum_t E_2^T(t) E_2(t), \tag{5.8}$$

where

$$E_2(t) = \nabla_{R(t)} J(t) - \gamma \nabla_{R(t)} J(t+1) - \nabla_{R(t)} U(t). \tag{5.9}$$

Here, the derivatives are taken with respect to the components of the vector $R(t)$. The need for the existence of derivatives in the error measure, and the training of the critic by minimizing a certain norm of the error, make it more complicated as compared to HDP.

5.1.2.3 Globalized Dual Heuristic Programming (GDHP)

Globalized Dual Heuristic Programming (GDHP) and its Action Dependent (AD) form are a fusion of HDP and DHP. Here, the critic network adapts in such a way that it estimates J and its derivatives with respect to the components of $R(t)$. The vector $R(t)$ is the vector of the states $x(t)$ of the plant in the case of GDHP, or a concatenation of the states $x(t)$ and the control vector $u(t)$ in the case of ADGDHP, as shown in Fig. 5.4. The output of a GDHP has two components: a scalar component $J(t)$, and the vector component, which consists of the derivatives of $J(t)$ with respect to the components of $R(t)$. Therefore the adaptation of a GDHP critic would involve minimizing an error measure that is equal to the sum of the error measures of HDP and DHP, as given in (5.6) and (5.9), respectively. GDHP and ADGDHP are the most complicated adaptive critic design approaches, and are expected to be superior to other methods [Pro97a, Pro97b].

Detailed illustrations of the different adaptive critic designs with examples can be found in Prokhorov et al. [Pro95, Pro97b] and Prokhorov [Pro97a].

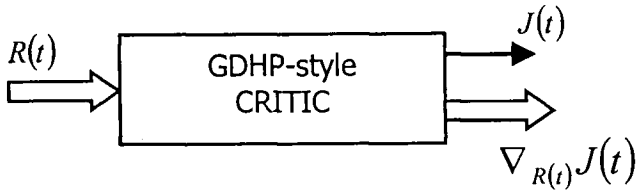


Fig. 5.4. Critic for a GDHP.

5.1.3 Control Strategy

It is desired to develop a controller for a plant so as to minimize occurrence of failures, and also to provide smooth operation. To achieve this control objective, the ADHDP strategy has been adopted wherein the critic and action networks work in tandem; the critic network tracks the performance of the plant from healthy to failure condition, while the actual control inputs are generated by the action network, based on the plant state. The quality of control inputs generated by the action network is indicated by binary signals generated at the output of the critic network. The complete scheme for ADHDP based controller is shown in Fig. 5.5.

A popular example of a plant where a controller of this type may be incorporated, is the inverted pendulum or the cart-pole problem as shown in Fig.5.6. The objective here is to balance the pole in the upright position. The bottom of the pole is hinged on a cart. The only control input available is in terms of the horizontal force that can be applied to the cart along the track. Failure occurs when the pole falls past a certain angle, or the cart hits the end of the track. Using this failure condition, the utility function of the cart-pole system is given by

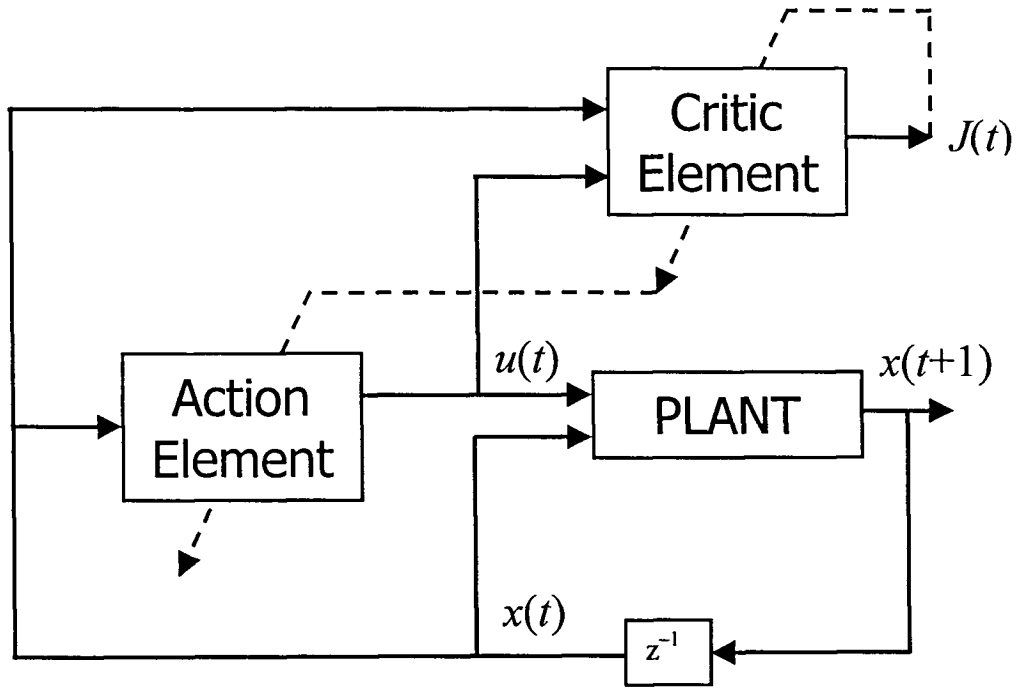


Fig. 5.5. Control using Action Dependent Heuristic Dynamic Programming.

$$U(t) = \begin{cases} 0, & \text{when no failure} \\ 1, & \text{failure condition.} \end{cases} \tag{5.10}$$

Due to this formulation of the utility function the reinforcement signal is obtained only when the controller is unable to control the plant. This information regarding the smooth operation to a failure condition of the plant's states is used for retraining the critic and action networks for ADHDP based design. The plant states and the corresponding utility value 0/1 depending upon no failure/failure constitute data for a binary classification problem. This data can be used for training a learning machine as a critic.

Training of an ACD is a sequential process, with the training alternating between the critic and action networks. The critic network is first trained with the

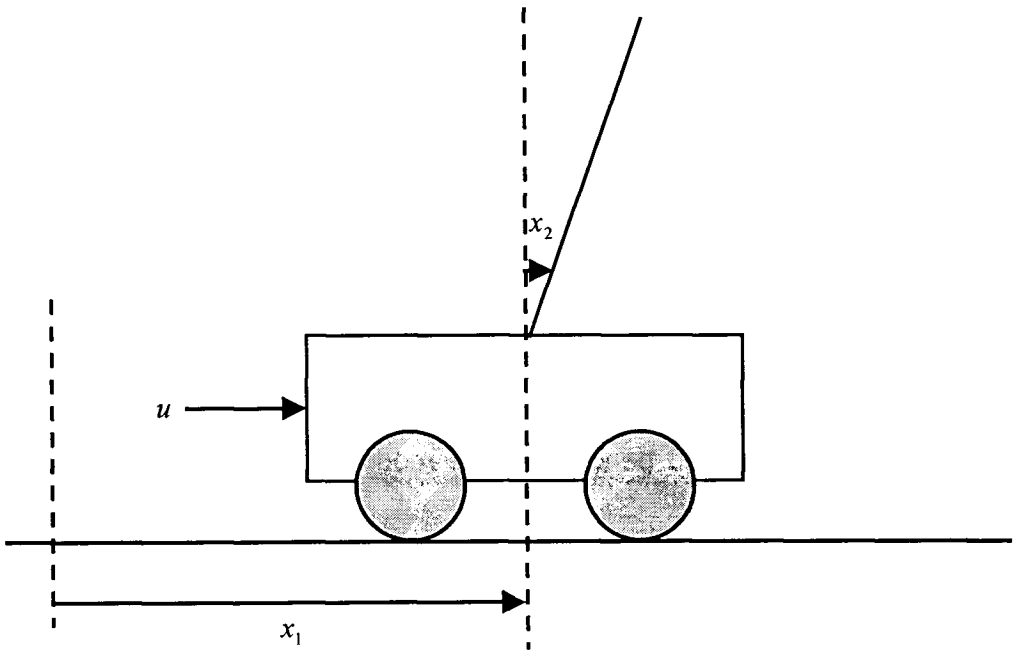


Fig. 5.6. The Cart-Pole Model.

failure data. Once the critic is trained, action training starts. It is desired to minimize $J(t)$ at the output of the critic in the immediate future, so that it would optimize the overall cost expressed as the sum of future $U(t)$. This will necessitate training of the action element, so that the output of the critic is as small as possible. Ideally, one would like to have a zero output of the critic network, so that the performance index J is minimized. The desired mapping needed for training the action network in the present ADHDP set up is given by

$$A: \{x(t)\} \rightarrow \{0(t)\}, \tag{5.11}$$

where $\{0(t)\}$ indicates that the target value at the output of the critic network is zero for an input $\{x(t)\}$ at the input of the action network. In other words, at

instant 't', the action network should generate appropriate control actions $\{u(t)\}$ in response to the state $\{x(t)\}$, so as to drive $J(t)$ given by (5.7) to zero.

5.1.4 Experimental Results: Control by ADHDP

5.1.4.1 Plant Description

To test the above control strategy, the chosen plant was a SIMO system, viz. the model of a single link inverted pendulum. The model and its parameters have been adopted from an actual experimental test bed made by Feedback Instruments [Fed]. Here the state of the system is the vector

$$x = [x_1 \quad x_2 \quad x_3 \quad x_4]^T, \tag{5.12}$$

where x_1 is the cart position (measured from the center of the rail); x_2 is the angle between the upward vertical and the axis of the pole, measured counter-clockwise from the cart; x_3 is the cart velocity, and x_4 is the pendulum angular velocity.

The pendulum can rotate in the vertical plane, while the cart can move on a horizontal rail lying in the plane of rotation. The only control that can be applied is in terms of a horizontal force u applied on the cart parallel to the rail. The dynamical equations governing the plant motion are given by

$$\dot{x}_1 = x_3, \tag{5.13a}$$

$$\dot{x}_3 = \frac{a\left(u - T_c - \mu x_4^2 \sin x_2\right) + l\cos x_2\left(\mu g \sin x_2 - f_p x_4\right)}{J + \mu l \sin^2 x_2}, \tag{5.13b}$$

$\dot{x}_2 = x_4$, and (5.13c)

$$\dot{x}_4 = \frac{l\cos x_2\left(u - T_c - \mu x_4^2 \sin x_2\right) + \mu g \sin x_2 - f_p x_4}{J + \mu l \sin^2 x_2}, \tag{5.13d}$$

where $a = l^2 + \frac{J}{m_c + m_p}$; $\mu = \left(m_c + m_p\right)l$; g is the acceleration due to gravity, m_c is the mass of the cart, m_p is the mass of the pendulum, l is the distance from the axis of rotation to the center of mass of the pendulum-cart system, J is the moment of inertia of the pendulum-cart system with respect to the center of mass, T_c denotes the friction in the motion of the cart; $T_c = f_c x_3$, and D_p is the moment of friction in the angular axis of the pendulum, proportional to the angular velocity; $D_p = f_p x_4$. The admissible controls are bounded, i.e. $|u(t)| \le M$. The different values of the pendulum-cart set up are given in Table5.1.

Table 5.1: Parameters of the Pendulum-Cart set up

Parameter	Value
Track limits	± 0.5m
Acceleration due to gravity, g	9.81 m/s^2
Distance between mass center and the axis of rotation, l	0.017 m
Mass of the cart, m_c	1.12 kg
Mass of the pole, m_p	0.11 kg
Magnitude of control force, M	17.0 N
Moment of Inertia of the system, J	0.0136 kg-m^2
Friction coefficient of pole rotation, f_p	Negligible
Friction coefficient of cart, f_c	0.05 Ns/m

5.1.4.2 Control Setting

5.1.4.2.1 Critic Element

As discussed in Section 5.1.3, in the control approach using ADHDP based design, a batch of failure data comprising the plant states and the corresponding utility values 0/1 depending upon no failure/failure of the plant constitute a binary classification data set. In this data set, the number of 'failure' states are far fewer than the number of 'no failure' states. Problems have been reported while training a critic to learn such data with multilayer networks using a "tanh" activation function [Liu2K2]. The analytical basis for failure to learn binary classification data by multilayer neural networks using "sigmoid" and "tanh" activation functions have been discussed in Section 1.4. This work attempts to overcome the difficulties for learning binary classification data by using the SVM based tree type neural network proposed in Chapters 2 and 3, as the critic. The constructive nature of this network, based on LP formulations of the problem, guarantee complete learning of any binary classification data, as has been demonstrated on real benchmark data sets in Chapter 3.

5.1.4.2.2 Action Element

The action element is the main controller that generates control actions depending on the plant states. Making use of the generalization capabilities of Support Vector Machines, a least squares support vector machine (LS-SVM) regressor (Section 4.6.1) with a RBF kernel function was used as the action

element. Consequently, the number of hidden units did not need to be determined a priori and centres do have not to be specified for Gaussian kernels, as the kernels were centred on the failure states. The regularization parameter γ and the kernel parameter σ^2 were tuned to minimize error during training.

5.1.4.2.3 Plant States

If the displacement of the cart and the angular displacement of the pole are denoted by $x(t)$ and $\theta(t)$, respectively, then the four components of the plant's state that have been considered in the control scheme are $x_1 = x(t)$, $x_2 = \theta(t), x_3 = \dot{x}(t)$, and $x_4 = \dot{\theta}(t)$.

5.1.4.2.4 Utility Function

The actual utility function used for developing the ADHDP type control scheme is given by

$$U(t) = \begin{cases} 0, & |\theta(t)| \leq 12^{\circ} \text{ and } |x(t)| \leq 0.5 \text{ m} \\ 1, & \text{otherwise.} \end{cases} \quad (5.14)$$

The utility function penalizes a state if the pole angle falls past either side of the vertical by 12°, or the cart reaches the end of the track 0.5 m from the center. For the plant being considered, the state variable of utmost interest is the pendulum vertical angle $\theta(t)$, that renders the plant to the 'failure' state most often.

5.1.4.3 Controller Implementation

Using the plant parameters as given in Table 5.1, the complete control scheme was built by using the simulation software SIMULINK (version 5.0) and by interfacing it with the MATLAB (version 6.5 (R13)) programming environment. The plant states and the corresponding critic output are available in the MATLAB workspace to retrain the action element after obtaining failure data.

In our implementation, all the state variables and the input u have been normalized to lie between –1 and 1. The maximum and minimum values used for normalizing the state variables of the Pendulum-Cart system are shown in Table 5.2.

Table 5.2: Maximum and Minimum values used for normalization of the state variables of the Pendulum-Cart system

State Variable	Maximum	Minimum
Pendulum Angle, $\theta(t)$ (radians)	$\pi/15$	$-\pi/15$
Pendulum Angular Velocity, $\dot{\theta}(t)$ (radians/sec)	2π	-2π
Cart Position, $x(t)$ (m)	0.5	-0.5
Cart Velocity, $\dot{x}(t)$ (m/sec)	3	-3

The states and the input of the critic and the action networks were randomly initialized. The initialized states and the input give the initial training set for the action element. A 3-layer feed-forward network was used as the initial action element, which during subsequent training with actual fault data, was replaced by a LS-SVM regressor. Applying the utility function on the randomly initialized states, the utility value of each state x can be obtained. The initial states,

inputs, and the corresponding utility value, act as the initial training set for the SVM based tree type neural network.

The plant was simulated with the initialized states, and training data was collected for both the critic and the action networks. Once a failed state is reached (which is indicated by the critic output of +1), both the critic and the action elements need to be updated. Once a batch of failure data is obtained by conducting experiments from some randomly generated initial conditions over the state space, a new critic has to be trained. Once a new critic has been obtained, out of a set $\{u_i, i = 1,2,...,k;\ u_i \in (-1,1)\}$ of k available controls, the proper control action was chosen. This is the action that produces a zero at the output of the critic due to the failure state, as well as providing a control force in the right direction to balance the cart-pole system. This control is now the desired control output from the action network for the failure state. Once the failure states and their desired control actions to balance the pole are known, the LS-SVM regressor is trained. The regularization parameter γ and the kernel parameter σ^2 were tuned by using the function *tunelssvm* of the LS-SVM toolbox [Pel]. This cycle of training the critic and the action element is continued whenever the critic network indicates a failure. The more the number of zero values at the output of the critic, the more the value function $J(.)$ is reduced.

The trained ADHDP based controller had a 14 node SVM based tree type neural network as the critic, and the parameters of the LS-SVM regressor based action element were $\gamma = 94.411$ and $\sigma^2 = 5.679$.

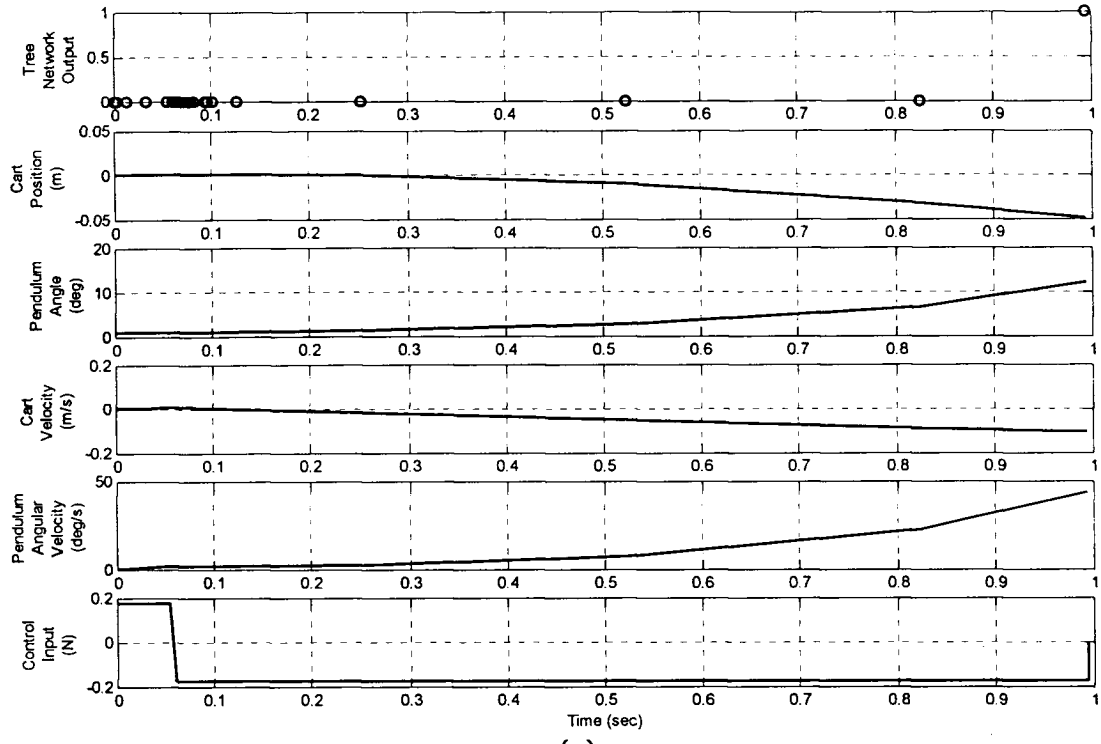
5.1.4.4 Simulation Results

The sample performance of the tree network as the critic and the progress of the state trajectories of the cart-pole system with the input during training of the critic and action elements in tandem, has been demonstrated in the subplots of Fig. 5.7. As pole angle falls past the vertical beyond $+12^{\circ}/-12^{\circ}$, the output of the critic switches from 0 to 1, indicating a failure condition and stopping further control input to the plant as shown in Figs. 5.7a and 5.7b.

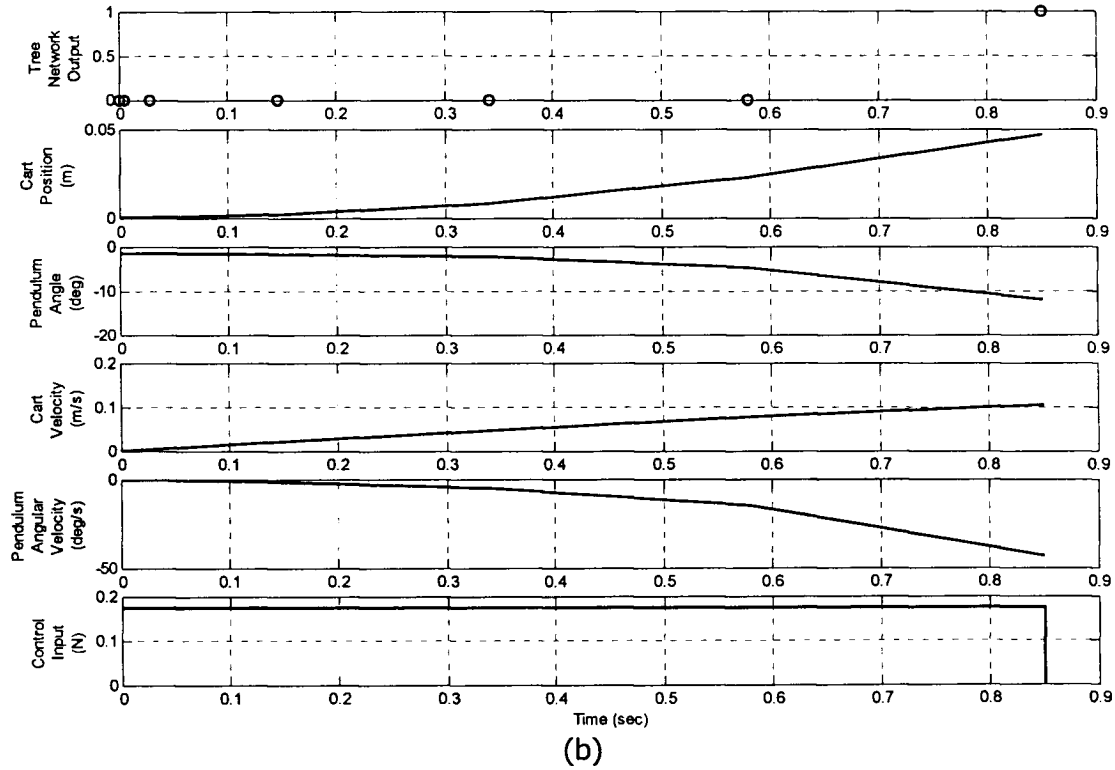
Figure 5.8 shows the performance of a trained ADHDP based controller with the pole initially placed at 10.8° from the vertical. The controller successfully steers the plant and it achieves an equilibrium state of

$$x^e = [0.146 \quad 0 \quad 0 \quad 0]^T.$$

The safe operation of the plant is been indicated by zero outputs at the output of the tree network, while the Gaussian-kernel based LS-SVM regressor provides a smooth control.



(a)



(b)

Fig. 5.7. Sample trajectories from the ADHDP training scheme of the Cart-Pole system. a) Failure condition when the pendulum angle falls past the positive limit; b) Failure condition when the pendulum angle falls past the negative limit.

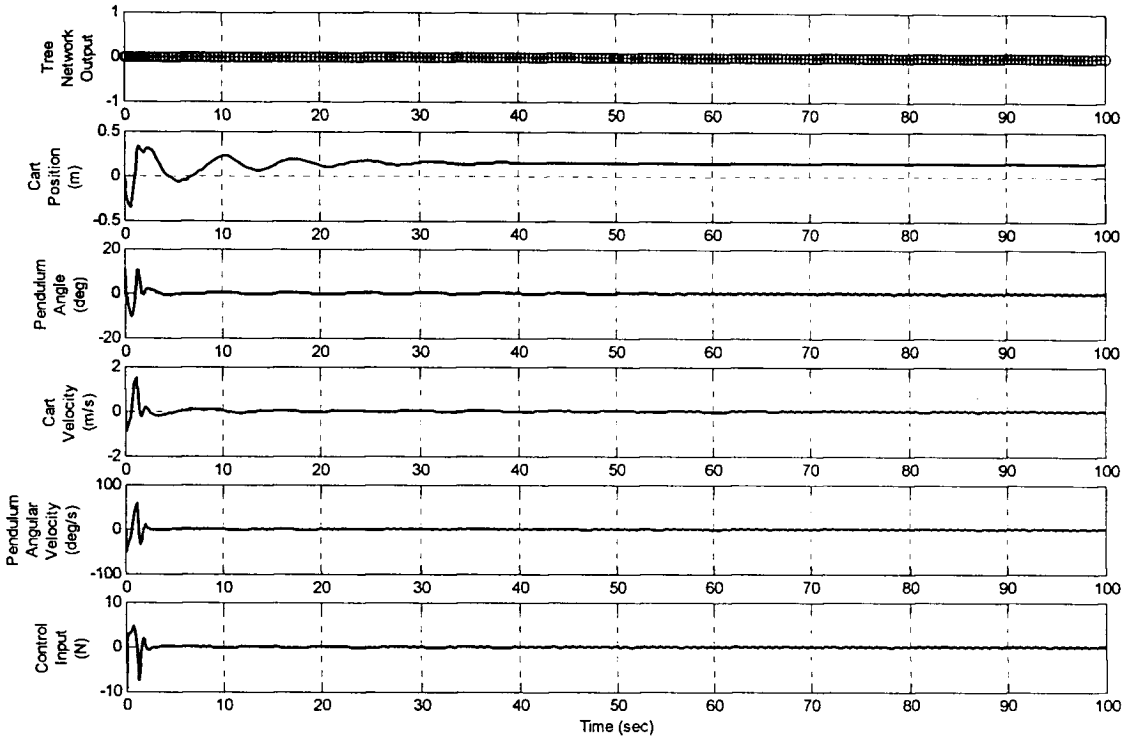
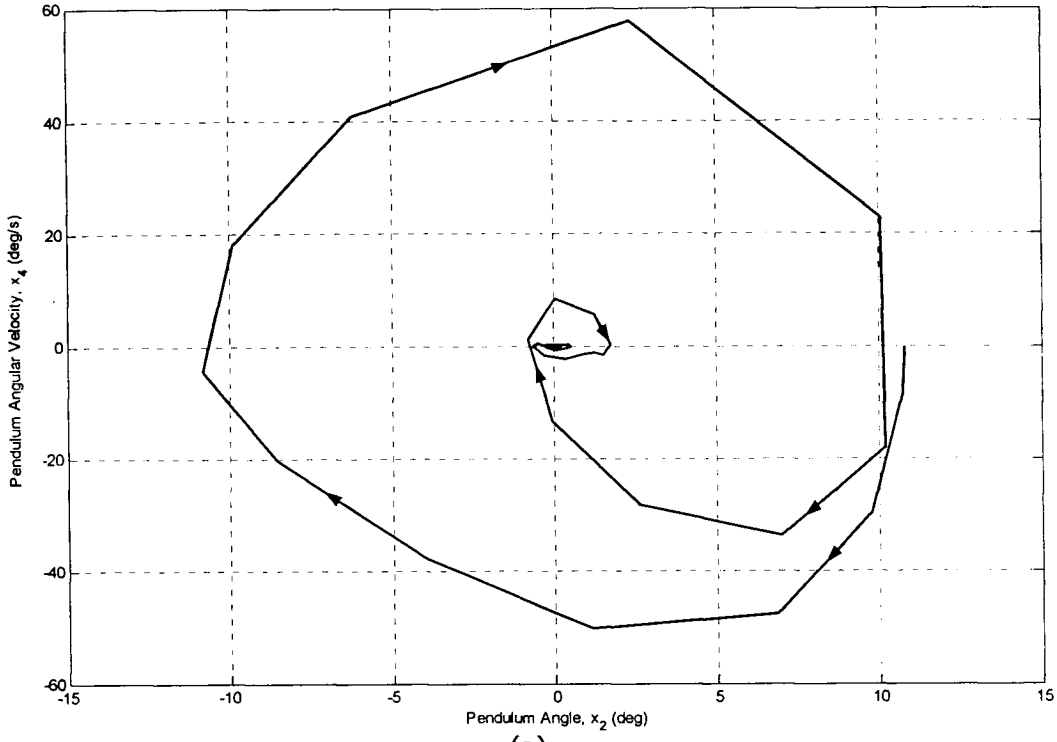
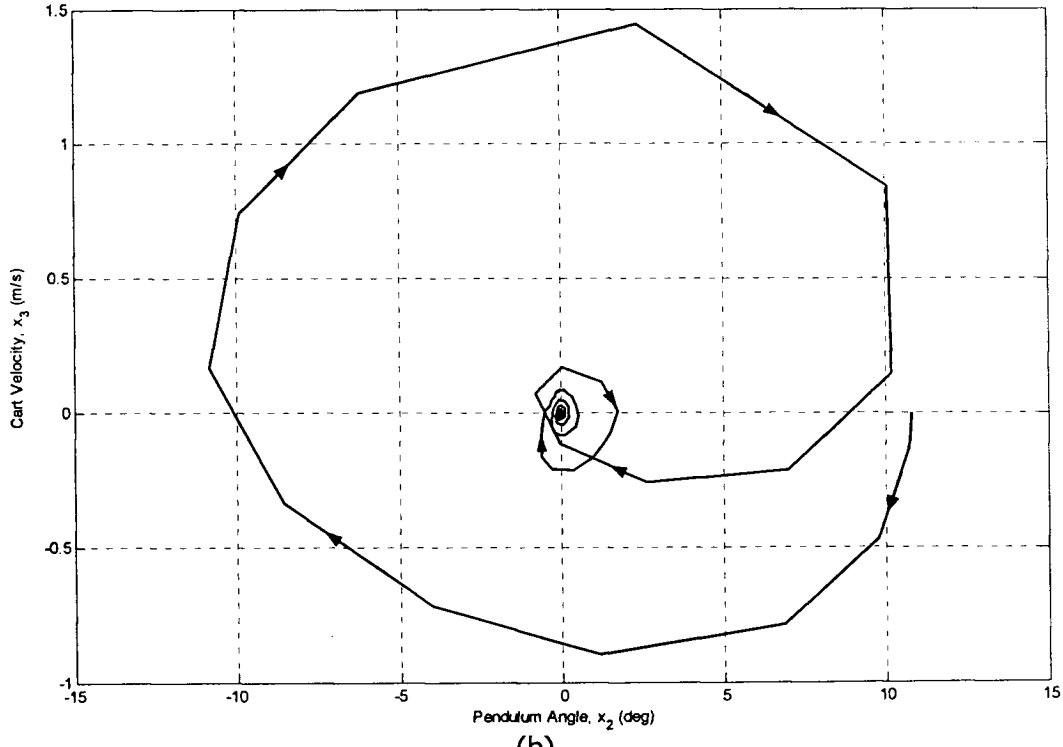


Fig. 5.8. Balancing the pendulum by the trained ADHDP based controller. Initial angle of the pendulum at 10.8° from the vertical.

The main objective of the pole-balancing problem is to keep the state of pole angle vertical within the bounds of the track. Convergence of the velocity components of an equilibrium state x^e to the origin ensures stability of a plant [Gop97]. The simulation data for an initial pole angle of +10.8° has been used to draw the phase-plane trajectories as shown in Figs. 5.9a and 5.9b. The trajectories depict the convergence of the pendulum angular velocity and the cart velocity to the origin, along with the pendulum vertical angle. With the final cart position well inside the track, the ADHDP based controller ensures smooth operation of the cart-pole system.



(a)



(b)

Fig. 5.9. Phase Plane plots showing convergence of the ADHDP based control scheme: a) Pendulum Angular Velocity (x_4) vs Pendulum Angle (x_2), b) Cart Velocity (x_3) vs Pendulum Angle (x_2).

The performance of the ADHDP based control scheme for the pole-balancing problem was tested for the initial position of the pole on both sides of the vertical. Figure 5.10 shows the trajectory of pendulum angle for initial angle of the pole at 10.8° and –10.8° from the vertical. The Gaussian-kernel based LS-SVM regressor that acts as the action element ensures smooth control while balancing the pole.

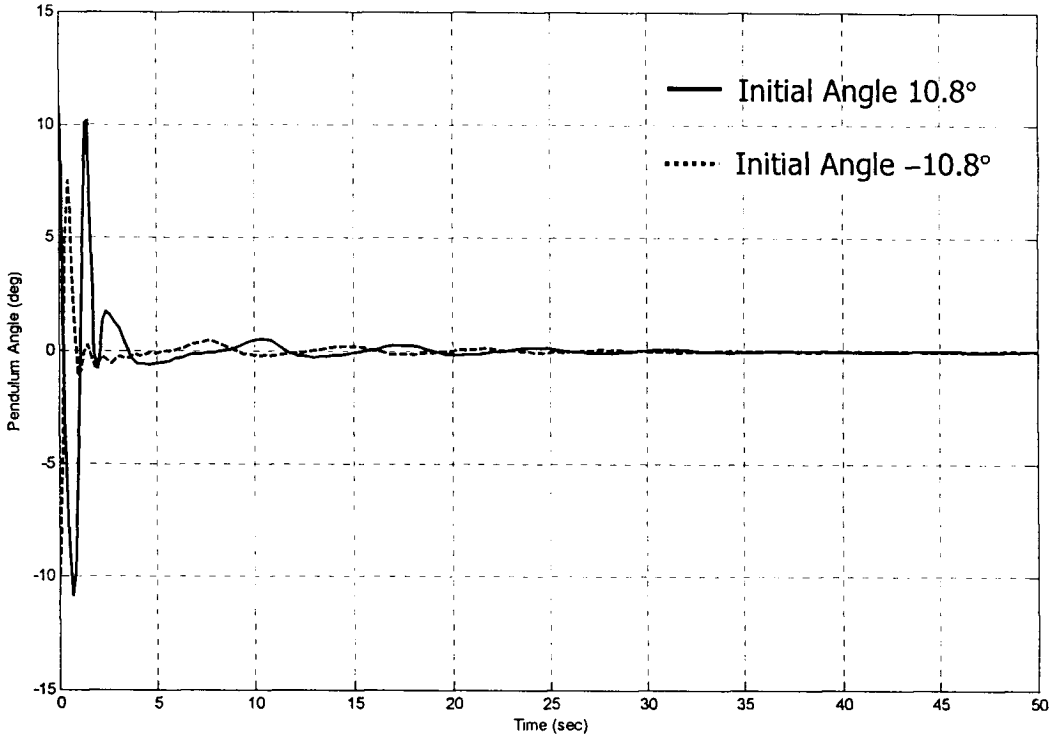


Fig. 5.10 Angular deviation of the Pendulum starting from initial pendulum angles on two sides of the vertical at ±10.8°.

5.1.4.5 Controller Robustness

One of the desirable attributes of a controller is its ability to perform under unforeseen conditions. These conditions may occur in a control system as some external disturbances, changes in plant parameters, or changes in plant operating conditions. A controller should be capable to operate the plant even under such conditions.

a) Disturbance Rejection

To demonstrate the disturbance rejection capability of a controller trained by the ADHDP based control scheme, an impulse input was applied to the cart at $t = 50$ secs, after the plant states had achieved steady state. A force of 10N was applied for 0.1 second at $t = 50$ secs to generate the impulse input. Figure 5.11 shows the corrective actions taken by the LS-SVM based controller to quickly restore the pole back to the stable condition on application of the disturbance. Throughout the simulation period, the SVM based tree type neural network, that acts as the critic, generates a zero output, indicating stable operation of the plant.

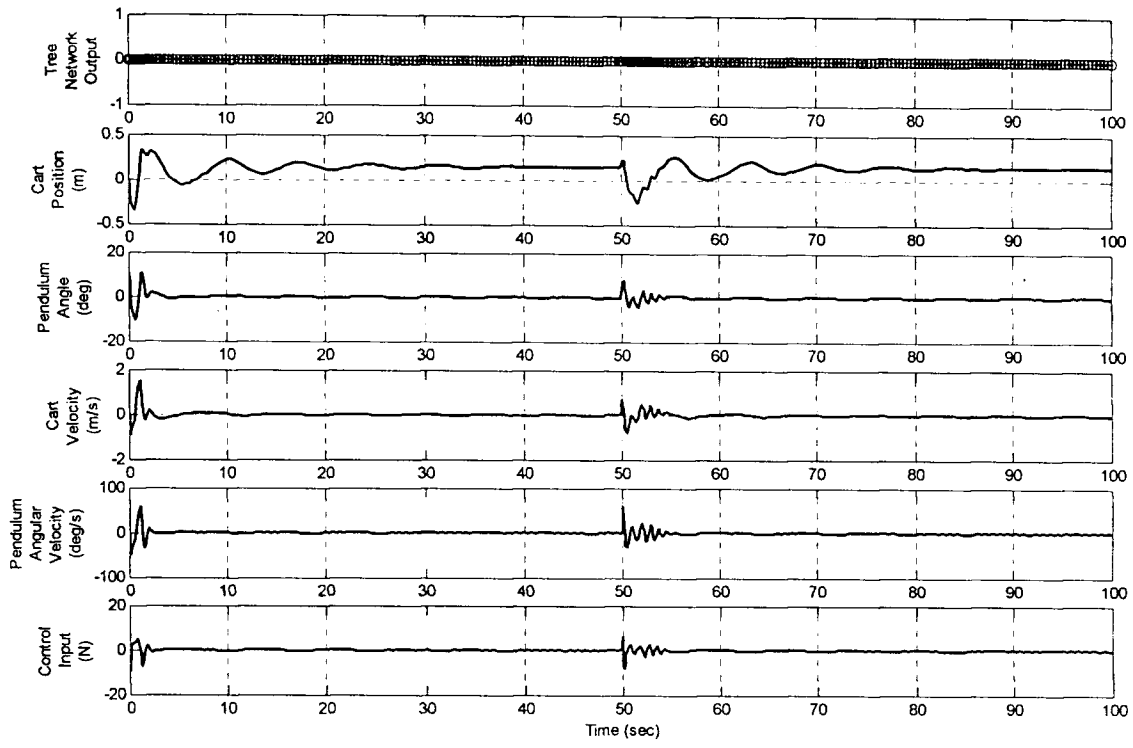


Fig. 5.11. Effect of disturbance due to an impulse input at the 50th second.

b) Change in Plant Parameters

To show the effect of change of plant parameters, the pole length was first reduced to half its original length. Figure 5.12 shows the simulation results of controlling such a pole by the trained ADHDP based controller. The trained controller takes prompt and smooth corrective action to restore the pole to the vertical position. In another test, to mimic the change in plant parameters, the mass of the cart was doubled. Figure 5.13 shows how the trained ADHDP based controller provides necessary control inputs that moves the cart both ways about the centre of the track to ultimately restore the pole to the vertical position.

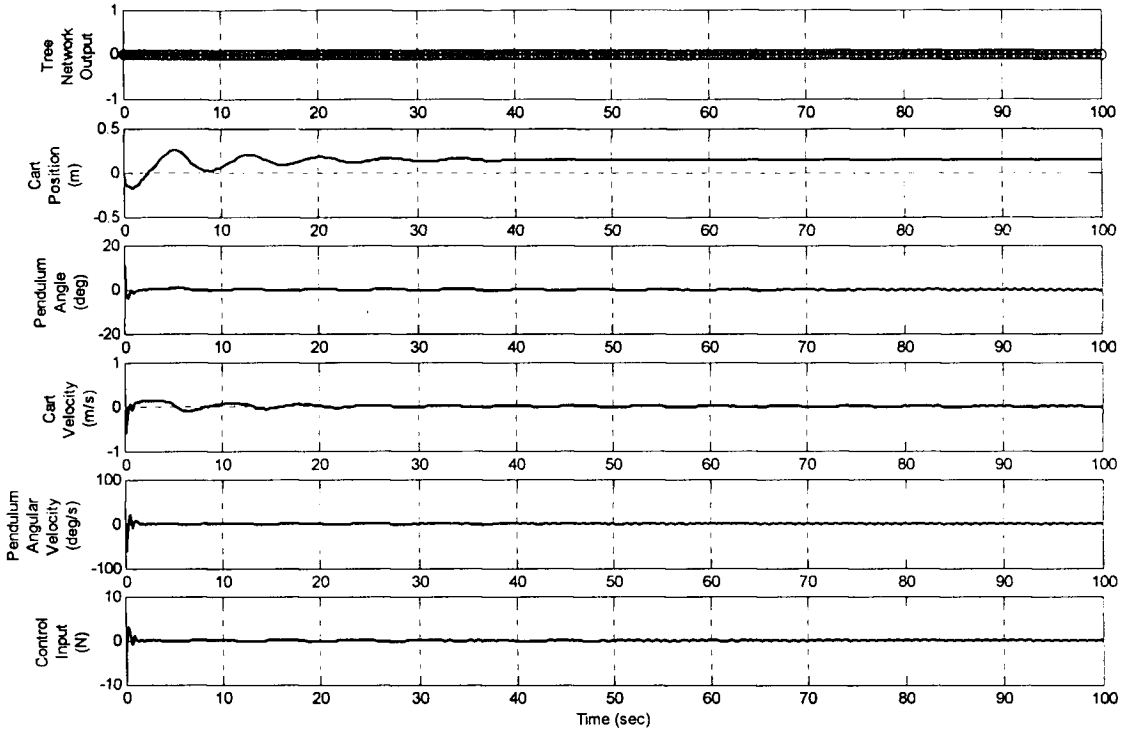


Fig. 5.12. Effect due to change in Plant parameters when the pole length is reduced to half.

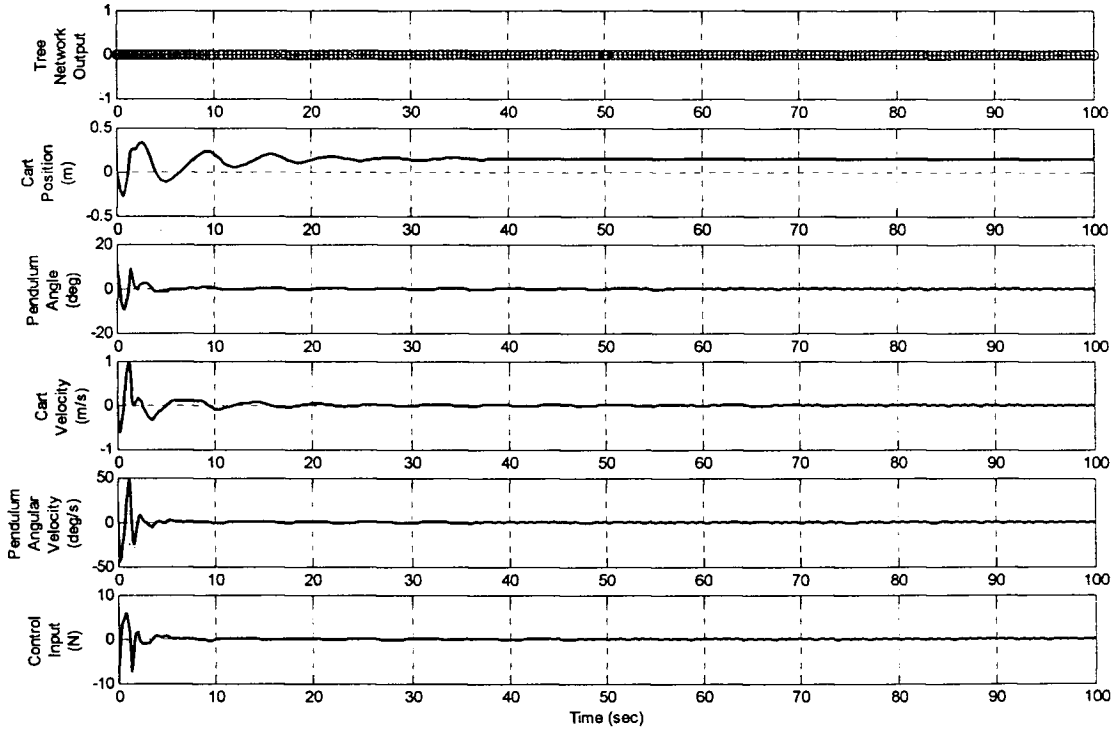


Fig. 5.13. Effect due to change in plant parameters when the mass of the cart is doubled.

These robustness tests show good control capabilities of the ADHDP based controller for appreciable changes in the plant parameters, and also demonstrate its disturbance rejection performance.

5.2 Induction Motor control using SVMs and SVM based networks

5.2.1 Introduction

The basic function of a variable speed drive (VSD) is to control the flow of energy from the mains to the process, supplied through the motor shaft. "Torque control" or "speed control" are ultimately required to control the two physical quantities, torque and speed, that describe the state of a shaft. When a VSD operates in the torque control mode, the load determines the speed. Likewise, when operated in the speed control mode, the load determines the torque. Initially, direct current (DC) motors were used as VSDs, because smooth control over the four-quadrants was possible by changing the field and armature currents to achieve the speed and torque requirements, without the need for sophisticated electronics. A general scheme of a VSD using a DC motor has been shown in Fig. 5.14a. The existence of a commutator and brushes are disadvantages of the use of DC motors, as they limit operation at higher speeds, necessitate periodic maintenance, and are a hazard in corrosive and explosive environments. Alternating current (AC) variable speed drive technology has been a much sought after alternative.

5.2.2 AC Drives

The evolution of AC variable speed drive technology has been partly driven by the desire to emulate the performance of the DC drive, such as fast torque response, speed and accuracy, while utilizing the advantages offered by the

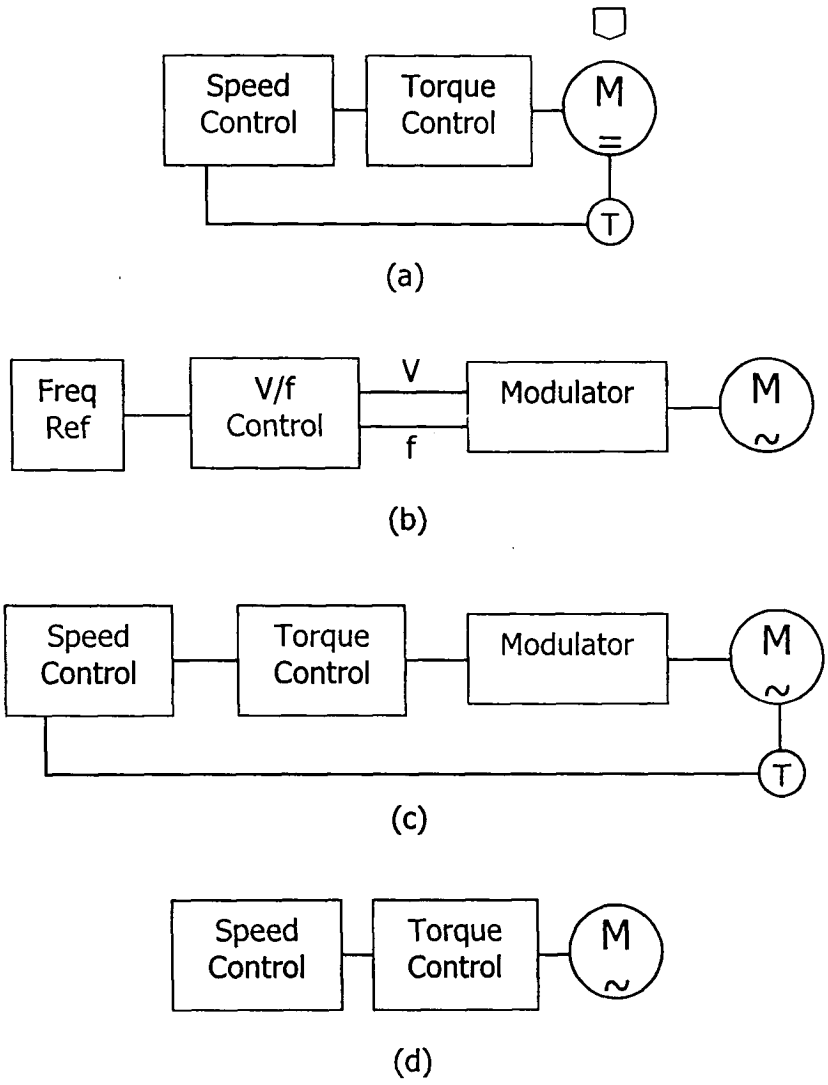


Fig. 5.14. Schematic diagrams of Variable Speed Drives. a) Control Loop of a DC Drive; b) Control Loop with Frequency Control; c) Control Loop with Flux Vector Control, and d) Control Loop of an AC Drive using Direct Torque Control.

standard AC motor. AC drive systems containing squirrel cage induction motors are advantageous because of their simple and rugged structure, and the cheap availability of induction motors at various power ratings. The use of static converters and torque control has provided AC VSDs a definite edge over high-performance four-quadrant DC drives.

5.2.2.1 AC Drives - Frequency control using Pulse Width Modulation

Unlike a DC drive, the AC drive frequency control technique shown in Fig. 5.14b uses parameters generated outside of the motor as controlling variables, namely voltage and frequency. Both voltage and frequency references are fed into a modulator that simulates an AC sine wave and feeds this to the motor's stator windings. This technique is termed Pulse Width Modulation (PWM) and utilizes the fact that there is a diode rectifier at the mains, and that the intermediate DC voltage is kept constant. The inverter controls the motor by applying a PWM pulse train dictating both the voltage and frequency. Significantly, this is an "open-loop drive", and does not feedback speed or position measurements from the motor's shaft into the control loop.

The features that make this AC drive advantageous over a conventional DC drive are its simpler circuit due to non-requirement of feedback devices, and hence a lower cost. Overall, it offers a simple and economical solution to controlling AC induction motors, that do not require high levels of accuracy or precision, such as those used in pumps and fans.

Although the use of AC drives (frequency control using PWM) has many advantages over the conventional DC drive, the associated drawbacks are non-utilization of the field orientation, lack of control of motor torque, and the use of a delaying modulator.

With this technique, sometimes known as scalar control, frequency and voltage are the main control variables that are applied to the stator windings

instead of the field orientation. The status of the rotor in terms of any feedback such as a speed or position signal is ignored. Therefore, torque cannot be controlled with any degree of accuracy.

5.2.2.2 AC Drives - Flux vector control using Pulse Width Modulation

To emulate the magnetic operating conditions of a DC motor, i.e. to perform the field orientation process, the flux-vector drive (Fig. 5.14c) needs to know the spatial angular position of the rotor flux inside the AC induction motor. With flux vector pulse width modulation (PWM) drives, field orientation is achieved by electronic means, instead of the mechanical commutator/brush assembly of the DC motor. This is a "closed-loop drive", as information about the rotor status is obtained by feeding back rotor speed and angular position relative to the stator field by means of a pulse encoder.

Furthermore, the motor's electrical characteristics are mathematically modeled and microprocessors are used to process the data. The electronic controller of a flux-vector drive creates electrical quantities such as voltage, current, and frequency, which are the controlling variables, and feeds these through a modulator to the AC induction motor. Thus torque is controlled indirectly.

The main advantages of this type of drive are good torque response, accurate speed control, and attainment of full torque at zero speed. Flux vector

control achieves full torque at zero speed, giving it a performance very similar to a DC drive.

The drawbacks associated with this type of AC drive are its high cost, and the requirement of modulator. It needs a feedback device to achieve a high level of torque response and speed accuracy, that makes the drive costly and brings in added complexity to the traditional simple AC induction motor. The use of a modulator slows down communication between the incoming voltage and frequency signals, and the motor needs to respond to this changing signal. Hence, although the motor is mechanically simple, the drive is electrically complex.

5.2.2.3 AC Drives - Direct Torque Control

With the revolutionary Direct Torque Control (DTC) technology (Fig.5.14d) developed by ABB [ABB, Tit96], field orientation is achieved without feedback by using advanced motor theory to calculate the motor torque directly, and without using modulation. The controlling variables are motor magnetizing flux and motor torque. With DTC, there is no modulator and no need of a tachometer or position encoder to feed back the speed or position of the motor shaft.

DTC uses the fastest processing hardware available and a more advanced mathematical understanding of how a motor works. The result is a drive with a torque response that is faster than any AC or DC drive. The dynamic speed accuracy of DTC drives is better than that of any open loop AC drive, and is

comparable to that of a DC drive that is using feedback. DTC produced the first "universal" drive with the capability to perform like either an AC or DC drive.

5.2.3 Comparison of VSDs and DTC Drive

A glance at Fig. 5.14 reveals the striking similarity between the control block of the DC drive (Fig. 5.14a) and the Direct Torque Control drive (Fig. 5.14d). Both schemes use motor parameters to directly control torque. However DTC has the added advantage of the absence of any feedback device. It offers all the benefits of an AC motor, and the absence of any external excitation. Table 5.3 compares the control variables of different drive systems.

Table 5.3: Comparison of Control Variables

Drive	Control Variables
DC Drives	Armature Current Field Current
AC Drives (PWM)	Output Voltage Output Frequency
Direct Torque Control (DTC)	Motor Torque Motor Magnetizing Flux

As can be seen from Table 5.3, both DC and DTC drives use actual motor parameters to control torque and speed. Thus, the dynamic performance is fast and easy. For most applications, DTC doesn't need feedback mechanisms such as a tachometer or encoder for obtaining speed or position signal. When compared with the two other AC drive control blocks shown in Figs. 5.14b and 5.14c, DTC, shown in Fig 5.14d shows up several differences. The main one is that no modulator is required with DTC. With PWM AC drives, the controlling variables are frequency and voltage, that need to go through several stages

before being applied to the motor. Thus, with PWM drives, control is handled inside the electronic controller and not inside the motor.

5.2.3.1 DTC Drive

Direct-torque control (DTC) is one of the most recent types of drive application. ABB, a well known electrical equipment manufacturer, have made commercially available DTC drives. DTC has been claimed to be a very innovative AC motor control method [ABB, Tit96].

In this scheme, control of the stator flux linkage and the electromagnetic torque of a DTC induction motor drive is possible by the selection of optimum inverter switching modes.

If, in the stationary reference frame of a three-phase induction machine, $\overline{\psi}_s$ denotes the stator flux linkage space vector and $\bar{i}_s$ denotes the stator-current space vector, the instantaneous electromagnetic torque is given by

$$t_e = \frac{3}{2} P \overline{\psi}_s \times \bar{i}_s . \tag{5.15}$$

Expressing $\overline{\psi}_s$ and $\bar{i}_s$ in polar form, we have

$$\overline{\psi}_s = |\overline{\psi}_s| e^{j\rho_s} , \tag{5.16}$$

$$\bar{i}_s = |\bar{i}_s| e^{j\alpha_s} , \tag{5.17}$$

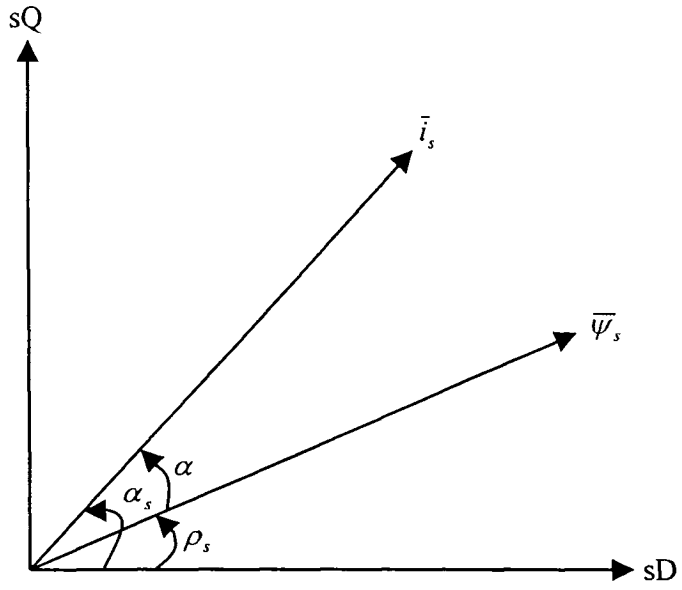


Fig. 5.15. Stator flux-linkage and stator current space vectors.

where ρ_s and α_s are the angle of the stator flux-linkage space vector, and stator-current space vector with respect to the direct-axis of the stator reference frame respectively. Equation (5.15) can be rewritten in the following form

$$\begin{aligned} t_e &= \frac{3}{2} P |\overline{\psi}_s| |\bar{i}_s| \sin(\alpha_s - \rho_s) \\ &= \frac{3}{2} P |\overline{\psi}_s| |\bar{i}_s| \sin(\alpha) \end{aligned} \quad , \tag{5.18}$$

where $\alpha = \alpha_s - \rho_s$ is the angle between the stator flux-linkage and stator-current space vector as shown in Fig. 5.15.

It can be shown [Vas98], that if the modulus of the stator flux-linkage space vector is kept constant, and the angle ρ_s is rapidly changed, the electromagnetic torque t_e can be rapidly changed. In other words, quick torque control can be performed by keeping the stator voltages (and hence stator flux)

constant and quickly rotating the stator flux linkage space vector to the position required by the torque demand. If the actual developed electromagnetic torque is less than the reference value, the electromagnetic torque should be increased as fast as possible by using the maximum possible value of $\frac{d\rho_s}{dt}$. If $\overline{\psi}_s$ is accelerated in the forward direction, positive electromagnetic torque is produced, while if it is decelerated backward, negative electromagnetic torque is produced. Thus, in effect, instantaneous torque control can be achieved by quickly changing the position of $\overline{\psi}_s$, or by quickly changing its speed. This type of control is referred to as direct torque control.

5.2.3.2 DTC schematic diagram

Figure 5.16 shows a general block diagram of a DTC that can be divided into two major sections: the Torque Control Loop and the Speed Control Loop. The different functions of the block diagram are explained as below:

i) Measurement of Voltage and Current

For normal operation, measurements of the two motor phase currents and the DC bus voltage as well as the switch positions of the inverter are taken into account.

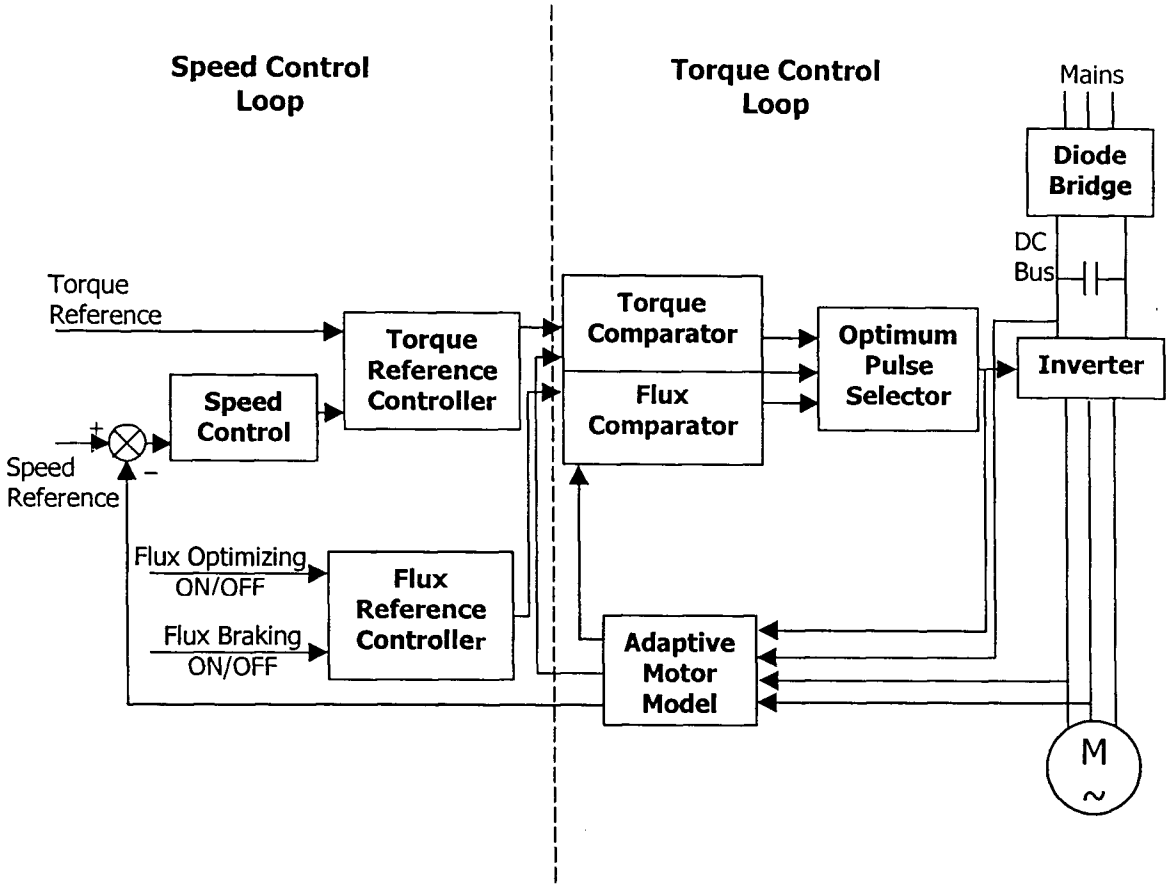


Fig. 5.16. Block diagram of the Direct Torque Control for a three-phase Induction Motor.

ii) Adaptive Motor Model

Measured information from the motor is fed to the adaptive motor model. The sophistication of this motor model allows precise data about the motor to be calculated. Before operating the DTC drive, prior information about the motor is collected during a motor identification run and fed into the motor model. This is called auto-tuning. Data such as stator resistance, mutual inductance, and saturation coefficients are determined along with the motor's inertia. The motor model outputs control signals that directly represent actual motor torque and actual stator flux. Shaft speed is also calculated within the motor model.

iii) Torque Comparator and Flux Comparator

The information to control power switches is produced at the Torque and Flux Comparator. Both actual torque and actual flux are fed to the comparators where they are compared to a torque and flux reference value. Torque and flux status signals are calculated by a two level hysteresis control method. These signals are then fed to the Optimum Pulse Selector.

iv) Optimum Pulse Selector

The Optimum Pulse Selector is used to determine the switching logic of the inverter. Ideally it can be built with digital signal processors (DSP), together with ASIC hardware. The correct switch combination is determined for every control cycle. There is no predetermined switching pattern. DTC has been referred to as "just-in-time" switching, because unlike traditional PWM drives where up to 30% of all switch changes are unnecessary, with DTC, each and every switching is needed and used. This high speed of switching is fundamental to the success of DTC. The main motor control parameters are updated several times a second. This allows extremely rapid response on the shaft and is necessary so that the motor model can update this information. It is this processing speed that yields high performance figures including a static speed control accuracy of ±0.5% and a torque response of less than 2ms, without the use of an encoder.

v) Torque Reference Controller

Within the Torque reference controller, the speed control output is limited by the torque limits and DC bus voltage. It also includes speed control for cases when an external torque signal is used. The internal torque reference from this block is fed to the Torque Comparator.

vi) Speed Controller

The Speed Controller block consists both of a PID controller and an acceleration compensator. The external speed reference signal is compared to the actual speed produced in the Motor Model. The error signal is then fed to both the PID controller and the acceleration compensator. The output is the sum of outputs from both of them.

vii) Flux Reference Controller

An absolute value of stator flux can be given from the Flux Reference Controller to the Flux Comparator block. The ability to control and modify this absolute value provides an easy way to realize many inverter functions such as Flux Optimization and Flux Braking.

5.2.3.3 Drawbacks of DTC

The drawbacks of DTC are sluggish response in both start up and due to changes in flux or torque, necessity of perfect knowledge of the motor model

and dynamics, difficulty to determine the parameters for flux estimation, and use of "sophisticated tables for state selection" [Vas98].

5.2.4 DTC and Machine Learning Approaches

The conventional DTC has some disadvantages as discussed above. The function approximation property of neural network can be used to overcome some of its drawbacks. Cabrera et al. [Cab97] uses neural networks to emulate the state selector of DTC by using training algorithms like backpropagation, adaptive neuron model, extended Kalman filter and the parallel recursive prediction error algorithms. A small structured neural network speed estimator for induction motors using direct torque control (DTC) has been used by Cruz et al. [Cru2K1], that was effective and gives satisfactory results in a closed loop DTC scheme. Cirrincione et al. [Cir2K3] uses a General Mapping Regressor (GMR) neural network for the direct torque control of an induction motor.

The function approximation properties of neural networks may be used for DTC in two ways:

a) Approximating the state selector function using an artificial neural network (ANN) so that the use of "sophisticated look up tables" [Vas98] is avoided. The ANN is trained to learn the complicated relation between the voltage vector and the error of the stator flux and output torque. After being well trained, it is used as the switching status selector for the DTC of an induction motor fed by the three-level VSI.

b) Using an ANN to estimate the flux and torque from the current and voltage values. This eliminates the requirement for values of the motor parameters, and effect due to variation in the parameters.

5.2.5 Simulation results

5.2.5.1 Induction Motor parameters

To test the above control strategy, the model of a three-phase induction motor was considered, whose different parameter values are given in Table 5.4. This motor model that was built and used in [Kum2K4b] has been used in our work.

Table 5.4: Induction Motor Parameters

Parameter	Value
Power rating, P	1 kw
Moment of Inertia, J	0.001 kg-m^2
Magnetizing Inductance, L_m	0.678 H
Rotor Inductance, L_r	0.7131 H
Stator Inductance, L_{sc}	0.0703 H
Rotor resistance, R_r	11.72 Ω
Stator resistance, R_s	9.45 Ω

5.2.5.2 Results

In our work, we have adopted method (a) described in Section 5.2.4. The SIMULINK model for DTC and the training data stored in a .mat file for speed control of the induction motor has been adopted from Kumar et al. [Kum2K4b].

If V_{sd} and V_{sq} respectively denote the direct and quadrature axis voltages, and I_{sd} and I_{sq} denote the direct and quadrature axis currents of modeling the motor, then an input data vector is of the form

$$[V_{sd} \quad V_{sq} \quad I_{sd} \quad I_{sq}]^T. \tag{5.19}$$

If S_a, S_b and S_c represent the inverter switchings of the three phases 'a', 'b' and 'c', then the desired output data vector is of the form

$$[S_a \quad S_b \quad S_c]^T. \tag{5.20}$$

The ON-OFF state of the inverter switchings can be considered as a binary signal, and in actual implementation, 1 and –1 values were used.

Kumar et al. [Kum2K4b] used the training data with binary valued outputs to train a LS-SVM classifier for the speed control problem to achieve a desired speed of 300 rads/s.

In our work, we have used the same motor model and training data used in [Kum2K4b]. We have used the SVM based tree type neural network to learn the switching sequences for achieving the target speed. In actual implementation, all the input variables V_{sd}, V_{sq}, I_{sd}, and I_{sq} were normalized to lie between 0 to 1. Figure 5.17 shows the variation of speed, 3-ph current, electromagnetic torque, direct and quadrature axis fluxes, and flux magnitude to achieve the desired speed.

5.3 Conclusion

This chapter deals with control applications of tree networks developed using a LP formulation. In the first application, a SVM based tree type neural network has been used as the critic in a ADHDP strategy for control. The critic's target of learning binary mapping has been formulated in a linear programming

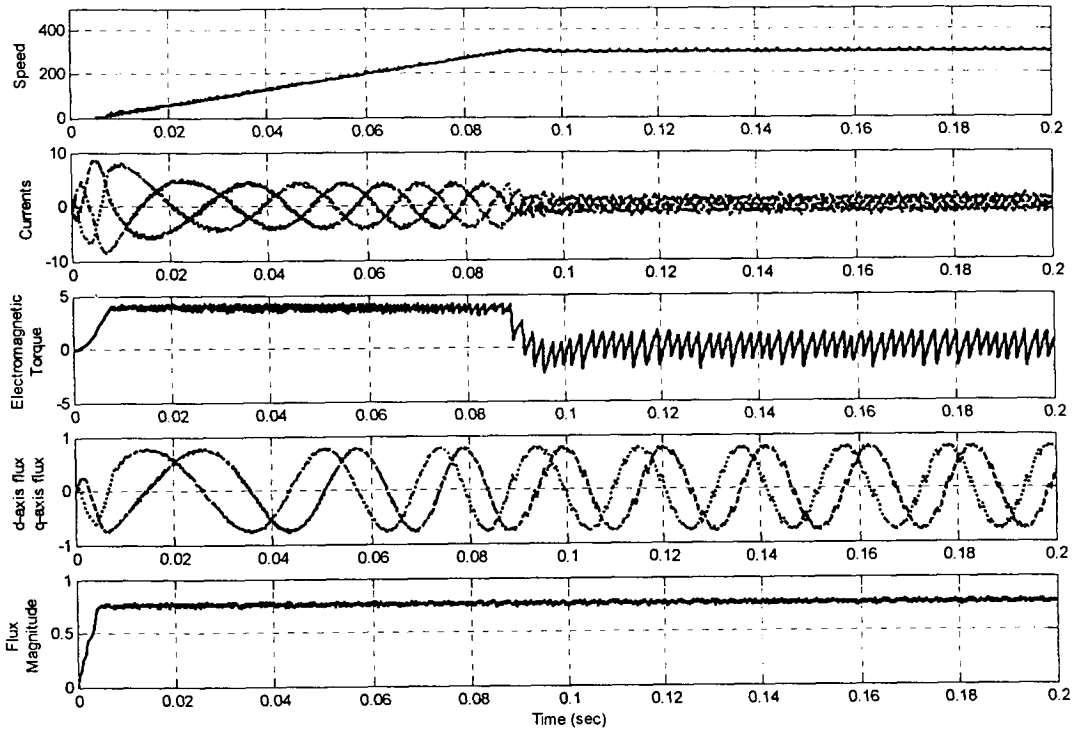


Fig. 5.17. Tracking performance by generation of the switching sequences by the SVM based tree type neural network.

framework. Advantages of linear programming based approaches are their simplicity, the possibility of parallelism, and the fast potential for learning. The controller is LS-SVM regressor with a Gaussian kernel. The inverted pendulum model was chosen as the plant for testing the control strategy. This type of control scheme offers an effective way of "failure avoidance control", where the LS-SVM controller provides near optimal control for a highly non-linear plant, while the SVM based tree type neural network provides fault tolerance to the control system. Nonetheless, the ability of the trained controller to reject disturbances and also to manoeuvre the plant under adverse conditions of plant parameter changes demonstrates its robust performance.

In the second part of the work, the SVM based tree type neural network was used for learning the inverter switching sequences for Direct Torque Control of an induction motor, with the direct and quadrature axes voltages and currents acting as the inputs. Accurate tracking during speed up was observed, culminating in the desired speed being achieved.

Chapter 6

Conclusion

In this thesis, we have proposed techniques for building constructive networks for the binary classification problem, and for function approximation. New network training algorithms for binary classification and function approximation in one-dimension has been proposed. A novel technique for regression has been devised and its generalization performance compared on synthetic and real data sets.

Some of these approaches are based on linear programming that makes problem formulation simpler. Efficient algorithms [Ign94, Wis71, Las70, Bra2K] are available for solving linear programming problems in a reasonable time. By using linear programming solutions, the decision region can be made piecewise smooth. To improve the generalization capabilities of the linear hyperplanes, use is made of SVMs that not only minimizes the empirical error, but also the structural risk when generating the hypothesis.

A SVM based tree type neural network has been proposed, where each of the neurons acts as a maximum-margin type SVM, optimizing the margin of separation between the two classes of its training set. Its ability to learn binary sequences [Jay2K2c] has also been shown. Two new types of objective functions in terms of the slack variables have been proposed for the linear-programming

based approaches. SVM N-tree, a training algorithm for the SVM based tree type neural network has been proposed that uses the training set and a validation set for better generalization. Performance of a pruned SVM based tree structured neural network has also been compared with other prevalent algorithms.

A method for building a tree type neural network for approximating one-dimensional functions using piecewise linear (PWL) elements has been proposed. A new support vector regression technique, PPSVR has been developed, that requires only a single matrix inversion, and is therefore computationally more efficient than other reported methods in the literature. Its performance on artificial and benchmark data sets has been evaluated. Linear programming based methods for performing matrix inversion are available, and may be used in the PPSVR approach.

The SVM N-tree network has been used as the critic for an Action Dependant Heuristic Dynamic Programming (ADHDP) approach for control in the benchmark pole-balancing problem. The SVM based binary classification network was capable of learning the failure data, where the number of failure states are far smaller than the number of healthy states. The SVM based tree type neural network has also been used for generating the switching signals for the direct torque control of a three-phase induction motor.

Like the piecewise linear neurons used to perform binary classification and one-dimensional function approximation, efforts can be made to develop the same functionality using non-linear activation functions, that promises to be more powerful in terms of the number of neurons and generalization capability. The PPSVR approach holds considerable promise as far as its computational complexity is concerned, but its efficiency depends on the suitability of the function that maps the input to some higher dimensional feature space. The efficiency of PPSVR can be augmented by devising ways for finding a suitable mapping function for a function approximation task, and developing schemes for finding the optimum value of its parameters. Matrix inversion using linear programming may be incorporated into the PPSVR approach. If these issues are efficiently implemented, PPSVR holds much promise for online and real time implementation.

References

[ABB] ABB Technical Guide No. 1- Direct Torque Control. http://www.slater-drives.com/user_manuals.html

[Alo90] N. Alon and N. Megiddo, "Parallel Linear Programming in Fixed Dimension Almost surely in Constant Time", *Procs. of the 31st Annual Symposium on Foundations of Computer Science*, Vol. 2, pp. 574-582, 1990.

[Ama99] S. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions," *Neural Networks*, Vol.12, pp. 783-789, 1999.

[Bar64] R.G. Bartle, "*The Elements of Real Analysis*", John Wiley & Sons, Inc., 1964.

[Ben92] K.P. Bennett and O.L. Mangasarian, "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", *Optimization Methods and Software*, Vol. 1, pp. 23-34, 1992.

[Ben94] K.P. Bennett and O.L. Mangasarian, "Multicategory Discrimination via Linear Programming", *Optimization Methods and Software*, 3, pp. 27-39, 1994.

[Ben95] R. Ben Yishai, R. Lev Bar-or, and H. Sompolinsky, "Theory of orientation tuning in visual cortex', *Proc. Natl. Acad. Sci.* USA, Vol. 92, pp. 3844-3848, April 1995.

[Bla] C.L. Blake and C.J. Merz, "UCI Repository for Machine Learning databases" Irvine, CA: University of California, Department of Information and Computer Sciences.
On-line at http://www.ics.uci.edu/~mlearn/MLRepository.html

[Bos98] N.K. Bose and P. Liang, "*Neural Network Fundamentals with Graphs, Algorithms and Applications*", Tata McGraw-Hill Publishing Company Limited, New Delhi, 1998.

[Bra2K] P.S. Bradley and O.L. Mangasarian, "Massive Data Discrimination via Linear Support Vector Machines," *Optimization Methods and Software*, Vol. 13, No. 1, pp. 1-10, 2000.

[Bre84] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, "*Classification and Regression Trees*", Chapman & Hall.

[Bre93] L. Breiman, "Hinging Hyperplanes for Regression, Classification and Function Approximation", *IEEE Trans. on Information Theory*, Vol. 39, No. 3, pp. 999-1013, 1993.

[Bul2K3] J.A. Bullinaria, "Evolving efficient learning algorithms for binary mappings", *Neural Networks*, 16, pp. 793-800, 2003.

[Cab97] L.A. Cabrera, M.E. Elbuluk and D.S. Zinger, "Learning Techniques to Train Neural Networks as a State Selector for Inverter-Fed Induction Machines Using Direct Torque Control", *IEEE Trans. on Power Electronics*, Vol. 12, No. 5, pp. 788-799, 1997.

[Cas97] G. Castellano, A.M. Fanelli and M. Pelillo, "An Iterative Pruning Algorithm for Feedforward Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 8, No. 3, pp. 519-531, 1997.

[Cha2K2] O. Chapelle, V. Vapnik, O. Bousquet and S. Mukherjee, "Choosing Multiple Parameters for Support Vector Machines", *Machine Learning*, 46(1), pp. 131-159, 2002.

[Cha88] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units", *Advances in Neural Information Processing Systems (NIPS)*, Vol. 1, pp. 519-526, 1988.

[Cha97] C.W. Ou and S. Ranka, "Parallel Incremental Graph Partitioning", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 8, No. 8, pp. 884-896, 1997.

[Cho92] C.H. Choi and J.Y. Choi, "Construction of neural networks to approximate arbitrary continuous functions of one variable", *Electronics Letters*, Vol. 28, No. 2, pp. 151-153, 1992.

[Cho94] C.H. Choi and J.Y. Choi, "Constructive Neural Networks with Piecewise Interpolation Capabilities for Function Approximation", *IEEE Trans. on Neural Networks*, Vol. 5, No. 6, pp. 936-944, 1994.

[Cir2K3] G. Cirrincione, M. Cirrincione, C. Lu and M. Pucci, "Direct Torque Control of Induction Motors by Use of the GMR Neural Network", *Procs. of IJCNN-2003*, Vol. 3, pp. 2106-2111.

[Cri2K] N. Cristianini and J. Shawe-Taylor, "*An Introduction to Support Vector Machines and other kernel based learning methods*," Cambridge University Press, 2000.

[Cri98] N. Cristianini, C. Campbell and J. Shawe-Taylor, "Dynamically Adapting Kernels in Support Vector Machines", *Advances in Neural Information Processing Systems (NIPS)*, Vol. 11, pp. 204-210, 1998.

[Cru2K1] P.P. Cruz and J.J.R. Rivas, "A Small Neural Network structure application in speed estimation of an Induction Motor using direct torque control", *IEEE Power Electronics Specialists Conference (PESC)*, Vol. 2, pp. 823-827, 2001.

[Cun90] Y.L. Cun, J.S. Denker and S.A. Solla, "Optimal Brain Damage", *Advances in Neural Information Processing Systems (NIPS)*, Vol.3, pp. 598-605, 1990.

[Cyb89] G. Cybenko, "Approximation by superposition of a sigmoidal function", *Math. Control Signals Syst.*, 2, pp. 303-314, 1989.

[Deb2K3] Deb A.K., Jayadeva, Chandra S, and Gopal M., "Modified Growth Network for Pattern Classification", Operations Research And Its Applications: Recent Trends (*APORS* 2003). In M.R Rao and M.C. Puri (Eds.), Allied Publishers Pvt Limited, Vol I, pp. 270-277.

[Del] Data for Evaluating Learning in Valid experiments
Online Available: http://www.cs.utoronto.edu/~delve/

[Don2K3] M. Dong, Y. Li and R. Kothari, "Theoretical Results on a Measure of Classification Complexity", *Procs. of ICAPR-2003*, Kolkata, India.

[Dra96] T. Draelos and D. Hush, "A Constructive Neural Network Algorithm for Function Approximation", *Procs. of IJCNN-1996*, Vol. 1, pp. 50-55, 1996.

[Dud2K3] R.O. Duda, P.E. Hart and D.G. Stork, "*Pattern Classification (Second edition.)*", John Wiley & Sons, Inc, New York.

[Epp99] W. Eppler and H.N. Beck, "Piecewise Linear Networks (PLN) for Function Approximation", *Procs. of IJCNN-1999*, Vol. 1, pp. 388-391, 1999.

[Esp2K] A. Esposito, M. Marinaro, D. Oricchio and S. Scarpetta, "Approximation of continuous and discontinuous mappings by a growing neural RBF-based algorithm", Neural Network, 13, 651-665, 2000.

[Fah89] S.E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture", *Advances in Neural Information Processing Systems (NIPS)*, Vol. 2, pp. 524-532, 1989.

[Fed] Feedback MATLAB Based Control Products-Digital Pendulum Control Systems, Feedback Instruments Limited, Park Road, Crowborough, E Sussex, TN6, 2QR, UK.

[Fra98] E. Frank, Y. Wong, S. Inglis, G. Holmes and I.H. Witten, "Using Model Trees for Classification," *Machine Learning*, Vol. 32, pp. 63-76, 1998.

[Fre90] M. Frean, "The Upstart Algorithm: a method for constructing and training feed-forward neural networks", *Neural Computation*, 2, pp. 198-209, 1990.

[Fun2K1] G. Fung and O.L. Mangasarian, "Proximal Support Vector Machine Classifiers", *Proceedings KDD-2001*, San Francisco August 26-29, 2001. Association for Computing Machinery, New York, 2001, pp. 77-86.

[Gol89] A.V. Goldberg, S.A. Plotkin, D.B. Shmoys and E. Tardos, "Interior-Point Methods in Parallel Computation", *Procs. of the 30th Annual Symposium on Foundations of Computer Science*, pp. 350-355, 1989.

[Gol96] Golub and Van Loan, "Matrix Computations", The John Hopkins Univ Press, Baltimore, Maryland, 3rd ed, 1996, pp. 51.

[Gop97] M. Gopal, "Chapter 8: Lyapunov Stability Analysis", Digital Control and State Variable Methods, Tata McGraw Hill Publishing Company Limited, New Delhi.

[Gun98] S.R. Gunn, "Support Vector Machines for Classification and Regression," *Technical Report*, School of Electronics and Computer Science, University of Southampton, Southampton, U.K, 1998.
On-line at http://www.isis.ecs.soton.ac.uk/resources/svminfo/

[Hah98] R.L.T. Hahnloser, "On the piecewise analysis of networks of linear threshold neurons", *Neural Networks*, 11, pp. 691-697, 1998.

[Has92] B. Hassibi and D.G. Stork, "Second order derivatives for network pruning: Optimal Brain Surgeon", *Advances in Neural Information Processing Systems (NIPS)*, Vol. 5, pp. 164-171, 1992.

[Has93] B. Hassibi, D.G. Stork and G. Wolff, "Optimal Brain Surgeon: Extensions and performance comparisons", *Advances in Neural Information Processing Systems (NIPS)*, Vol. 6, pp. 263-270, 1993.

[Hay98] S. Haykin, "*Neural Networks: A Comprehensive foundation (2nd edition)*", Prentice Hall, USA, 1998.

[Her91] J. Hertz, A. Krogh and R.G. Palmer, "*Introduction to the Theory of Neural Computation*", Addison-Wesley Publishing Company, USA, 1991.

[Hoc2K4] S. Hochreiter and K. Obermayer "*Classification, regression, and feature selection on matrix data*", Technical report, Technische Universität Berlin, Fakultät für Elektrotechnik und Informatik, 2004.
Http://ni.cs.tu-berlin.de/publications/ni-pubs-2004.html

[Hor89] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, 2, pp. 359-366, 1989.

[Hus98] D.R. Hush and B. Horne, "Efficient Algorithms for Function Approximation with Piecewise Linear Sigmoidal Networks", *IEEE Trans. on Neural Networks*, Vol. 9, No. 6, pp. 1129-1141, 1998.

[Ign94] J.P. Ignizio and T.M. Cavaliar, "*Linear Programming*," Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[Jay2K1] Jayadeva, "From Neurons to Networks: Back to Square One", *Biophysical Processes in Living Systems* (ed. P. P. Saradhi), pp. 343-356, 2001.

[Jay2K2a] Jayadeva, A.K. Deb and S. Chandra, "Algorithm for building a neural network for function approximation", *IEE Proc.-Circuits, Devices Syst.*, Vol. 149, No. 5/6, pp. 301-307, 2002.

[Jay2K2b] Jayadeva, A.K. Deb and S. Chandra, "Binary Classification by SVM Based Tree Type Neural Networks," *Proceedings of IJCNN-2002*, Vol. 3, pp. 2773-2778, Honolulu, Hawaii, USA, May 12-17, 2002.

[Jay2K2c] Jayadeva and Alok Kanti Deb, "Learning Sequences using Tree Type neural networks", *Proceedings of the Sixth International Conference of Cognitive and Neural Systems*, Dept of Cognitive and Neural Systems, Boston University, USA, May 30-June 1, 2002.

[Joa99] T. Joachims, "Making Large-Scale Support Vector Machines Practical," "*Advances in Kernel Methods-support vector learning*," (edited by B Schölkopf, C.J.C. Burges and A.J. Smola), pp. 169-184, The MIT Press, 1999.

[Kam97] N.S. Kambo, "*Mathematical Programming Techniques*", Affiliated East-West Press Pvt Ltd, New Delhi, 1991.

[Kar90] Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks", *IEEE Trans. on Neural Networks*, Vol.1, No. 2, pp. 239-242, 1990.

[Kee2K2] S.S. Keerthi, "Efficient Tuning of SVM Hyperparameters Using Radius/ Margin Bound and Iterative Algorithm", *IEEE Trans. on Neural Networks*, Vol. 13, No. 5, pp. 1225-1229, 2002.

[Kla2K] D. Klabjan, E.L. Johnson and G.L. Nemhauser, "A parallel primal-dual simplex algorithm", *Operations Research Letters*, 27, pp. 47-55, 2000.

[Kum2K4a] S. Kumar, "*Neural Networks-A Classroom Approach*", Tata McGraw Hill, New Delhi, India, 2004.

[Kum2K4b] G. Kumar and K. Malhotra, "Neural Network Application of variable speed drives", *B. Tech dissertation*, Dept. of Electrical Engineering, IIT Delhi, 2004.

[Kwo97a] T.Y. Kwok and D.Y. Yeung, "Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems", *IEEE Trans. on Neural Networks*, Vol. 8, No. 3, pp. 630-645, 1997.

[Kwo97b] T.Y. Kwok and D.Y. Yeung, "Objective Functions for Training New Hidden Units in Constructive Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 8, No. 5, pp. 1131-1148, 1997.

[Las70] L.S. Lasdon, "*Optimization Theory for Large Systems*," The MacMillan Company, 1970.

[Lin] http://www.lindo.com/downloads/downloadm.html

[Liu2K2] D. Liu, "Adaptive Critic Designs for Problems with Known Analytical Form of Cost Function", *Proceedings of IJCNN-2002*, Vol. 3, May 12-17, 2002, Honolulu, Hawaii, USA, pp.1808-1813.

[Lu99] B.L. Lu, H. Kita and Y. Nishikawa, "Inverting Feedforward Neural Networks Using Linear and Nonlinear Programming", *IEEE Trans. on Neural Networks*, Vol. 10, No. 6, pp. 1271-1290, 1999.

[Lus96] I.J. Lustig and E. Rothberg, "Gigaflops in linear programming", *Operations Research Letters*, 18, pp. 157-165, 1996.

[Ma2K3] L. Ma and K. Khorasani, "A new strategy for adaptively constructing multilayer feedforward neural networks", *Neurocomputing*, 51, pp. 361-385, 2003.

[Man2K2] C.J. Mantas, J.M.M. Ruiz and F. Rojas, "A procedure for improving generalization in classification trees," *Neurocomputing.*, Vol. 48, pp. 727-740, 2002.

[Man93] O.L. Mangasarian, "Mathematical Programming in Neural Networks", *ORSA Journal of Computing*, Vol. 5, No. 4, pp. 349-360, 1993.

[Mar90] G. Martinelli, L.P. Ricotti, S. Ragazzini and F.M. Mascioli, "A Pyramidal Delayed Perceptron", *IEEE Trans. on Circuits and Systems*, Vol. 37, No. 9, pp. 1176-1181, 1990.

[Mas95] F.M.F. Mascioli and G. Martinelli, "A Constructive Algorithm for Binary Neural Networks: The Oil-Spot Algorithm", *IEEE Trans. on Neural Networks*, Vol. 6, No. 3, pp. 794-797, 1995.

[MAT] http://www.mathworks.com

[Moz88] M.C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment", *Advances in Neural Information Processing Systems (NIPS)*, Vol.1, pp. 107-115, 1988.

[Mus95] M. Muselli, "On Sequential Construction of Binary Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 6, No. 3, pp. 678-690, 1995.

[Par2K] R. Parekh, J. Yang and V. Honavar, "Constructive Neural-Network Learning Algorithms for Pattern Classification", *IEEE Trans. on Neural Networks*, Vol. 11, No. 2, pp. 436-451, 2000.

[Pel] K. Pelckmans, J.A.K. Suykens, T.V. Gestel, J.D. Brabanter, L. Lukas, B. Hamers, B.D. Moor and J. Vandewalle, "LS-SVMlab Toolbox User's Guide" http://www.esat.kuleuven.ac.be/sista/lssvmlab/

[Pre97] L. Prechelt, "Investigation of the CasCor Family of Learning Algorithms", *Neural Networks*, Vol. 10, No. 5, pp. 885-896, 1997.

[Pro95] D.V. Prokhorov, R.A. Santiago and D.C. Wunsch II, "Adaptive Critic Designs: A Case Study for Neurocontrol", *Neural Networks*, Vol. 8, No. 9, pp. 1367-1372, 1995.

[Pro97a] D.V. Prokhorov, "*Adaptive Critic Designs and Their Applications*", Ph.D dissertation, Department of Electrical Engineering, Texas Tech University, October, 1997.

[Pro97b] D.V. Prokhorov and D.C. Wunsch II, "Adaptive Critic Designs", *IEEE Trans. on Neural Networks*, Vol. 8, No. 5, pp. 997-1007, 1997.

[Ree93] R. Reed, "Pruning Algorithms-A Survey", *IEEE Trans. on Neural Networks*, Vol. 4, No. 5, pp. 740-747, 1993.

[Roy95] A. Roy, S. Govil and R. Miranda, "An Algorithm to Generate Radial Basis Function (RBF)-Like Nets for Classification Problems", *Neural Networks*, Vol. 8, No. 2, pp. 179-201, 1995.

[Roy97a] A. Roy, S. Govil and R. Miranda, "A Neural-Network Learning Theory and a Polynomial Time RBF Algorithm", *IEEE Trans. on Neural Networks*, Vol. 8, No. 6, pp. 1301-1313, 1997.

[Roy97b] A. Roy and S. Mukhopadhyay, "Iterative Generation of Higher-Order Nets in Polynomial Time Using Linear Programming," *IEEE Trans. on Neural Networks*, Vol. 8, No. 2, pp. 402-412, 1997.

[San91] T.D. Sanger, "A Tree-Structured Adaptive Network for Function Approximation in High-Dimensional Spaces", *IEEE Trans. on Neural Networks*, Vol. 2, No. 2, pp. 285-293, 1991.

[Sau98] C. Saunders, A. Gammerman and V. Vovk, "Ridge Regression Learning Algorithm in Dual Variables", *Procs. of the 15th International Conference on Machine Learning, ICML-98*, pp. 515-521.

[Sch2K2] B. Schölkopf and A.J. Smola, "Learning with Kernels-Support Vector Machines, Regularization, Optimization and Beyond", MIT Press, Cambridge, MA, 2002.

[Sch2K2] A. Schwaighofer, "SVM Toolbox for MATLAB". http://www.igi.tugraz.at/aschwaig/software.html

[Sch97] B. Schölkopf, K.K. Sung, C.J.C. Burges, F. Girosi, P. Niyogi, T. Poggio and V. Vapnik, "Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers," *IEEE Trans. on Signal Processing*, Vol. 45, No. 11, pp. 2758-2765, 1997.

[Set2K] R. Setiono and A. Gaweda, "Neural network pruning for function approximation", *Procs. of IJCNN-2000*, Vol. 6, pp. 443-448.

[Set2K1] R. Setiono, "Feedforward Neural Network Construction using Cross Validation," *Neural Computation*, Vol. 13, No. 12, pp. 2865-2877, 2001.

[Set95] R. Setiono and L.C.K. Hui, "Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm", *IEEE Trans. on Neural Networks*, Vol. 6, No. 1, pp. 273-277, 1995.

[Set97] R. Setiono, "A Penalty-Function Approach for Pruning Feedforward Neural Networks", *Neural Computation*, 9, pp. 185-204, 1997.

[Smi68] F.W. Smith, "Pattern Classifier Design by Linear Programming", *IEEE Trans. on Computers*, Vol. C-17, No. 4, pp. 367-372, 1968.

[Smo98a] A.J. Smola and B. Schölkopf, "On a Kernel Based Method for Pattern Recognition, Regression, Approximation and Operator Inversion," *Algorithmica*, Vol. 22, pp. 211-231, 1998.

[Smo98b] A.J. Smola, B. Schölkopf and K.R. Müller, "The connection between regularization operators and support vector kernels", *Neural Networks*, 11(5), pp. 637-649, 1998.

[Sri93] U. Sridhar and A. Basu, "Data Partitioning Schemes for the Parallel Implementation of the Revised Simplex Algorithm for LP problems", *Procs. of Seventh International Parallel Processing Symposium*, pp. 379-383, 1993.

[Suy2K1] J.A.K. Suykens, J. Vandewalle and B. De Moor, "Optimal control by least squares support vector machines", *Neural Networks*, 14, pp. 23-35, 2001.

[Suy99] J.A.K. Suykens and J. Vandewalle, "Least Squares Support Vector Machines Classifiers", *Neural Processing Letters*, 9 (3), pp. 293-300, 1999.

[Tho96] H.H. Thodberg, "A Review of Bayesian Neural Networks with an Application to Near Infrared Spectroscopy", *IEEE Trans. on Neural Networks*, Vol. 7, No. 1, pp. 56-72, 1996.

[Tit96] P. Tiitinen, "The Next Generation Motor Control Method, DTC, direct torque control", *Procs. of International Conference on Power Electronics, Drives and Energy Systems (PEDES) for Industrial Growth*, Vol. 1, Jan 1996, pp. 37-43.

[Tre99] N.K. Treadgold and T.D. Gedeon, "Exploring Constructive Cascade Networks", *IEEE Trans. on Neural Networks*, Vol. 10, No.6, pp. 1335-1350, 1999.

[Vap2K] V. Vapnik, "*The Nature of Statistical Learning Theory* (Second Ed.)", Springer-Verlag New York Inc., 2000.

[Vap98] V. Vapnik, "*Statistical Learning Theory*," Wiley, 1998.

[Vas98] P. Vas, "*Sensorless Vector and Direct Torque Control*", Oxford University Press, 1998.

[Wai] Http://www.cs.waikato.ac.nz/~ml/weka/index.html

[Wer2K1] H. Wersing, W.J. Beyn, H. Ritter, "Dynamical Stability conditions for Recurrent Neural Networks with Unsaturating Piecewise Linear Transfer Functions", *Neural Computation*, 13, pp. 1811-1825, 2001.

[Wer90] P.J. Werbos, "*A menu of designs for reinforcement learning over time.* In W. Miller, R. Sutton & P. Werbos (Eds.)", Neural Networks for Control. Cambridge, MA: MIT Press.

[Wis71] D.A. Wismer, "*Optimization Methods for Large-Scale systems...with applications*," McGraw Hill Book Company, 1971.

[You98] S. Young and T. Downs, "CARVE- A Constructive Algorithm for Real-Valued Examples", *IEEE Trans. on Neural Networks*, Vol. 9, No. 6, pp. 1180-1190, 1998.

[Zur97] J.M. Zurada, "*Introduction to Artificial Neural Systems*", Jaico Publishing House, Mumbai, India, 1997.

About the Author

Born in 1971, Alok Kanti Deb obtained First Class in B.E. degree in 1994 from the Department of Electrical Engineering, Bengal Engineering College, Shibpur, Howrah, West Bengal, India affiliated to Calcutta University. Subsequently, he joined as Engineer with The Calcutta Electric Supply Corporation Limited and worked till 1997, having experienced in the erection, commissioning, operation, maintenance, and condition monitoring of turbine-generator sets and other power plant auxiliaries. He then pursued the full-time M. Tech programme in Control Engineering and Instrumentation at the Department of Electrical Engineering, IIT Delhi and continued for the Ph.D programme in the same department under the supervision of Prof. Jayadeva and Prof. M Gopal. His research interests include Control Engineering and Computational Intelligence. He is the co-author of the following publications:

1) Deb A.K., Jayadeva, Chandra S., and Gopal M., "Modified Growth Network for Pattern Classification", Operations Research And Its Applications: Recent Trends (APORS 2003). In M.R. Rao and M.C. Puri (Eds.), Allied Publishers Pvt Limited, Vol I, pp. 270-277.

2) Jayadeva, A.K. Deb and S. Chandra, "Algorithm for building a neural network for function approximation", IEE Proc.- Circuits, Devices, Syst. Vol 149, No. 5/6, pp. 301-307, Oct/Dec 2002.

3) Jayadeva, Alok Kanti Deb, and Suresh Chandra, "Binary Classification by SVM Based Tree Type Neural Networks", Proceedings of International Joint Conference of Neural Networks (IJCNN)-2002, Vol. 3, 12-17 May, 2002, Honolulu, Hawaii, USA, pp. 2773-2778.

4) Jayadeva and Alok Kanti Deb, "Learning Sequences using Tree Type neural networks", Proceedings of the Sixth International Conference of Cognitive and Neural Systems, Dept of Cognitive and Neural Systems, Boston University, USA, May 30-June 1, 2002.

He is the recipient of the following award:

1) Student Travel Grant Award at the World Conference on Computational Intelligence (WCCI), May 12-17, 2002, Honolulu, Hawaii, USA.