OPTIMAL USE OF REGULARIZATION AND CROSS-VALIDATION IN NEURAL NETWORK MODELING

By

DINGDING CHEN

Bachelor of Science
Beijing Agricultural Engineering University
Beijing, China
1982

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1996

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 2000

OPTIMAL USE OF REGULARIZATION AND CROSS-VALIDATION

IN NEURAL NETWORK MODELING

Thesis Approved:

[illegible]

Thesis Adviser

[illegible]

[illegible]

[illegible]

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation and gratitude to Dr. Martin Hagan for serving as my major advisor. His support, guidance, encouragement, and friendship contributed immeasurably to my research effort. I would also like to extend my appreciation to Dr. George Scheets, Dr. Carl Latino and Dr. Glenn Kranzler for serving on my graduate committee. Their suggestions and thorough editing and proof reading of my dissertation were very helpful.

Additionally, I would like to give a special thanks to Halliburton Energy Services in Houston for supporting this research and providing experimental data.

I would especially like to thank my wife Kexin and son Jiayue. Their love, understanding and encouragement during this endeavor make my success possible.

Finally, I dedicate this dissertation to my parents. They undertook the awesome responsibility and sacrifice to support me to get my first college degree, and encouraged me to pursue every academic or professional goal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

NOMENCLATURE

Symbols

$\boldsymbol{a}^m$	The net output vector in layer m
a_j^m	The output of the neuron j in layer m
$\boldsymbol{b}^m$	The bias vector in layer m
b_j^m	The bias of the neuron j in layer m
D_n	The data set including n input-target pairs $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), ...(\boldsymbol{x}_n, \boldsymbol{y}_n)\}$
$\boldsymbol{f}^m$	A set of transfer functions in layer m
E^D	General error function for the given data set
E_t^D	Error function of the training data set
E_v^D	Error function of the validation data set
$\boldsymbol{e}(\boldsymbol{w})$	General error vector for the given data set
$\boldsymbol{e}_t(\boldsymbol{w})$	Error vector of the training data set
$\boldsymbol{e}_v(\boldsymbol{w})$	Error vector of the validation data set
$F(\boldsymbol{w})$	General performance function
$F_t(\boldsymbol{w})$	Training performance function
$F_v(\boldsymbol{w})$	Validation performance function
$\boldsymbol{G}(\boldsymbol{w})$	Fisher information matrix

H_o	Entropy of the teacher network
$H_\alpha(w_{k+1}(\alpha_k))$	Incremental Hessian of the validation error with respect to α
$\boldsymbol{H}(\boldsymbol{w})$	Hessian Matrix of the training data set
$\boldsymbol{H}_v(\boldsymbol{w})$	Hessian matrix of the validation data set
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{J}(\boldsymbol{w})$	Jacobian matrix of the training set
$\boldsymbol{J}_v(\boldsymbol{w})$	Jacobian matrix of the validation set
n	The number of total samples
n_t	The number of training samples
n_v	The number of validation samples
N	The number of network parameters
$N(\boldsymbol{w})$	Student network
$N(\boldsymbol{w}_o)$	Teacher network
$P(\boldsymbol{y}_i \| \boldsymbol{x}_i, \boldsymbol{w})$	Conditional probability of $\boldsymbol{y}_i$ given the input $\boldsymbol{x}_i$ and weight vector $\boldsymbol{w}$
$P(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{w})$	Joint probability of $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ with network parameters $\boldsymbol{w}$
$P(D \| \boldsymbol{w}, \beta)$	Likelihood function in Bayesian regularization
$P(D \| \alpha, \beta)$	The evidence for α and β in Bayesian regularization
$P(\boldsymbol{w} \| D, \alpha, \beta)$	Posterior density function of the weights in Bayesian regularization
$P(\boldsymbol{w} \| \alpha)$	Prior density function of the weights in Bayesian regularization
$P(\alpha, \beta \| D)$	Posterior density function of α and β in Bayesian regularization
$P(\boldsymbol{x}_i)$	Unknown input probability
$P(\alpha, \beta)$	Prior density function of α and β in Bayesian regularization
Q_δ	The set of possible solutions of $\boldsymbol{w}$ with noise corrupted measurements

r	The ratio of the training samples to the total samples
r'	The ratio of the validation samples to the total samples
$R_{\text{train}}(\boldsymbol{w})$	The training entropy of the student network
$R_{\text{gen}}(\boldsymbol{w})$	The generalized entropy of the student network
$\langle R_{\text{train}}(\hat{\boldsymbol{w}})\rangle$	Asymptotic expectation of the entropic training error
$\langle R_{\text{gen}}(\hat{\boldsymbol{w}})\rangle$	Asymptotic expectation of the entropic generalization error
$\langle R_{\text{gen}}(\boldsymbol{w}^*, r_{\text{opt}})\rangle$	Asymptotic expectation of the entropic generalization error using cross-validated early stopping with the optimal data splitting ratio
S^m	The number of neurons in layer m
$\boldsymbol{u}$	Output vector of linear systems
$\boldsymbol{u}_\delta$	Noise corrupted output vector of linear systems
U	Real output space of linear systems
$\boldsymbol{x}$	General input vector
$\boldsymbol{x}^m$	The net input vector in layer m
$y(\boldsymbol{x})$	Target function which produces the mapping of the input vector $\boldsymbol{x}$
$Y(\boldsymbol{x}, \boldsymbol{w})$	The approximating function of $y(\boldsymbol{x})$ with parameter vector $\boldsymbol{w}$
$\boldsymbol{w}$	Weight (parameter) vector for linear or nonlinear systems
$\boldsymbol{w}_o$	The true weight vector of the teacher network
$\boldsymbol{w}_k$	The weight vector resulting from the kth iteration
$\boldsymbol{w}_{MP}$	The most probable weight vector in Bayesian regularization
$\boldsymbol{w}_t$	The true weight vector of the linear system
$\boldsymbol{w}_\delta$	Inverse solution of the linear system with noise corrupted outputs
$\boldsymbol{w}^m$	The weight vector in layer m

w^*	The weight vector corresponding to the early stopping point
w_{ij}^{m}	The weight connected from neuron i in layer $m-1$ to neuron j in layer m
$\hat{w}$	The weight vector corresponding to the minimum of the unregularized performance function
$\widehat{w}$	The weight vector corresponding to the minimum of the regularized performance function
$\tilde{w}$	The weight vector corresponding to the minimum of the validation error function
W	Real weight space for linear or nonlinear systems
α	Regularization parameter
β	Performance function parameter in Bayesian regularization
γ	The effective number of parameters
δ	Small change in output measurements
ε	Small change in parameter estimation
η	Learning rate of the gradient descent training algorithm
λ_i	Eigenvalues of the Hessian matrix
μ_w	Tunable coefficient in weight update equation using the Levenberg-Marquardt algorithm
μ_α	Tunable coefficient in α update equation
Σ	The variance-covariance matrix of multivariate w
σ	Standard deviation of the noise in the data
$\Omega(w)$	Stabilizing functional of the regularized performance function
$\nabla F(w)$	Gradient vector of the performance function with respect to w
$\nabla_\alpha F_v(w_{k+1}(\alpha_k))$	Incremental gradient of the validation error with respect to α

Abbreviations

AIC	Akaike information criterion
BR	Bayesian regularization
BRES	Bayesian regularization with retrained early stopping
CDT	Conditional updating
CVG	Convergent updating
DSR	Data splitting ratio
ES	Cross-validated early stopping
FDVR	First-derivative-of-validation-error regularization
flops	Float point operations
GPE	Moody's generalized prediction error
GNBR	Gauss-Newton approximation to Bayesian learning
GPBR	GPE-validated Bayesian regularization
INC	Incremental updating
logsig	log-sigmoid transfer function
MFNN	Multilayer feedforward neural network
MSE	Mean-squared-error
MSLE	Mean-squared-log-error
NIC	Network information criterion
OBD	Optimal brain damage
OBS	Optimal brain surgeon
PPR	Pattern / parameter ratio
purelin	Pure linear transfer function
RCVE	Relative change of validation error
RTES	Retrained early stopping
SDVR	Second-derivative-of-validation-error regularization
SSM	Statistical stepwise method
SSW	Sum-squared-weights

std	Standard deviation
tansig	Hyperbolic tangent-sigmoid transfer function
VBBR	Validation-set-based Bayesian regularization

CHAPTER 1

INTRODUCTION

Objectives

Regularization and cross-validation are two important techniques used for improving performance of neural networks. Our goal in this study is to investigate how to make optimal use of regularization and cross-validation in developing neural network models.

In this chapter, we will define the dissertation research, justify its significance, describe our contributions and outline the flow of this document. This chapter gives an overview of the dissertation study. The relevant materials will be explained in detail in the rest of the chapters.

Overview

Many problems in the areas of signal processing, system identification, time series analysis and adaptive control can fall into the category of nonlinear regression or function approximation. Some of these problems can be solved with linear models, the other problems may display very complex behavior which can be captured only by using nonlinear techniques. Neural networks provide a very useful and convenient tool for a broad range of applications. In this proposal, we concentrate our study on nonlinear regression or function approximation problems using neural networks. The advantages of using nonlinear neural network modeling are:

- Neural network architectures are very flexible, which enables them to accurately model strong nonlinear relationships.
- The adjustment of network parameters is systematic and efficient for many standard optimization algorithms.

As with other nonlinear methods, the neural network model is developed from a set of input/output examples through a training or learning process. The main concern in assessing a neural network model is not how accurate the mapping is between the given training inputs and outputs, but the potential of the resulting network in prediction with new inputs which are not used for the training. Usually, more exact representation of the training data can be achieved by using more network parameters. However, a model with too many coefficients may give poor prediction for new inputs, since it may fit the noise in the training data, or it may have too much flexibility. This problem is called overfitting. Training a network to an adequate accuracy with less risk of overfitting is a challenging task. It is also an open area in neural network research.

A number of different methods, such as network pruning, regularization, cross-validation, network committee, model selection criteria and so on (which will be introduced in Chapter 2), have been developed to avoid overfitting. Among these approaches, we find that regularization and cross-validation deserve extra attention. The standard regularization method uses a stabilized objective function to train networks. The effective complexity or smoothness of the model is determined by the value of a coefficient called the regularization parameter. This parameter can be automatically updated during the training when the Bayesian technique [MacK92] [FoHa97] is applied. The cross-validation method uses a withheld validation data set, which is disjoint from the training set, to assess the model after training, or to determine when to stop training. A typical application of this technique is called early stopping, which terminates the network training as soon as the validation error increases [Bish96] [AmMu97].

In this research, we will extend regularization and cross-validation techniques by developing some new methods. The new methods will be compared with current methods on a variety of applications to shed light on the optimal use of these techniques.

First, we will investigate the possibility of retraining a network using the combined data set after early stopping. It might improve the parameter estimation if limited retraining is performed around the early stopping point.

Secondly, we will control the effective complexity of neural networks using the concept of validation-set-based regularization. Since the validation set is disjoint with the training set, it is reasonable to use the validation set to estimate the prediction potential of neural networks. The model performance can be improved if the validation set is actively used to determine the optimal regularization parameter.

We propose a new procedure called retrained early stopping (RTES) to improve the performance of networks trained with conventional early stopping. We also propose two approaches for validation-set-based regularization. In the first approach, the regularization parameter is automatically adapted by minimizing the validation error. A second-derivative-of-validation-error regularization (SDVR) algorithm is developed for this purpose. In the second approach, we propose a validation-set-based Bayesian regularization (VBBR) method, in which the regularization parameters are adapted to maximize the Bayesian evidence on the validation set. We present a thorough investigation of the first approach in Chapter 5. The second approach, along with the other validation-incorporated Bayesian methods, will be fully discussed in Chapter 7.

Main Contributions

The main contributions of this research are listed below according to the order of their appearance in the dissertation:

- Development of a new RTES procedure. This procedure performs a controlled full-data retraining to obtain better parameter estimation, after reasonable model accuracy and complexity are determined using partial data from the conventional cross-validated early stopping.
- Derivation of a new SDVR algorithm which uses the second order information to set the optimal learning rate for the regularization parameter to minimize the validation

error. The regularization parameter is updated in each training epoch, rather than after the weights have converged. This method is called incremental updating.

- Implementation of several variations of the SDVR algorithm. These variations can be put into the same SDVR framework, which allows the regularization parameter to be calculated with same update equation, but at a variable weight update interval. With these variations, the SDVR algorithm becomes more efficient on a variety of different problems.
- Comparison of the SDVR algorithm and RTES procedure with Bayesian regularization and cross-validated early stopping through extensive simulations. The relative advantages and limitations of each algorithm are investigated to enable optimal algorithm selection for different applications.
- Development of the validation-incorporated Bayesian learning. Three variations of Bayesian methods are proposed and implemented. The VBBR (validation-set-based Bayesian regularization) method updates the Bayesian regularization parameters by maximizing the validation evidence. The BRES (Bayesian regularization with early stopping) approach uses the validation set to determine when to stop the training. The GPBR (GPE-validated Bayesian regularization) implementation employs Moody's generalized prediction error (GPE) as a stop criterion instead of measuring the validation error. These variations improve the standard Bayesian method for some applications.
- Demonstration of the performance of the new methods on real-world problems. These examples include indirect measurement of engine emission from speed and fueling,

prediction of a chaotic intensity pulsation time series of an NH_3 Laser, determination of formation parameters from well logging tool responses, and indirect measurement of cholesterol.

Outline

This dissertation is composed of nine chapters.

Chapter 2 briefly reviews the basic concepts of nonlinear modeling using neural networks. This includes discussions of the architecture and capability of multilayer feed-forward neural networks, practical training methods, the concept of generalization, and techniques to improve generalization.

In Chapter 3 the regularization technique is investigated in depth. We explain how standard regularization can be used to solve an ill-posed problem, and to reduce variance in model performance. Two variations of regularization, the discrepancy method and the Bayesian method, are discussed and demonstrated.

Chapter 4 describes a method which uses a validation data set to determine when to stop training. The results from Amari's asymptotic statistical theory of cross-validation [AmMu97] are introduced in this chapter, along with a discussion of the use of cross-validated early stopping in non-asymptotic cases. Later in this chapter, the RTES procedure is proposed and demonstrated.

The concept of validation-set-based regularization is introduced at the beginning of Chapter 5. Then the basic SDVR algorithm is derived and compared with the gradient descent method. The other two variations of the SDVR algorithm are also implemented. This is followed by numerical experiments to investigate the optimal use of the SDVR framework.

Chapter 6 includes extensive simulations of the SDVR algorithm, Bayesian regularization, cross-validated early stopping, and retrained early stopping. The relative advantages and limitations of each method are discussed.

In Chapter 7, the GPBR, VBBR and BRES methods, which use validation-incorporated Bayesian learning, are proposed. The objective of these methods is to improve generalization performance and training efficiency of the standard Bayesian regularization algorithm. Simulation results obtained with these methods are compared with the standard method.

In Chapter 8, we apply the new developed methods to real-world problems, and compare them with the algorithms of Bayesian regularization and cross-validated early stopping. Five representative problems with realistic complexity are selected to test the algorithms. The relevant data analysis and preprocessing are also addressed in these case studies.

Chapter 9 contains a summary of the dissertation research. This is followed by a recommendations for future work.

CHAPTER 2

NONLINEAR MODELING WITH NEURAL NETWORKS

Objectives

This chapter reviews important background material for nonlinear modeling with neural networks. This review will lay the foundation for subjects discussed in the following chapters.

Introduction

In nonlinear regression, the major task can be regarded as approximating an underlying function based on a set of examples. The generated model retrieves the appropriate output when presented with an example input and generalizes when presented with new inputs. Many classical approaches for this problem are based on approximation theory and system identification techniques.

Approximation theory deals with the problem of approximating or interpolating a continuous, multivariate function $y(\boldsymbol{x})$ by an approximating function $Y(\boldsymbol{x}, \boldsymbol{w})$, where $\boldsymbol{x}$ is an input vector and $\boldsymbol{w}$ is a parameter vector. For a choice of a specific Y, the problem is then to find the set of parameters $\boldsymbol{w}$ that provides the best possible approximation of y on the set of examples. This is the learning step. Needless to say, it is very important to choose an approximating function Y that can accurately represent y. There would be little point in trying to learn, if the chosen approximation function $Y(\boldsymbol{x}, \boldsymbol{w})$ could only give a very poor representation of $y(\boldsymbol{x})$, even with optimal parameter values. Therefore, the first concern in nonlinear regression is to determine which approximation to use, i.e., which class functions of $y(\boldsymbol{x})$ can be effectively approximated by which approximating functions $Y(\boldsymbol{x}, \boldsymbol{w})$. This is a representation problem. The second concern is to determine which algorithm to use for finding the optimal values of the parameters $\boldsymbol{w}$ for a given choice of Y.

In this chapter, we will briefly review the relevant background of using neural networks as universal function approximators to solve nonlinear regression problems. The main topics include general concepts about network structure, training method and generalization.

Multilayer Feedforward Neural Networks

Although the biological inspiration for neural networks is important, we only discuss extremely simplified models called artificial neural networks, which are basically engineering functions rather than physiological forms. Artificial neural networks can be regarded as a graphic notation for a large class of algorithms by arranging the basic computing elements into various configurations. In this study, we will use the most popular configuration, the multilayer feedforward neural network (MFNN), to solve nonlinear modeling problems.

Architecture

A simplified graphical notation for the MFNN is illustrated which is in Figure 1. This architecture consists of a number of processing elements, called neurons (s^1, s^2) which are connected in a network by weights (w^1, w^2). The input vector $\boldsymbol{x}$ feeds through the network to the output $\boldsymbol{a}^2$ in one direction and no feedback connections are present.

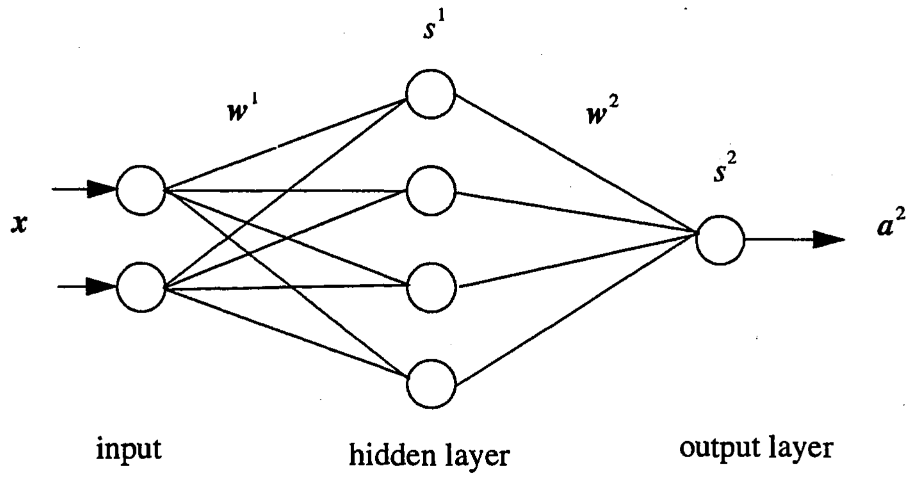


Figure 1 Two-layer Network, Simplified Connection

The neurons are organized in layers such that the outputs of the neurons in one layer act as the inputs for the neurons in the next layer. The layer which produces the output is denoted as the output layer. Any layer which is not an output layer is called a hidden layer. In this terminology the inputs are not considered as an independent layer. For instance, we will use 2-4-1 to denote a two-layer network with 2 inputs, 4 neurons in the hidden layer and 1 neuron in the output layer. Similarly, 5-10-10-1 is a denotation for a three-layer network using 5 inputs, 10 neurons in the first hidden layer, 10 neurons in the second hidden layer with a single output.

A single neuron extracted from the hidden layer is depicted in Figure 2, and an abbreviated, but more standard notation of a two-layer network is given in Figure 3, where f^m is a transfer function in layer m, $\boldsymbol{w}^m$ and $\boldsymbol{b}^m$ are the weight and bias vector, and $\boldsymbol{a}^m$ is the layer output.

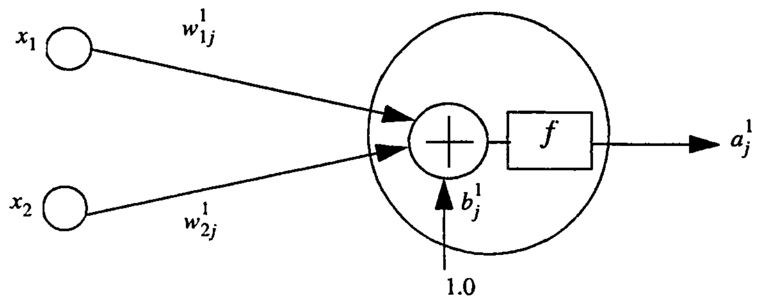


Figure 2 Single Neuron Structure

To understand better the MFNN scheme in nonlinear modeling, first let's see what is the general approximation form of $Y(\boldsymbol{x}, \boldsymbol{w})$ that the MFNN represents. If we put $\boldsymbol{w}^m$ and

b^m together by redefining $w^m = \begin{bmatrix} w^m & b^m \end{bmatrix}$ and let $a^{m-1} = \begin{bmatrix} a^{m-1} & 1 \end{bmatrix}^T$, then $Y(x, w)$ can be expressed as

$$Y(x, w) = a^m = f^m(w^m f^{m-1}(w^{m-1} f^{m-2}(\ldots f^1(w^1 x)\ldots))). \tag{1}$$

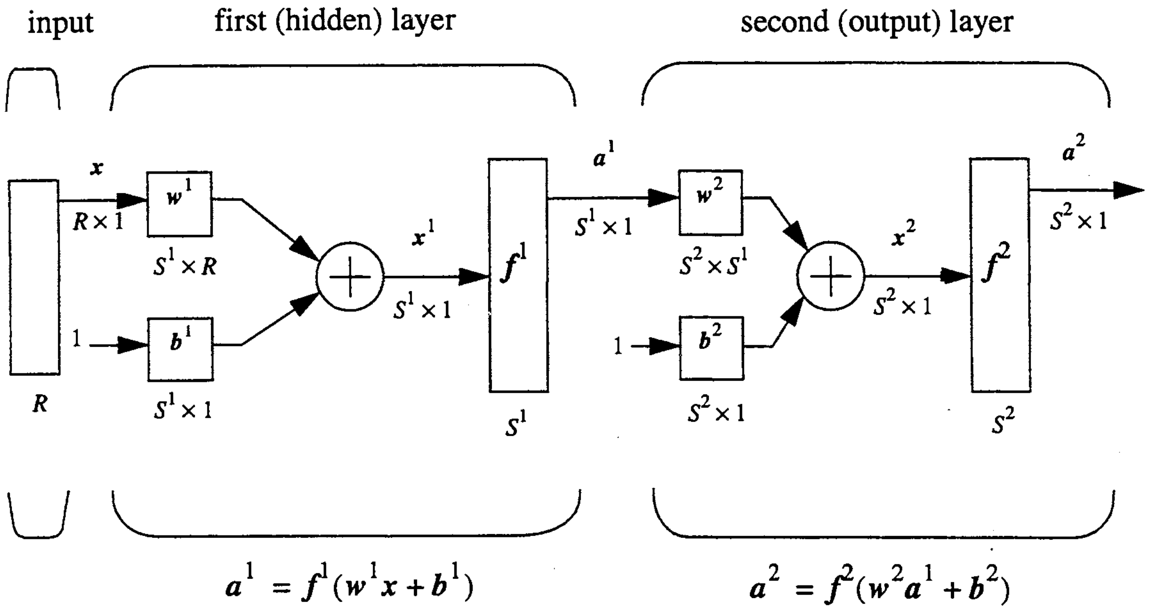


Figure 3 Two-layer Network, Abbreviated Notation

Note that Eq. (1) is a nested transformation with transfer function $f^m, \ldots f^1$. Since nonlinear transfer functions are usually used in the hidden layers of the MFNN structure, the MFNN approximation function is a highly nonlinear function of both input x and parameter w. This is different from the classical linear approximation

$$Y(x, w) = w \cdot x, \tag{2}$$

or the classical approximation scheme which is linear in a suitable basis function $\phi(.)$ of the original inputs

$$Y(x, w) = w \cdot \phi(x). \tag{3}$$

Transfer Function

Now let's look into the internal neuron structure of the MFNN scheme. The neuron is characterized by its transfer function (it is also called an activation function). For regression problems, the typical transfer function used on hidden layers is the log-sigmoid (logsig) function or the hyperbolic tangent sigmoid (tansig) function. The transfer function used on the output layer is usually a linear (purelin) function. To reduce the computational complexity, we assume that the neurons on the same layer are activated by the same transfer function. The logsig function is defined by

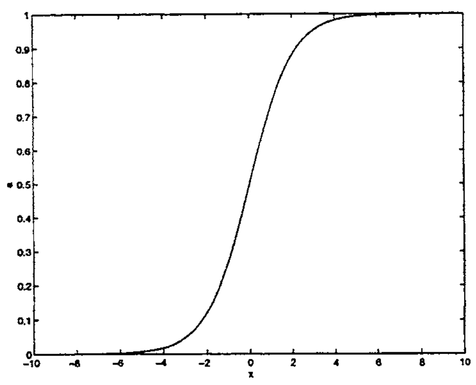
$$a = 1/(1 + e^{-x}), \tag{4}$$

where the output a ranges from 0 to 1 as the input x changes from the minus infinity to infinity. The tansig function has the form of

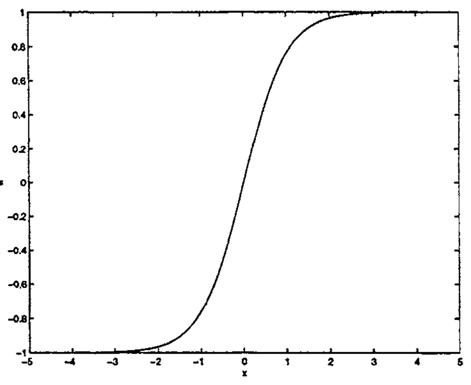
$$a = (e^{x} - e^{-x})/(e^{x} + e^{-x}). \tag{5}$$

Its shape looks like the logsig function, but shows odd symmetry and ranges from -1 to 1. These two functions are displayed in Figure 4.

Note that both the logsig and the tansig functions are squashing functions, since they compress an infinite input range into a finite output range. Also, they have a roughly linear response within a narrow input range, but produce more of a binary type output out of that range. In comparison, the purelin function will produce the pure linear output over the whole input range. Using networks with linear output neurons does not restrict the class of functions which such networks can approximate. The use of sigmoid neurons on the output layer would limit the range of possible outputs to the range attainable by the sigmoid, and in some cases this would be undesirable.



a. logsig function



b. tansig function

Figure 4 Typical Hidden Layer Transfer Functions

Another reason to choose the transfer functions described above is for their continuity and differentiability. Smooth input-output transformations can be achieved with these functions, and the derivatives of these functions have simple forms which make optimizing parameters convenient. Since each neuron input x is determined by its inputs and the connecting parameters (weights and bias), adjusting these parameters will affect the neuron output. The final network output will be determined by the composite connection and weighting of the specified MFNN structure.

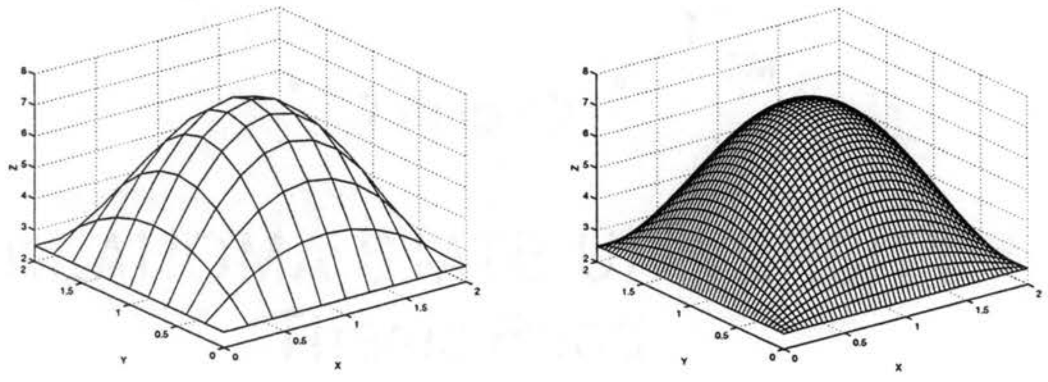
Capability

Many mathematical results prove that the MFNN is a general function approximator [Cybe89], [Funa89], [HoSt89], [Horn91]. If enough neurons are used in the hidden layer, even a simple two-layer (one hidden layer) network will be sufficient to represent any arbitrary continuous function. In this study, we will restrict the architecture by using two-layer and three-layer networks. Although three-layer networks may require more computations than two-layer networks, they may require fewer network parameters when the input dimensionality is large [ChHa98b].

As an example to show the capacity of the MFNN in nonlinear modeling, let's consider a parabolic function

$$Z = 2.5 + 5.0X(2-X)Y(2-Y), \tag{6}$$

where both X and Y range from 0 to 2. Assume that 121 input-output pairs, which are indicated by the cross-points on the grid surface of Figure 5 (a), can be used to learn the function. We choose a 2-8-1 network with the tansig transfer function in the hidden layer and the purelin function in the output layer. The reconstructed surface is shown in Figure 5 (b), which is a very accurate approximation to the true function.



a. input-output pairs b. surface reconstruction

Figure 5 Function Approximation Using MFNN

Training Methods

As we addressed early in this chapter, an important concern in nonlinear modeling is the choice of the appropriate algorithm to find the optimal parameters. Using the MFNN as a general function approximator, the network parameters are optimized through a training process by minimizing the performance function measured on a set of examples called training data. The choice of the performance function is problem dependent [Bish95], [ChHa98b].

In this study, we assume that the approximation error is additive, thus

$$y(i) = Y(\boldsymbol{x}(i), \boldsymbol{w}) + e(\boldsymbol{x}(i), \boldsymbol{w}) \ , \tag{7}$$

where $\{\boldsymbol{x}(i), y(i)\}$ are an input-output pair of training examples, $Y(\boldsymbol{x}(i), \boldsymbol{w})$ is the network output, and $e(\boldsymbol{x}(i), \boldsymbol{w})$ is the approximation error. The training performance function is the sum-of-squared error over all training data, i.e.,

$$F(\boldsymbol{w}) = E^D = \sum_{i=1}^{n} (y(i) - Y(\boldsymbol{x}(i), \boldsymbol{w}))^2 = \boldsymbol{e}^T(\boldsymbol{w})\boldsymbol{e}(\boldsymbol{w}) \ , \tag{8}$$

where $\boldsymbol{e}(\boldsymbol{w})$ is an $n \times 1$ error vector and n is the number of training examples. For multilayer networks, the performance function will typically be a highly non-linear function of the weights, and there may exist many local minima. As a consequence of the non-linearity of the performance function, it is not in general possible to find closed-form solutions for the minima. Instead, we consider algorithms which involve a search through weight space consisting of a succession of steps of the form

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \Delta\boldsymbol{w} \tag{9}$$

The problem of minimizing continuous, differentiable functions of many variables is one which has been widely studied ([Pola71], [Gill81], [DeSc83], [Luen84], [Flet87]), and many of the conventional approaches to this problem are directly applicable to the training of neural networks, since the derivative of performance function with respect to the network parameters can be obtained in a computationally efficient way using back-propagation [RuHi86][RuMc86]. In this section, we will briefly review two of the most important training algorithms. One is the gradient descent, which updates the network parameters with a small step in a descent direction. The other is the powerful Levenberg-Marquardt algorithm, which is applicable specifically to a sum-of-squared error function.

Gradient Descent

Assume that the network parameters are organized in an $N \times 1$ vector and iteratively updated during the training process with a small step along a search direction. Then we can rewrite Eq. (9) as

$$w_{k+1} = w_k + \eta_k p_k, \tag{10}$$

where k is an iteration index number, η_k is a small positive scalar called the learning rate or step size, and p_k is a search direction. For a given η_k, the goal of getting the new parameter vector w_{k+1} is to determine the search direction p_k. In the optimization algorithm, we would like to have the error function decrease at each iteration. In other words,

$$F(w_{k+1}) < F(w_k). \tag{11}$$

Using the first-order Taylor series expansion, we have

$$F(w_{k+1}) \approx F(w_k) + (\nabla F(w_k))^T (w_{k+1} - w_k). \tag{12}$$

To make $F(w_{k+1}) < F(w_k)$, the inner product $(\nabla F(w_k))^T(w_{k+1} - w_k)$ must be negative. Since $w_{k+1} - w_k = \eta_k p_k$ from Eq. (10), the descending error condition becomes

$$(\nabla F(w_k))^T \eta_k p_k < 0 . \tag{13}$$

For a fixed η_k, the inner product $(\nabla F(w_k))^T p_k$ will be most negative if the search direction is equal to the negative gradient, i.e.,

$$p_k = -\nabla F(w_k) . \tag{14}$$

Putting Eq. (14) into Eq. (10), we obtain the weight updating equation of the gradient descent training algorithm:

$$w_{k+1} = w_k - \eta_k \nabla F(w_k) . \tag{15}$$

In Eq. (15), the gradient vector is calculated by differentiating Eq. (8) with respect to w,

$$\nabla F(w_k) = \left. \frac{\partial e^T(w) e(w)}{\partial w} \right|_{w = w_k} = 2 J^T(w_k) e(w_k) , \tag{16}$$

where $J(w)$ is an $n \times N$ Jacobian matrix defined by

$$J(\mathbf{w}) = \frac{\partial e(w)}{\partial w} = \begin{bmatrix} \frac{\partial}{\partial w_1} e_1(w) & \frac{\partial}{\partial w_2} e_1(w) & \cdots & \frac{\partial}{\partial w_N} e_1(w) \\ \frac{\partial}{\partial w_1} e_2(w) & & & \\ & & & \\ \frac{\partial}{\partial w_1} e_n(w) & & \cdots & \frac{\partial}{\partial w_N} e_n(w) \end{bmatrix} . \tag{17}$$

Calculating the Jacobian matrix is very important in training neural networks. Recall that for each training iteration, the error vector is measured on the output layer. However, since the training errors are composite non-linear functions of the network parameters, their derivatives with respect to the network parameters have to be evaluated layer by layer, function by function. For most of the currently used training algorithms, the Jacobian matrix is calculated efficiently using back-propagation, which is an application of the chain rule to the nested MFNN scheme. The back-propagation algorithm is one of the key developments in the history of artificial neural networks. Some in-depth discussions and examples can be found in many text books [HaDe96], [Bish95].

With standard gradient descent, the learning rate η_k in Eq. (15) is held constant throughout the training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. There are other variations of gradient descent which use variable learning rate and momentum to speed up convergence [VoMa88], [HaDe96]. All of these approaches are easy to implement, but suffer from the problem of slow convergence because of the nature of the first-order methods.

Levenberg-Marquardt

The derivation of the gradient descent algorithm was based on the first-order Taylor series expansion. The Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute the second derivative of error with respect to the weights. To explain the Levenberg-Marquardt algorithm, let's start with Newton's method.

Newton's method is based on the local quadratic approximation to the performance function using the second-order Taylor series expansion:

$$F(w_{k+1}) \approx F(w_k) + (\nabla F(w_k))^T (w_{k+1} - w_k) + \frac{1}{2}(w_{k+1} - w_k)^T H(w_k)(w_{k+1} - w_k), \quad (18)$$

where $H(w_k)$ is the Hessian matrix defined by

$$H(w_k) = \left. \frac{\partial^2 F(w)}{\partial w^2} \right|_{w = w_k}. \quad (19)$$

In order to minimize the performance function $F(w_{k+1})$, let's differentiate Eq. (18) with respect to w_{k+1} and set it to zero,

$$\nabla F(w_k) + H(w_k)(w_{k+1} - w_k) = 0. \quad (20)$$

Solving w_{k+1} from Eq. (20), we get

$$w_{k+1} = w_k - (H(w_k))^{-1} \nabla F(w_k), \quad (21)$$

in which $(H(w_k))^{-1}$ is the inverse of the Hessian matrix. The vector $-(H(w_k))^{-1} \nabla F(w_k)$ is known as the Newton direction or the Newton step, which forms the basis of a variety of optimization strategies. The gradient $\nabla F(w)$ is given in Eq. (16) for the sum-of-squared performance function. The Hessian matrix can be expressed as:

$$H(w) = \frac{\partial}{\partial w} \nabla F(w) = 2J^T(w)J(w) + 2S(w), \quad (22)$$

where

$$S(w) = \sum_{i=1}^{n} e_i(w) \nabla^2 e_i(w), \quad (23)$$

with $\nabla^2 e_i(w)$ being an $N \times N$ matrix:

$$\nabla^2 e_i(w) = \begin{bmatrix} \frac{\partial^2}{\partial w_1 \partial w_1} e_i(\mathbf{w}) & \frac{\partial^2}{\partial w_1 \partial w_2} e_i(\mathbf{w}) & \cdots & \frac{\partial^2}{\partial w_1 \partial w_N} e_i(\mathbf{w}) \\ \frac{\partial^2}{\partial w_2 \partial w_1} e_i(\mathbf{w}) & & & \\ & & & \\ \frac{\partial^2}{\partial w_N \partial w_1} e_i(\mathbf{w}) & & \cdots & \frac{\partial^2}{\partial w_N \partial w_N} e_i(\mathbf{w}) \end{bmatrix}. \qquad (24)$$

Newton's method is much faster than gradient descent for local convergence. However, the problem with the full Newton approach is that $S(w)$ is too expensive to obtain. To reduce the computational complexity, we assume that $S(w)$ is close to zero, thus

$$H(w) \approx 2J^T(w)J(w). \qquad (25)$$

Substituting Eq. (25) and Eq. (16) into Eq. (21), we obtain

$$w_{k+1} = w_k - (J^T(w_k)J(w_k))^{-1} J^T(w_k)e(w_k). \qquad (26)$$

The iterative method using Eq. (26) is called the Gauss-Newton method. This method can also be derived using the first-order Taylor series expansion of the error vector,

$$e(w_{k+1}) \approx e(w_k) + J(w_k)(w_{k+1} - w_k), \qquad (27)$$

and then solving w_{k+1} by minimizing $e(w_{k+1})^T(e(w_{k+1}))$.

The advantage of Gauss-Newton over the standard Newton's method is that it does not require calculation of second derivatives. One problem with the Gauss-Newton method is that the step size given by Eq. (26) could turn out to be relatively large. In that case, the linear approximation in Eq. (27) would no long be valid. The other problem is that the matrix $J^T(w_k)J(w_k)$ may not be invertible.

The Levenberg-Marquardt algorithm uses an approximation to the Hessian matrix in the following Newton-like update:

$$w_{k+1} = w_k - [J^T(w_k)J(w_k) + \mu_w I]^{-1} J^T(w_k)e(w_k). \tag{28}$$

When the scalar μ_w is zero, this is just the Gauss-Newton method. When μ_w is large, this becomes gradient descent with a small step size. Many versions of the Levenberg-Marquardt method have been coded using various strategies to choose μ_w. Since the Gauss-Newton method is fast and accurate near an error minimum, we would like the training to shift towards Gauss-Newton method as quickly as possible. For the standard training algorithm, μ_w is decreased after each successful step ($F(w_{k+1}) < F(w_k)$) and is increased only when a tentative step would increase the performance function. In this way, the performance function will always be reduced at each iteration of the algorithm.

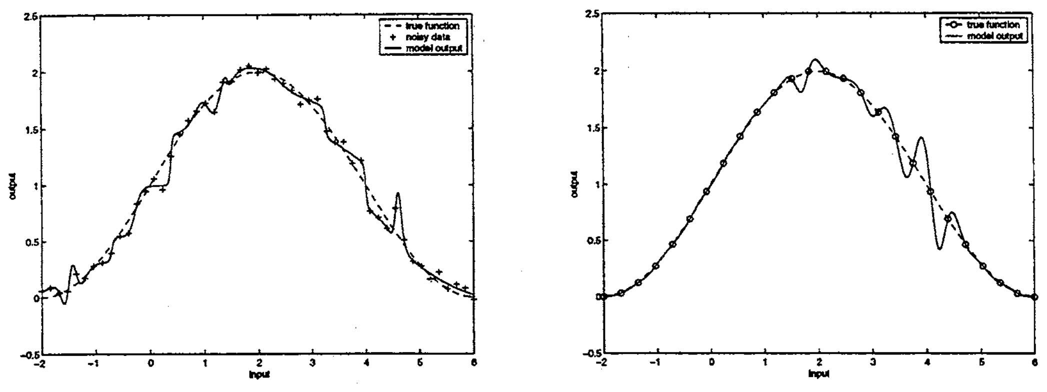
The Levenberg-Marquardt algorithm is very powerful in practice and has been recommended for the general solution of non-linear least-squares problems [DeSc83]. Compared with some other currently used algorithms for training neural networks, the Levenberg-Marquardt algorithm is fast, but requires more storage [HaMe94]. In this study, we will use the Levenberg-Marquardt algorithm unless otherwise specified.

Generalization

It is not difficult to achieve high accuracy in approximating the training set with the MFNN and an appropriate training algorithm. However, the goal of network training is not to learn an exact representation of the training data, but rather to build a statistical model of the process which generates the data. The model generalizes well if its performance with new data is as good as its performance on the training data. This concept of generalization is common to all of nonlinear modeling.

In practice, a model with good generalization performance is hard to achieve, because of the problem of overfitting. Overfitting is typically caused by overtraining on noisy examples, which makes the model very sensitive to the particular data set. Assume that we have additive noise in the training targets, which are the desired network outputs. Performing too many iterations of training to minimize the sum-of-squared error will cause the network to learn the noise in the training data, and the model will perform poorly when making predictions for new inputs. In other words, an overfitted model will exhibit large variance. Overfitting can also be caused by using a complicated model structure with many redundant parameters. When the number of parameters is larger than the number of available training patterns, using the standard least squares techniques will lead to ill-determined final parameters. These two types of problems are illustrated in Figure 6. The variance displayed in Figure 6 (a) is caused by overtraining noise corrupted data using a 1-15-1 network. In Figure 6 (b), even the training data is noise free, the generalization error is still large.

In addition to overfitting, data deficiency (extrapolation) is another problem in nonlinear modeling which might lead to poor generalization. When overfitting is discussed, we assume that the training set is representative of the population under investigation. If the training set is not complete (does not cover the entire input space), then the prediction on new data with unlearned features cannot be guaranteed. Data deficiency will make the training of neural network models, especially the models with multiple inputs, less accurate. However, since procedures for collecting data are beyond the scope of this study, we will only discuss the overfitting problem and will assume that the training patterns cover the input space.



a. noise-oriented overfitting (1-15-1) b. structure-oriented overfitting (1-30-1)

Figure 6 Examples of Overfitting

Many techniques have been developed to improve generalization performance in neural network modeling. In the rest of this chapter, we will give a brief summary of some important techniques. Among them, regularization and early stopping are closely related to our new proposed algorithm and will be investigated in separate chapters.

Network Pruning

Network pruning reduces model complexity by eliminating insignificant parameters which contribute little to training performance but may increase the risk of overfitting with new inputs. With the use of this method, a relatively large network is first trained to convergence. Then each parameter is examined according to the specified criterion to determine its saliency. The low-salience parameters will usually be set to zero and frozen there during the retraining of the network. This process is repeated until no more parameters can be eliminated.

The simplest pruning is based on the magnitude of the parameter [HeKr91]. This naive approach often eliminates the wrong weights for nonlinear system, because small weights can be necessary for low error. The other method called Optimal Brain Damage (OBD) uses the criterion of minimal increase in training error for weight pruning [CuDe90]. In this approach, the second derivative information of the training performance function is critical to decision making. For computational simplicity, OBD assumes that the Hessian matrix is diagonal. Since Hessian matrices are strongly non-diagonal for many problems, Optimal Brain Surgeon (OBS) [HaSt92] suggests calculating the full Hessian to eliminate the unnecessary weights. For the same error on the training set, OBS permits the pruning of more weights than the magnitude based method and OBD, and yields better generalization on test data.

A different approach to weight elimination is using a statistical stepwise method (SSM) [CoGi95]. This method offers a quantitative measure of weight significance. After the network is trained to convergence, the standard deviation of each weight parameter can be calculated from the training performance function and its second derivative information. The least significant parameter might be removed, based on the predetermined confidence interval of the standard statistic test. The SSM method works well for the nonlinear autoregressive time series process, but usually requires that all available data be used in order to develop a proper model. Also, the final model depends strongly on initial weights.

Regularization

While network pruning reduces variance by eliminating the insignificant weights, regularization smooths the model by limiting parameter magnitudes so that the network cannot learn minor features of the training set.

A simplest regularization can be implemented in network training by adding a regularization term $\alpha\Omega(\boldsymbol{w})$ to the standard (unregularized) performance function E^D. The new performance function, called the regularized performance function, becomes

$$F = E^D + \alpha\Omega(\boldsymbol{w}) \tag{29}$$

where $\Omega(\boldsymbol{w})$ is a weight penalty term and α is a regularization parameter. For the given E^D, sum-of-squared error for example, we can control the degree of regularization, and hence the complexity of the model, by choosing the form of $\Omega(\boldsymbol{w})$ and determining the appropriate regularization parameter α. A common form of regularization, known as weight decay, can be written as

$$\Omega(\boldsymbol{w}) = \sum_i w_i^2 \tag{30}$$

where the sum runs over all parameters in the network. There are other forms of $\Omega(\boldsymbol{w})$ discussed in the neural network literature [WeRu91], [Sjöb95], [Will95].

For many problems in nonlinear regression, determining the regularization parameters α is a balance between the smoothness of the curve and the accuracy of the data fitting. If α is too large, the regularization term will dominate the performance function, and the model might be too simple to reflect the complexity of the true process. If α is too small, the emphasis will be placed on the training error, leading to the overfitting. In practice, multiple regularization parameters can also be used in neural network modeling. Determining the optimal regularization parameter is one of the objectives of this study. We will give a thorough investigation in Chapter 3 and Chapter 5.

Early Stopping

Another well-known method to avoid overfitting is early stopping. This method simply divides all available examples into two disjoint sets, a training set and a validation set. The training set is used for learning and the validation set is used for evaluating the generalization error. For many applications, the training error monotonically decreases with the number of iterations, but the error measured on the validation set decreases in an early period but it increases after a critical period. This is the optimal time to stop, because further training would only lead to overfitting.

In general, early stopping can be applied to any training algorithm and can be used with either noisy data or noiseless data. However, its effectiveness strongly depends on the ratio of the number of training samples to the number of network parameters. If the number of training examples n is much larger than the number of parameters N, then early stopping is not needed [AmMu97]. If n is only slightly larger than N, early stopping is effective [MuFi96]. When n is less than or equal to N, using simple validation set does not work well, and some more complicated approaches should be attempted. We will discuss these more complex methods in Chapter 4.

Training with Noisy Inputs

Another method for improving model generalization is to train with noisy inputs, in which random noise is added to the original inputs before each training iteration takes place [SiDo91]. Heuristically, we might expect that the randomly added noise in each iteration will make it difficult to fit individual data points precisely, since there is no exact replication of training patterns in each presentation. It can be proved that training with noise is closely related to the technique of regularization [Bish95], but the problem with this method is that the noise level is usually difficult to determine, especially when example inputs vary over a wide range. In practice, prior knowledge about the given system is important to appropriate application of this method.

Network Committee

A Network committee is a group of networks which could be trained with a variety of structures and initial conditions, or with different data splittings. To improve generalization performance, we can predict outputs with each member network, and then average results over all member networks to produce the committee outputs. Since these outputs are averaged over many solutions, a reduced variance is expected. This approach is different from the common practice, in which many different candidate networks are trained, but only the best one, based on the validation error for instance, is finally selected. However, the risk of keeping one and discarding other networks is high if the training is performed on sparse and noisy data. It is quite often true that the network which has best performance on the validation set may not be the one with the best performance on new test data.

The advantage of using the committee of networks is that all of the effort involved in training the candidate networks will be used, and significant improvement in prediction on new data can be achieved with limited computation overhead. Since the extra variance can be removed by averaging, we can use the standard unregularized optimization algorithm to train the member networks with less worry about overfitting [Bish95].

One problem associated with the committee of networks is the number of parameters. Since multiple models are generated, a tremendous number of parameters might need to be used. For some real world applications, using a more complicated committee to avoid the problem caused by using a complicated single network may not be the best choice.

Model Selection Criteria

Selecting a model from many candidates using some developed algebraic criteria is very common in conventional statistics with linear systems. Usually, the model selection criterion consists of two terms, the training error term and the complexity term. One criterion used to evaluate autoregressive models is the well known Akaike Information Criterion (AIC) [Akai69],

$$\mathrm{AIC} = \ln\frac{F(\mathbf{w})}{n} + \frac{2N}{n}, \tag{31}$$

or the modified BIC criterion

$$\mathrm{BIC} = \ln\frac{F(\mathbf{w})}{n} + \frac{N\log n}{n}, \tag{32}$$

where $F(\mathbf{w})$ is the sum-of-squared errors performance function, n is the number of data points, and N is the number of parameters. Both criteria contain a penalty term, which gives a cost to any additional parameters. If the model is too simple, it will give a large value for the criterion, because the training error is large. On the other hand, if the model is too complex, the criterion will also have a large value, because the complexity is large. In model selection, the smaller the criterion, the better the model.

This idea has been extended to the neural network training to deal with non-linear models. Two criteria which have been used with neural networks are the Network Information Criterion (NIC) [MuYo94], and the Generalized Prediction Error (GPE) [Mood92]. The GPE value can be calculated from the regularized performance function which has the form

$$\mathrm{GPE} = \frac{2F(w)}{n} + \frac{2\gamma}{n}\sigma^2, \tag{33}$$

in which γ is called the effective number of parameters, and σ^2 is the estimated data variance. Using the Gauss-Newton approximation to the Hessian matrix, the parameter γ is given by

$$\gamma = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i + \alpha}, \tag{34}$$

where the λ_i are the eigenvalues of the Hessian matrix of the unregularized performance function, and α is the regularization parameter in Eq. (29). For each network construction, the GPE can be computed at the end of training. The final model could be the one with smallest GPE value among several candidates, or could be determined by using both cross-validation and GPE.

CHAPTER 3

REGULARIZATION TECHNIQUES FOR LEARNING

Objectives

This chapter presents a review of regularization techniques in neural network modeling. The emphasis is put on nonlinear systems, but the traditional use of regularization in linear systems is also described. The purpose here is to explain why we need to use regularization and how we determine the appropriate regularization for a given problem. The chapter includes both theory and examples.

Introduction

We have mentioned in Chapter 2 that one of the well-known methods to improve generalization in neural network modeling is regularization. In this chapter, we continue our discussion of regularization by investigating the problems that regularization techniques are used to solve, and by presenting the practical training methods that are used in neural network applications.

The basic concept of the regularization method was developed for solving ill-posed problems in linear systems. In that case, regularization is applied to find a reasonable inverse solution of the system parameters. In nonlinear systems, especially when a large number of parameters is used, regularization techniques are very effective in reducing the variance of the model and avoiding overfitting.

The key step in applying the regularization technique is determining the regularization parameter, which is straightforward when the noise level is known, but is quite difficult otherwise. In order to automatically optimize the regularization parameter, we introduce the Bayesian regularization method, which produces a probabilistic model to match the training data in a statistically appropriate manner.

Basic Concepts

Ill-posed problems exist in many linear and nonlinear systems where the standard techniques are inadequate to determine appropriate parameter estimates. As a specific treatment, regularization is used to add smoothness constraints to the problems to assist in finding good approximate solutions.

Regularization for Linear System

Consider a system of linear algebraic equations

$$\boldsymbol{A}\boldsymbol{w} = \boldsymbol{u}, \tag{35}$$

where $\boldsymbol{w}$ is an $N \times 1$ unknown parameter vector, $\boldsymbol{u}$ is an $N \times 1$ known output vector, and $\boldsymbol{A}$ is an $N \times N$ square matrix. The problem of determining the solution $\boldsymbol{w}$ in the real space W from the data $\boldsymbol{u}$ in the real space U is said to be well-posed if the following conditions are satisfied:

1. for every element $\boldsymbol{u} \in U$ there exists a solution $\boldsymbol{w} \in W$;
2. the solution is unique;
3. the problem is stable on the spaces (W, U).

Here we consider that the problem is stable on the spaces (W, U) if, for any positive number ε, there exists a positive number $\delta(\varepsilon)$ such that the small change on the right side of Eq. (35) $\rho_U(\boldsymbol{u}_1, \boldsymbol{u}_2) \le \delta(\varepsilon)$ implies the change $\rho_W(\boldsymbol{w}_1, \boldsymbol{w}_2) \le \varepsilon$ on the left side. Where $\boldsymbol{w}_1 = R(\boldsymbol{u}_1)$ and $\boldsymbol{w}_2 = R(\boldsymbol{u}_2)$ are inverse solutions with $\boldsymbol{u}_1$ $\boldsymbol{u}_2$ in U and $\boldsymbol{w}_1$ $\boldsymbol{w}_2$ in W. The function ρ_U or ρ_W can be defined as the root of the sum-of-squared error for simplicity. Problems that do not satisfy the three conditions above are said to be ill-posed. One example of an ill-posed problem is when the output measurement $\boldsymbol{u}$ contains noise. Another example is the singular $\boldsymbol{A}$ matrix. If the matrix $\boldsymbol{A}$ is singular, the parameter $\boldsymbol{w}$ cannot be solved using the standard matrix inverse routine. From a numerical point of view, ill-conditioned systems also behave like singular ones. In all these situations, some additional information is needed to find a good approximation to $\boldsymbol{w}$ that satisfies an approximate equation $\boldsymbol{A}\boldsymbol{w} \approx \boldsymbol{u}$.

Regularization techniques can be used to obtain meaningful solution estimates for such ill-posed problems. Suppose that measurements of $\boldsymbol{u}_\delta$ differ from their exact values $\boldsymbol{u}$ by no more than δ. It then is natural to seek an approximate solution in the class Q_δ of elements $\boldsymbol{w}_\delta$ such that the error function

$$E^D = (A\boldsymbol{w}_\delta - \boldsymbol{u}_\delta)^T (A\boldsymbol{w}_\delta - \boldsymbol{u}_\delta) \le \delta^2. \tag{36}$$

This Q_δ is the set of possible solutions. However, many candidate vectors may meet this requirement. We cannot take an arbitrary element $\boldsymbol{w}_\delta$ of Q_δ as the approximate solution, because such a solution will not in general be continuous with respect to δ. One principle for selecting the possible solutions is to minimize a stabilizing functional $\Omega(\boldsymbol{w})$, called the regularizer, which possesses the following properties:

1. $\Omega(\boldsymbol{w})$ is continuous, nonnegative and everywhere dense in W;
2. the true solution $\boldsymbol{w}_t$ belongs to the domain of definition of $\Omega(\boldsymbol{w})$;
3. for every positive number d, the set of elements $\boldsymbol{w}$ for which $\Omega(\boldsymbol{w}) \le d$ is a compact subset of W.

One of the simplest forms of $\Omega(\boldsymbol{w})$ is called weight decay and consists of the sum of the squares of the parameters

$$\Omega(\boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{w}. \tag{37}$$

It can be proven that the solution $\boldsymbol{w}_\delta$ determined in this way is continuous on δ, hence it is a reasonable approximation of $\boldsymbol{w}_t$ [TiAr77]. In practice, the numerical solution using this approach is often difficult. Instead of minimizing $\Omega(\boldsymbol{w})$ under the constraints $E^D \le \delta^2$, we can minimize the function

$$F = E^D + \alpha\Omega(w), \tag{38}$$

where α is a nonnegative coefficient called the regularization parameter which can be estimated from extreme condition $E^D = \delta^2$. The regularization parameter controls the compromise between the degree of smoothness of the solution and its closeness to the data. The mathematical results show that the solution of this new problem is close to the original one under a small change in $\boldsymbol{u}$ and is stable, since the class of possible solutions is narrowed through the introduction of the regularizer $\Omega(w)$ with the properties described above [TiAr77].

Regularization for Nonlinear Systems

Frequently, ill-conditioned or singular systems also arise in the iterative solution of nonlinear systems or optimization problems. We have shown in Chapter 2 that the iterative solution to neural network parameters using Newton's method can be expressed as:

$$\boldsymbol{H}(w)\Delta w = -\nabla F(w), \tag{39}$$

which is a linear system analogous to $\boldsymbol{A}w = \boldsymbol{u}$. Therefore, as we discussed in the linear case, any uncertainty associated with Hessian matrix $\boldsymbol{H}(w)$ and gradient vector $\nabla F(w)$ will affect the solution of Δw. When the Gauss-Newton approximation to the Hessian matrix is adopted in the training with the Levenberg-Marquardt implementation, we can avoid the singular problem in finding the inverse of $\boldsymbol{H}(w)$. However, the noise oriented uncertainty on the right-hand side of Eq. (39) is still not compensated, which may consequently lead to ill-determined Δw.

In addition, many neural network models are characterized by the high dimension of parameter w, which is far from the goal of parsimony in the conventional view of statistical modeling. A well known result from estimation theory shows the relationship between the model variance and the number of model parameters [Ljun87]. Assume that the observed data can be described by the true function plus white noise, i.e.,

$$y(i) = Y(x(i), w) + e(i), \tag{40}$$

and the model parameter w is estimated using the standard least-squares method using n training patterns. For many realizations with same sample size, the variance of the model error over many different estimates of w can be approximated as:

$$\overline{V_n} \approx \sigma^2\left(1 + \frac{N}{n}\right), \tag{41}$$

where σ^2 is the variance of the white noise, and N is the number of parameters. Eq. (41) says that regardless of how important a certain parameter is to the data fit, its contribution to the variance is σ^2/n. This indicates that using the standard optimization method may lead to overfitting if the model does not explain the behavior of the available training data in a parsimonious and statistically adequate manner.

As in linear system applications, regularization techniques can also be used to effectively reduce variance and to remedy numerical problems in nonlinear systems. Let's adopt the form of Eq. (38) as the regularized training performance function and use sum-of-squared parameters as the regularizer, then we have

$$F = E^D + \alpha\Omega(w) = e^T(w)e(w) + \alpha w^T w. \tag{42}$$

The gradient vector $\nabla F(w)$ is

$$\nabla F(w) = 2J^T(w)e(w) + 2\alpha w, \tag{43}$$

and the Gauss-Newton approximation to the Hessian becomes

$$H(w) = 2J^T(w)J(w) + 2\alpha I_N. \tag{44}$$

These expressions are different from those used in the standard (unregularized) training algorithms as we discussed in Chapter 2, and will affect the iterative solution of the weight parameters. Apparently, training will move along a smooth descent direction, and the problem of Jacobian deficiency will be remedied if the value of α is not too small. Using the Levenberg-Marquardt algorithm, the weight update equation can be expressed as:

$$w_{k+1} = w_k - [J^T(w_k)J(w_k) + (\alpha + \mu_w)I]^{-1}[J^T(w_k)e(w_k) + \alpha w_k]. \tag{45}$$

Eq. (45) is important in optimizing the regularized performance function for neural networks. Many resulting models trained with noisy data have demonstrated good generalization, provided that the value of parameter α is properly determined [Fore96]. Similar to the form of Eq. (41), the regularized model quality with respect to the variance can be approximated by

$$\overline{V_n} \approx \sigma^2\left(1 + \frac{\gamma}{n}\right), \tag{46}$$

with γ being the effective number of parameters [Sjob95] [Mood92]. Recall that

$$\gamma = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i + \alpha}, \tag{47}$$

in which λ_i is the eigenvalue of the Hessian matrix. Since α is a positive coefficient,

$\lambda_i/(\lambda_i+\alpha)$ will be close to 1 if the eigenvalue λ_i is significantly larger than α, and close to zero if λ_i is significantly smaller than α. In neural network applications, λ_i usually spans a wide range. Even though a large number of parameters is used in modeling, the effective number of parameters, γ in Eq. (47), might be significantly smaller than the total number of parameters N. Therefore, the model variance with the use of regularization can be dramatically reduced compared with the model variance using the standard optimization algorithms.

Determination of Parameter α

If we choose a regularized performance function to train a neural network model, the choice of the regularization parameter α will determine the model complexity, smoothness and the distribution of the weights. The difficulty of determining the value of α is problem dependent. For some cases, when the level of inaccuracy in measurements is known, we can determine the regularization parameter accordingly [TiAr77]. We take a monotonic sequence of α's, find each corresponding solution of $\boldsymbol{w}$, and choose α_δ for which the sum-of-squared error has the required accuracy, i.e., $E^D = \delta^2$.

Let's consider an example of a function approximation problem and show how to determine the regularization parameter using the discrepancy method. The true function is a shifted sinusoidal curve defined by

$$y = 1 + \sin\left(\frac{\pi}{4}x\right). \tag{48}$$

Assume that we have 51 samples evenly distributed from -2 to 6 over the input x. The measurements in outputs are corrupted with random Gaussian noise of zero mean and 0.01 variance. For a specific realization, the sum-of-squared error between the noise output and the true function output is 0.435. Let's define the inaccuracy δ^2 to be 0.45, and train a 1-15-1 neural network model with $\alpha = 0, 0.0001, 0.001, 0.01, 0.1, 1.0$ respectively. The resulting E^D, $\Omega(w)$ and F are summarized in Table 1.

parameter α	$E^D = e^T(w)e(w)$	$\Omega(w) = w^T w$	$F = E^D + \alpha\Omega(w)$
0	0.088	3328.9	0.088
0.0001	0.274	262.5	0.300
0.001	0.340	14.3	0.354
0.01	0.349	10.5	0.454
0.1	0.439	7.8	1.219
1.0	1.511	5.1	6.611

Table 1 Determining α from Discrepancy

For different choices of α, we have different performance functions. As the regularization parameter α increases, E^D and F are nondecreasing, and $\Omega(w)$ is nonincreasing. Therefore, we cannot determine the optimal α based on these functionals only. Additional constraints must be considered for decision making. Using discrepancy method, the additional information is the prior knowledge about the inaccuracy of the measurements which is 0.45 in this example. If the the resulting E^D is much less than 0.45, the selected α is too small, indicating overfitting. If the resulting E^D is much greater than 0.45, the selected α is too large, indicating underfitting. Choosing $\alpha = 0.1$, the corresponding E^D is 0.439 which is the closest one to the required accuracy.

Figure 7 shows the model behavior with four α values. Obviously, overfitting occur for $\alpha = 0$ and $\alpha = 0.0001$. When a large α is chosen, $\alpha = 1.0$ for example, E^D increases quickly, leading to a large bias which makes the network output on average different from the true function. Since $\alpha = 0.1$ produces adequate accuracy, it is a reasonable solution to the given problem.

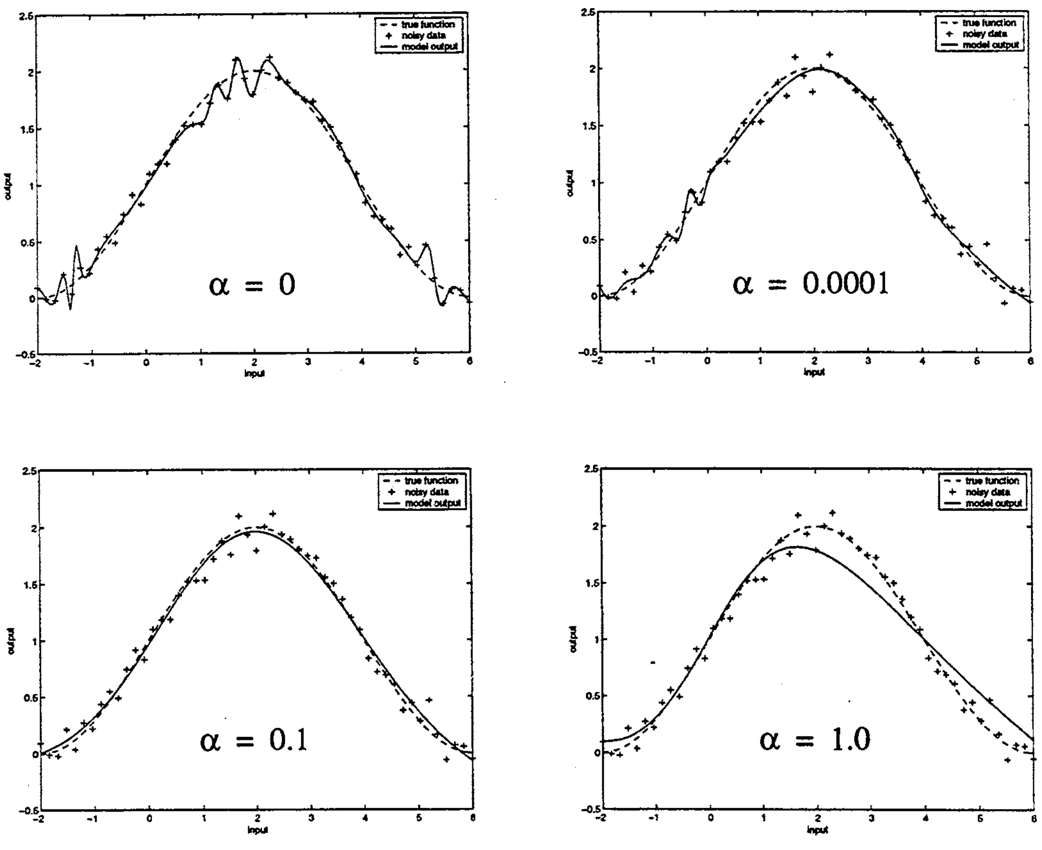


Figure 7 Determining α from Discrepance

The discrepancey method is based on the assumption that the measurement inaccuracy is known. There cannot be absolutely reliable ways to determine regularization parameter α in the absence of information about the size of the noise level and the degree of smoothness. However, this lack of information is very common in practice. Some stochastic techniques are reported for linear systems in the absence of information about the error level [Neum98]. In the stochastic approach, the regularization parameter is determined through variance component estimation methods. Since these methods are computationally expensive, they are not suitable for nonlinear neural network applications.

One successful method for determining the smoothness of the model is Bayesian regularization [MacK92] [FoHa97]. In the Bayesian method, both the model errors and the model weights are described by certain probabilistic distributions. The parameters of the distributions are automatically updated during the iterative learning process. They will finally match the training data in a statistic adequate manner under the given assumptions. In the next section, we will discuss the Bayesian regularization method in neural network modeling.

Bayesian Regularization

Bayesian regularization uses the concepts of Bayesian statistics to train neural networks. In this approach, the regularized performance function can be written as:

$$F = \beta E^D + \alpha\Omega(\mathbf{w}) \tag{49}$$

In this expression, both the training error and the network weights are random variables. The performance function parameters, β and α, are the variance related parameters. Using the Bayesian method, the network training is performed in a hierarchical fashion. The first level involves the determination of the most probable network weights for the given β and α. At the second level, the parameters β and α are optimized to maximize the evidence on the training data. These two-level operations can be performed alternately during the training. Now let's see how to implement the Bayesian regularization from the fundamental Bayes' rule.

Determining Most Probable Weights

Bayesian regularization is based on Bayes' rule, which is expressed in the form of a conditional probability density function. Assuming the weights of the network are random variables, the posterior density function of the weights can be updated after the data is taken:

$$P(w|D, \alpha, \beta) = \frac{P(D|w, \beta)P(w|\alpha)}{P(D|\alpha, \beta)} \tag{50}$$

where D represents the data set, w is the vector of the network weights. $P(w|\alpha)$ is the prior density, which represents our knowledge of the weights before any data is collected. $P(D|w, \beta)$ is the likelihood function, which is the joint probability of the data occurring, given the weights w. The term in the denominator, $P(D|\alpha, \beta)$, is a normalization factor, which guarantees that the total probability is 1.

In this implementation, we assume that the noise in the training set data with n measurements is independent. Therefore, the joint probability density function can be written as

$$P(D|w, \beta) = P(e_1, e_2, \ldots, e_n|w, \beta) = \prod_{i=1}^{n} P(e_i|w, \beta). \tag{51}$$

For the Gaussian noise with zero mean and variance of σ^2,

$$P(e_i|w, \beta) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{e_i^2}{2\sigma^2}\right), \tag{52}$$

Eq. (51) then becomes

$$P(D|w, \beta) = \frac{1}{(2\pi)^{n/2}\sigma^n}\exp\left(-\frac{\sum e_i^2}{2\sigma^2}\right) = \frac{1}{Z_D(\beta)}\exp(-\beta E^D), \tag{53}$$

where $\beta = \frac{1}{2\sigma^2}$, $E^D = e^T(w)e(w)$, and $Z_D(\beta) = (\pi/\beta)^{n/2}$.

Similarly, if we assume that the prior density of the weights is independent and Gaussian distributed with zero mean and variance of σ_w^2, then

$$P(w|\alpha) = \frac{1}{Z_W(\alpha)}\exp(-\alpha\Omega(w)), \tag{54}$$

with $\alpha = \frac{1}{2\sigma_w^2}$, $\Omega(w) = w^T w$, and $Z_W(\alpha) = (\pi/\alpha)^{N/2}$. Under these assumptions, Eq. (50) can be simplified as

$$P(w|D, \alpha, \beta) \propto \exp(-(\beta E^D + \alpha\Omega(w))), \tag{55}$$

which indicates that the posterior density function of the weights belongs to the same Gaussian family as the prior density of weights does. For the given prior distribution, its posterior distribution depends on the observed data only through βE^D.

In this Bayesian framework, the optimal weights, which are also called the most probable weights w_{MP}, should maximize the posterior density $P(w|D, \alpha, \beta)$. As we can see from Eq. (55), maximizing the posteriori density is equivalent to minimizing the regularized performance function $F = \beta E^D + \alpha\Omega(w)$. The iterative solution to the weight update can be found using the Gauss-Newton method with the Levenberg-Marquardt implementation [FoHa97]:

$$w_{k+1} = w_k - [\beta J^T(w_k)J(w_k) + (\alpha + \mu_w)I]^{-1}[\beta J^T(w_k)e(w_k) + \alpha w_k] \ . \tag{56}$$

Optimizing α ***and*** β

Now we consider the second level Bayesian analysis which applies Bayes' rule to optimize the performance function parameters β and α. If we assume that α and β are random variables, then their posterior density function conditional on the given data set can be expressed as

$$P(\alpha, \beta|D) = \frac{P(D|\alpha, \beta)P(\alpha, \beta)}{P(D)} \tag{57}$$

Since $P(D)$ is a normalization factor, we can simplify Eq. (57) as

$$P(\alpha, \beta|D) \propto P(D|\alpha, \beta)P(\alpha, \beta). \tag{58}$$

If we assume a uniform prior density $P(\alpha, \beta)$ for parameters α and β, then

$$P(\alpha, \beta|D) \propto P(D|\alpha, \beta), \tag{59}$$

which indicates that maximizing the posterior can be achieved by maximizing the likelihood function $P(D|\alpha, \beta)$. Note that this likelihood function is the normalization factor for Eq. (50). This factor is called the evidence for α and β. Solving Eq. (50) for the normalization factor, we have

$$P(D|\alpha, \beta) = \frac{P(D|w, \beta)P(w|\alpha)}{P(w|D, \alpha, \beta)}. \tag{60}$$

Recall that the posterior density function of weights is within the Gaussian family, and can be expressed as

$$P(w|D, \alpha, \beta) = \frac{1}{Z_F(\alpha, \beta)}\exp(-(\beta E^D + \alpha\Omega(w))). \tag{61}$$

Putting Eq. (53), Eq. (54) and Eq. (61) into Eq. (60), we obtain

$$P(D|\alpha, \beta) = \frac{Z_F(\alpha, \beta)}{Z_D(\beta) Z_W(\alpha)}. \tag{62}$$

To solve $Z_F(\alpha, \beta)$ on the right-hand side of Eq. (62), let's use the second order Taylor series expansion to approximate the regularized performance function $F = \beta E^D + \alpha\Omega(\boldsymbol{w})$ around the most probable weight vector $\boldsymbol{w}_{MP}$,

$$F \approx F(\boldsymbol{w}_{MP}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_{MP})^T \boldsymbol{H}_{MP}(\boldsymbol{w} - \boldsymbol{w}_{MP}), \tag{63}$$

then the Gaussian approximation to the posterior distribution of the weights becomes

$$P(\boldsymbol{w}|D, \alpha, \beta) \approx \exp(-F(\boldsymbol{w}_{MP}))\frac{1}{Z_F(\alpha, \beta)}\exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_{MP})^T \boldsymbol{H}_{MP}(\boldsymbol{w} - \boldsymbol{w}_{MP})\right). \tag{64}$$

Note that the standard multivariate Gaussian distribution has a form of

$$P(\boldsymbol{w}) = \frac{1}{(2\pi)^{N/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_{MP})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{w} - \boldsymbol{w}_{MP})\right) \quad , \tag{65}$$

where $\boldsymbol{\Sigma}$ is the variance-covariance matrix of multivariate $\boldsymbol{w}$, which is approximated by the inverse of the Hessian matrix $\boldsymbol{H}_{MP}$ in our application. We can obtain Z_F by comparing the coefficients of Eq. (64) and Eq. (65)

$$Z_F = (2\pi)^{N/2}(\det(\boldsymbol{H}_{MP}^{-1}))^{1/2}\exp(-F(\boldsymbol{w}_{MP})). \tag{66}$$

Now every term on the right-hand side of Eq. (62) is known. The optimal α and β which maximize the evidence can be determined by seeking the extreme value of the log of Eq. (62) using the standard zero-partial-derivative method [Bish96] [Fore96]. The solutions are given below:

$$\alpha_{MP} = \frac{\gamma}{2\Omega(w_{MP})} \quad (67)$$

$$\beta_{MP} = \frac{n-\gamma}{2E^D(w_{MP})} \quad (68)$$

where

$$\gamma = N - \alpha \mathrm{tr}(H_{MP}^{-1}) \quad (69)$$

is called the effective number of parameters. Since this number is smaller than the total number of parameters for ill-posed problems, the predicting variance of the model will be reduced. $H = \beta\nabla^2 E^D + \alpha\nabla^2\Omega(w)$ is the Hessian matrix of the regularized performance function. In practice, the Hessian matrix can be approximated using the Gauss-Newton method, i.e., $H \approx 2\beta J^T(w)J(w) + 2\alpha I_N$.

Training Example

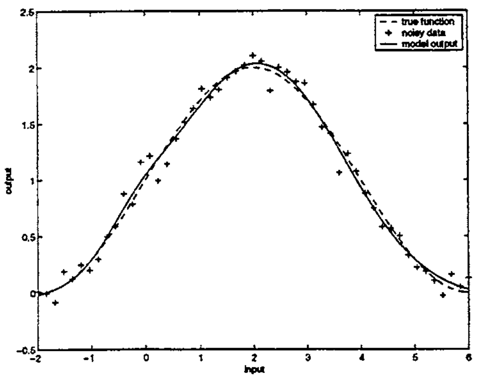
Let's consider the same shifted sinusoidal function given in Eq. (48), and train a neural network model with the noisy data set using the Bayesian regularization method. The algorithm required for Bayesian optimization of the performance function parameters can be implemented in different ways [Bish96] [Fore96]. Here we update the network weights and the function parameters alternately in the following steps [FoHa97]:

1. Initialize α, β and weights. Using $\alpha \approx 0$ and $\beta \approx 1$ for most applications.
2. Take one step of training to minimize $F = \beta E^D + \alpha\Omega(w)$ by using Eq. (56).
3. Compute the effective number of parameters γ using Eq. (69), making use of the Gauss-Newton approximation to the Hessian matrix.

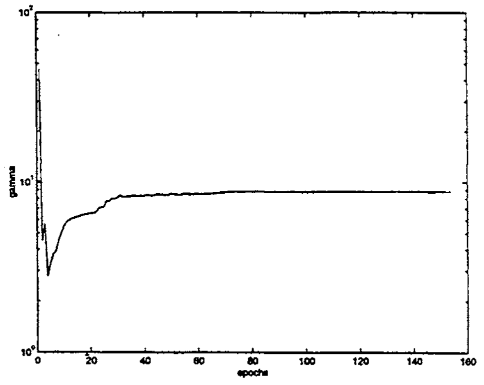
4. Compute new estimates of α and β using Eq. (67) and Eq. (68).

5. Iterate step 2 through 4 until convergence.

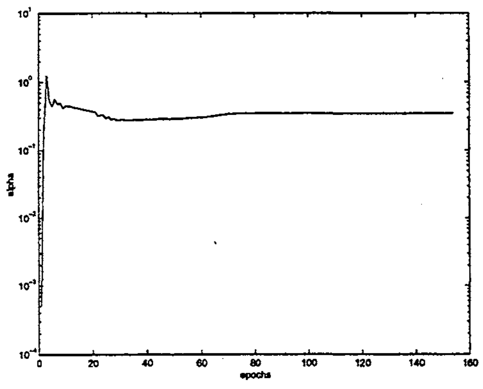
The training results are illustrated in Figure 8. Figure 8 (a) shows that the model trained with the Bayesian regularization algorithm produces a smooth function approximation to the true function with the presence of noise in the measurements.



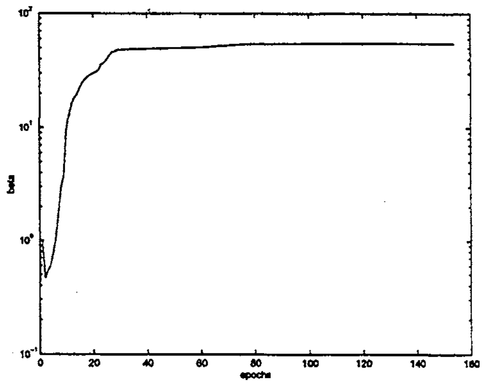
(a). Training results



(b). Parameter γ



(c). Parameter α



(d). Parameter β

Figure 8 Training Example with Bayesian Regularization

Figure 8 (b) illustrates how the effective number of parameters changes during the training. Although we use a 1-15-1 network model which has 46 parameters in total, the effective number of parameters converges to 9 by the end of the training. The other superfluous parameters which did not contribute to the error reduction no longer have a side effect on the model performance.

Figure 8 (c) presents a trajectory of the parameter α which is automatically adapted in each training epoch. The initial α is set to 0.0005 ($\sigma_w^2 \approx 30$), assuming that the network weights spread over a wide range. After the optimization converges, the optimal α is 0.35 ($\sigma_w^2 \approx 1.2$), indicating a more concentrated posterior density function of the weights.

The different values of the parameter β in Figure 8 (d) reflect the noise assessment during the iterative training. A small β value suggests a large variance in the training error, and a large β value indicates a small variance in the training error. In this example, we add a random Gaussian noise sequence (zero mean and 0.01 variance) to the function outputs. The resulting β is about 55 ($\sigma^2 \approx 0.009$), which is close to the underlying noise distribution.

Features and Limitations

Some important features of the Bayesian method have been discussed in the literature [BuWe91][MacK92][MacK94][Bish95][Neal96][Fore96][FoHa97], which demonstrate that Bayesian regularization is very practical for neural network training. One of these features is making an efficient use of the available training data to produce a probabilistic model based on prior assumptions. Using priors in the Bayesian framework allows a shift in the weight space toward a set of weights yielding a smooth network response. Different priors may give different probabilistic regimes for modeling this shift. However, the Gaussian prior for the network weights is found to be simple and robust and produces reasonable results most of the time.

In addition, Bayesian regularization can be implemented in different ways under the same framework. Although we discussed only a single class of weights in this chapter, the results for complicated problems can be improved by dividing the total weights into several classes, each class with its own regularization constant. Using multiple Gaussian priors makes it possible to build a huge, flexible model with a large number of parameters, and to control the complexity of the model with multiple regularization parameters. This will enable the model to capture the local smoothness of the problem and approximate the properties of the training data better.

One controversial issue associated with the Bayesian regularization is whether training the network with all available data is an advantage. In common practice, the Bayesian model is constructed with explicit assumptions including the Gaussian prior for the network weights, the Gaussian noise model, and the uniform density function for α and β. These assumptions, however, can be quite hard to justify. Bayesian network users may have difficulty in explaining their selections for the models and priors, because no matter what problem it is, the training is always performed to capture the specified prior beliefs. As a consequence, the generalization error which is measured on new testing data may not relate to the training results in a simple way, which is our primary interest in later chapters. Still, using all available data in the training is computationally expensive, if the Jacobian matrix is too large.

In Chapter 4, we will introduce the use of cross-validated early stopping in neural network modeling. This procedure is computationally efficient, does not need any statistical assumptions, and can be used with any of the training performance functions.

CHAPTER 4

CROSS-VALIDATED EARLY STOPPING

Objectives

The technique of cross-validated early stopping is widely used in neural network modeling, but its effectiveness is very problem-dependent. The features of early stopping will be addressed in this chapter. In order to improve early stopping techniques, we propose and demonstrate a new procedure called retrained early stopping.

Introduction

The regularization method introduced in the previous chapter uses a modified performance function to control the effective complexity of the neural network. In this chapter, we examine a different method which improves generalization by stopping the training early, before overfitting occurs.

The stopping rule has been proposed based on cross-validation, which will be described at the beginning of this chapter. Then the relationship between early stopping and regularization will be briefly discussed.

To better understand cross-validation, we will introduce its statistical properties in the asymptotic case of a large number of training examples. The non-asymptotic features of early stopping, however, can only be investigated empirically, and are very problem-dependent. We will investigate the dependence of effective early stopping on several influential factors.

Conventional early stopping uses only part of the available data for parameter estimation, which often prevents the generated network from achieving superior performance. To overcome this weakness, we will propose and demonstrate a new procudure which allows a combined retraining after the conventional early stopping.

Basic Concepts

As we stressed in Chapter 2, the goal of neural network training is not to memorize the training examples, but rather to model the underlying function which generates the data, so that the best possible prediction can be made when the trained network is subsequently presented with new inputs. This task is usually difficult if the training performance is measured by the sum-of-squared errors and the measurements are corrupted with noise, or if an overly complicated network structure is used. In practice, the mapping between training inputs and outputs is roughly approximated during the first stage of training. The details are gradually refined as training proceeds. Therefore, it is widely believed that the generalization error decreases as the common features of the problem are learned in the early period of training, but then later increases as training goes on. Usually, the increase in the generalization error indicates that further training will lead to overfitting, even though the training error will monotonically decrease. To avoid overfitting, it is considered better to stop training at an adequate time before the generalization error increases. This technique is often referred to as early stopping.

General Description

The following simple stopping rule has been proposed based on cross-validation. Divide all the available examples into two disjoint sets. One set is used for training. The other set is used for validation. The behavior of the trained network is evaluated by using the cross-validation examples, and the training is stopped at the point that minimizes the error on the validation set. Note that the assumption behind the cross-validated early stopping is that the validation set is a reasonable representative of unknown new data sets, and, therefore, the validation error is an estimate of the generalization error.

Early stopping can be applied to any standard optimization algorithm which minimizes the sum-of-squared error training function. In each training epoch, the network weights are updated using the training data set, then the validation error is computed using the latest updated weights. The smallest validation error and the corresponding network weights are saved. These weights are usually overwritten in each training epoch during the early period of training, because the validation error is descending along with the training error. This can be shown in Figure 9 in which the validation error monotonically decreases before the point A.

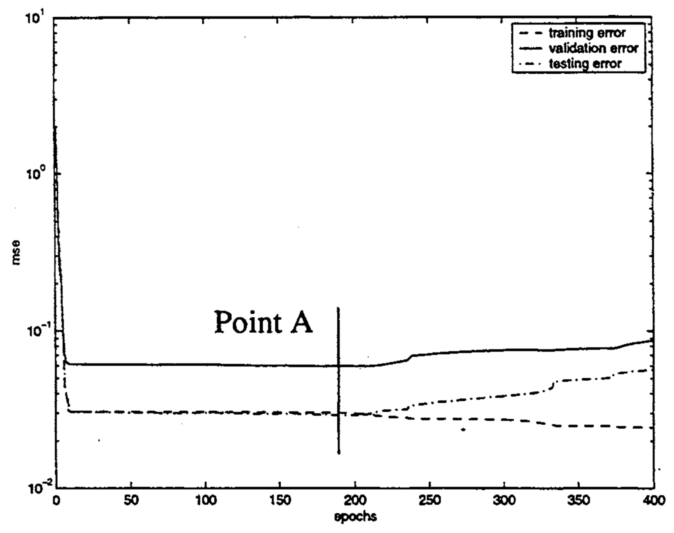


Figure 9 Cross-validated Early Stopping

In Figure 9, the validation error reaches a minimum at the point A. After that, the training error keeps decreasing as the training refinement continues, but the validation error increases. Further refinement on the training data might be noise-oriented, which is random in nature and not related to data in the validation set. Therefore, the training is terminated at the point A, since continued training will only lead to overfitting. If the selected validation set is really a good representative of new data (as represented by the testing error curve in Figure 9), then the network obtained with early stopping will have the minimum generalization error.

Note that the cross-validation data set described above is only used passively to stop the training. The weight update is computed to minimize the training error only. In the next chapter we will discuss a training procedure that actively uses the validation set.

We have assumed in this chapter that the generalization error of a network trained on the full data set, using an unregularized performance function, is always high. This is not always the case, as is discussed in [AmMu97]. We will explain later in this chapter that cross-validated early stopping is useless in the asymptotic region where the number of training examples is much lager than the number of network parameters.

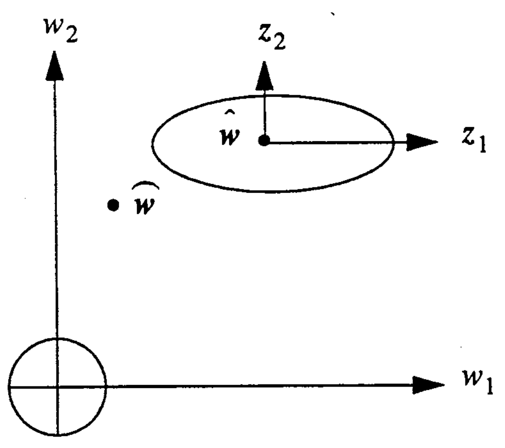
Relation to Regularization

Early stopping is methodically different than regularization. However, the effects of the two techniques are similar. Sjöberg [Sjöb95] called early stopping "implicit regularization," and showed that the number of the training iterations was inversely proportional to the regularization parameter. Bishop [Bish95] graphically demonstrated the analogy between regularization and early stopping.

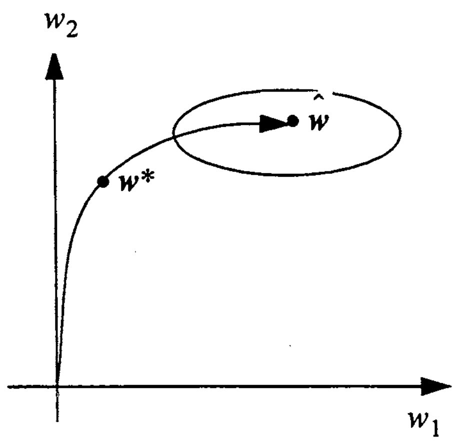
To explain this analogy, let's assume that we have a two-dimensional weight space, and the unregularized error function has a quadratic form:

$$E^D(\boldsymbol{w}) = E_0^D + \boldsymbol{b}^T\boldsymbol{w} + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{H}\boldsymbol{w}. \qquad (70)$$

The effect of a simple weight-decay regularizer $\Omega(\boldsymbol{w})$ on a quadratic error function is illustrated in Figure 10 (a). The circle represents a contour along which the weight-decay term is constant, the ellipse represents a contour of constant unregularized error. $\hat{\boldsymbol{w}}$ is the weight vector corresponding to the minimum of $E^D(\boldsymbol{w})$. z_1 and z_2 are the eigenvectors of the Hessian matrix $\boldsymbol{H}$ which determine the length of the long and short axis of the ellipse.



(a). Regularization



(b). Early stopping

Figure 10 Analogy between Regularization and Early Stopping

For simplicity, we rotate the axes in weight space to be parallel with z_1 and z_2. When we add $\alpha\Omega(\boldsymbol{w})$ to $E^D(\boldsymbol{w})$, the effect of the regularizer is to shift the minimum of the error function from $\hat{\boldsymbol{w}}$ to $\widetilde{\boldsymbol{w}}$. The relation between the minima of the original and the regularized error functions [Bish95] can be expressed as

$$\widetilde{w}_j = \frac{\lambda_j}{\lambda_j + \alpha}\hat{w}_j , \tag{71}$$

where the λ_j are the eigenvalues of the Hessian matrix $\boldsymbol{H}$. Since the eigenvalue value λ_1 is smaller than λ_2, the ratio of $\lambda_1/(\lambda_1 + \alpha)$ is greater than $\lambda_2/(\lambda_2 + \alpha)$. Therefore, the value of $\widetilde{w}_1$ at the minimum is significantly reduced, and the value of $\widetilde{w}_2$ is only slightly affected by the regularization.

Early stopping produces results that are similar to regularization. This can be understood from Figure 10 (b). The weight vector starts at the origin and proceeds during training along a path which follows the local descent direction. In this example, the eigenvalues of the Hessian differ widely, which can be seen from the shape of the ellipse. Thus, the weight vector will move initially parallel to the w_2 axis to a point corresponding roughly to w^* and then move towards the minimum of the error function $\hat{w}$. Stopping at a point near w^* is therefore similar to $\widehat{w}$, which is obtained with regularization

Although early stopping is qualitatively similar to regularization, the quantitative analysis of early stopping is very difficult. However, under the asymptotic condition in which a large number of samples are available, we can statistically compare early stopping with the full set training method, and give a quantitative description of the each generalization error.

Asymptotic Cross-Validation Properties

In this section, we will introduce the asymptotic cross-validation theory mainly investigated by Amari et al. [AmMu93] [AmMu96] [AmMu97]. This statistical theory explains when cross-validation is asymptotically effective and describes how to divide the total samples into training and validation sets to obtain the optimum performance. Although the asymptotic region (large sample size) is often inaccessible in practical applications, we will see a study of this region makes it possible to exactly calculate the generalization error for hypothetical models. Moreover, investigating asymptotic statistical properties of cross-validation will help us to understand better the advantages and limitations of early stopping in practical applications. In this section, instead of showing Amari's derivations and mathematical proofs, we will mainly review the results of their study and the assumptions made in their calculation.

Stochastic Network and Asymptotic Learning

In Amari's study, a stochastic network was used and the network performance was evaluated by the average predictive entropy.

Assume that there exists a teacher network $N(\boldsymbol{w}_0)$ which generates training examples. Each input $\boldsymbol{x}_i$ is randomly chosen from an unknown probability $P(\boldsymbol{x}_i)$. The input-target relation of the network is specified by the conditional probability $P(y_i|\boldsymbol{x}_i, \boldsymbol{w}_0)$, where $\boldsymbol{w}_0$ is the true network weight vector. The joint probability of $(\boldsymbol{x}_i, y_i)$ of $N(\boldsymbol{w}_0)$ is given by

$$P(\boldsymbol{x}_i, y_i;\boldsymbol{w}_0) = P(\boldsymbol{x}_i)P(y_i|\boldsymbol{x}_i, \boldsymbol{w}_0). \tag{72}$$

Assume that the training set D_n has n independent patterns generated by the distribution $P(y|\boldsymbol{x}, \boldsymbol{w}_0)$, and n is asymptotically large. A student network having the same number of parameters is trained to learn the input-target mapping from the training set

$$D_n = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\} . \tag{73}$$

The training function is defined by

$$R_{\text{train}}(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n} \log P(\boldsymbol{x}_i, y_i;\boldsymbol{w}), \tag{74}$$

which is the entropy of the student network. Let us denote $f(\boldsymbol{x}_i;\boldsymbol{w})$ as the output calculated by the multilayer feedforward network $N(\boldsymbol{w})$ for the given input $\boldsymbol{x}_i$ and weights $\boldsymbol{w}$. The difference $y_i - f(\boldsymbol{x}_i;\boldsymbol{w})$ is Gaussian with zero mean and σ^2 variance. Then

$$P(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2\sigma^2}|y_i - f(\boldsymbol{x}_i;\boldsymbol{w})|^2\right\}. \tag{75}$$

From Eq. (72), we obtain

$$-\log P(x_i, y_i; w) = \frac{1}{2\sigma^2}|y_i - f(x_i; w)|^2 + c(x), \tag{76}$$

where $c(x)$ does not depend on w. The training performance function defined by Eq. (74) therefore measures the mean-squared-error.

The generalization error is the average predictive entropy of a trained network for a new example. The asymptotic theory of statistics proves that the maximum likelihood estimator $\hat{w}$ is asymptotically normally distributed with mean w_0 and variance $\frac{1}{n}G^{-1}$, where G is the Fisher information matrix. Its elements $G_{ij}(w)$ are defined by

$$G_{ij}(w) = E\left[\frac{\partial}{\partial w_i}\log P(x, y; w)\frac{\partial}{\partial w_j}\log P(x, y; w)\right], \tag{77}$$

in which E denotes the expectation with respect to $P(x, y; w)$. When $\hat{w}$ belongs to the $1/\sqrt{n}$ neighborhood of w_0, the Taylor expansion of the training and the generalization error can be expressed as

$$R_{\text{train}}(\hat{w}) \approx \hat{H}0 - \frac{1}{2}(\hat{w} - w_0)^T G(w_0)(\hat{w} - w_0), \tag{78}$$

and

$$R_{\text{gen}}(\hat{w}) \approx H_0 + \frac{1}{2}(\hat{w} - w_0)^T G(w_0)(\hat{w} - w_0). \tag{79}$$

where H_0 denotes the entropy of the teacher network

$$H_0 = -E[\log P(x, y; w_0)]. \tag{80}$$

This is an unknown constant and indicates the stochastic uncertainty of the teacher network itself. $\hat{H}0$ is the estimate of H_0 from the training examples. $G(w_0)$ is the Fisher matrix, which is the expectation of the second derivative of $-\log P(\boldsymbol{x}, y; \boldsymbol{w}_0)$ with respect to $\boldsymbol{w}_0$.

Considering the distribution of $\hat{\boldsymbol{w}}$ with different D_n, the entropic training error is expressed as the asymptotic expectation of $R_{\text{train}}(\hat{\boldsymbol{w}})$

$$\langle R_{\text{train}}(\hat{\boldsymbol{w}}) \rangle \approx H_0 - \frac{N}{2n}, \tag{81}$$

and the entropic generalization error is expressed as the asymptotic expectation of $R_{\text{gen}}(\hat{\boldsymbol{w}})$ [AmMu93]

$$\langle R_{\text{gen}}(\hat{\boldsymbol{w}}) \rangle \approx H_0 + \frac{N}{2n}. \tag{82}$$

The training error is smaller than the generalization error. Since H_0 is unknown in general, we can relate generalization error to the training error,

$$\langle R_{\text{gen}}(\hat{\boldsymbol{w}}) \rangle = \langle R_{\text{train}}(\hat{\boldsymbol{w}}) \rangle + \frac{N}{n}. \tag{83}$$

We want to know if the generalization error decreases when we terminate the training before $\hat{\boldsymbol{w}}$ is reached.

Let us consider the gradient descent learning rule

$$\hat{\boldsymbol{w}}(k+1) = \hat{\boldsymbol{w}}(k) - \eta \frac{\partial}{\partial \boldsymbol{w}} R_{\text{train}}(\hat{\boldsymbol{w}}(k)). \tag{84}$$

It has been shown [AmMu97] that

$$\hat{\boldsymbol{w}}(t) = (1-\eta)^{t-k}\{\hat{\boldsymbol{w}}(k) - \hat{\boldsymbol{w}}\} + \hat{\boldsymbol{w}}, \qquad t > k, \tag{85}$$

thus, the trajectory $\hat{\boldsymbol{w}}(t)$ linearly approaches $\hat{\boldsymbol{w}}$ in the neighborhood of $\hat{\boldsymbol{w}}$. We call $\hat{\boldsymbol{w}}(t)$ a ray. The direction from which the ray $\hat{\boldsymbol{w}}(t)$ approaches $\hat{\boldsymbol{w}}$ depends on the initial $\hat{\boldsymbol{w}}(0)$. In our calculation, the ensemble average $\langle R_{gen}(\hat{w}(t))\rangle$ is the expectation with respect to the initial condition $\hat{w}(0)$ and with respect to different realizations of D_n, therefore, we can assume that $\hat{w}(t)$ approaches $\hat{w}$ isotropically.

Virtual Optimal Stopping Point

Now, let us go back to Eq. (79). The Taylor expansion of the generalization error can be rewritten as

$$R_{\text{gen}}(t) \approx H_0 + \frac{1}{2}\left|\hat{w}(t) - w_0\right|^2 . \tag{86}$$

Here we take the coordinate system such that the Fisher information matrix in Eq. (79) is equal to the identity matrix at w_0, i.e., $G(w_0) = I$. As we discussed before, the true w_0 and $\hat{w}$ are in general different, and the distance between them is of order $1/\sqrt{n}$. To visualize the relations among the different weight estimates, we compose a sphere centered at $(\hat{w} + w_0)/2$, as is shown in Figure 11.

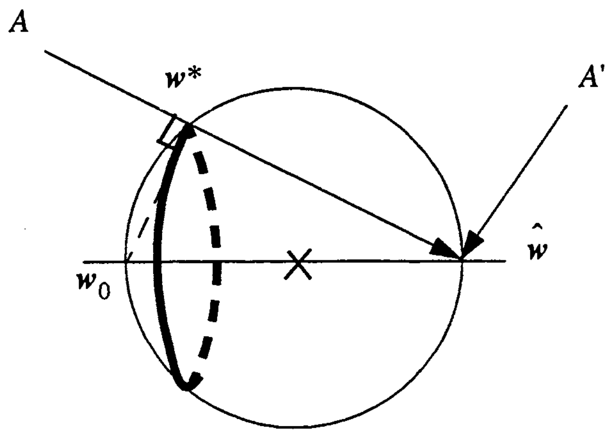


Figure 11 Geometric Picture to Determine w^*

Assuming that A is a ray, which is the trajectory of $\hat{w}(t)$, the optimal stopping point w^* will be the one that minimizes the generalization error. As $\hat{w}(t)$ moves along the ray A (see Figure 11), the generalization error $R_{\text{gen}}(t)$ is minimized at the point w^* where the line between w^* and w_0 is orthogonal to the ray A. The point w^* is the first intersection of the ray A and the sphere. When ray A' is approaching $\hat{w}$ from the opposite side of w_0, the first intersection point is $\hat{w}$ itself. In this case, the optimal stopping point is $\hat{w}$.

It can be proved [AmMu97] that, when $\hat{w}(t)$ approaches $\hat{w}$ isotropically, the average generalization error at the optimal stopping point is asymptotically given by

$$\langle R_{\text{gen}}(t_{opt})\rangle \approx H_0 + \frac{1}{2n}\left(N - \frac{1}{2}\right). \tag{87}$$

Comparing Eq. (87) with Eq. (82), if we could know the optimal stopping time t_{opt} for each trajectory, the generalization error decreases by $1/4n$, which has the effect of decreasing the effective dimensions by $1/2$. This effect is negligible when N is large. However, it is impossible to know the optimal stopping time when using the full data set in training. If we stop learning at an estimated optimal time, we have some gain when ray A is from the same side as w_0, but we have some loss when ray A is from the opposite direction. Since the optimal stopping time can not be determined from the training set with the use of the unregularized optimization algorithm, an indirect method to estimate the optimal stopping time is to use the validation set.

Cross-Validated Optimal Stopping

Assume that n samples are divided into $r \cdot n$ examples of the training set and $r' \cdot n$ examples of the validation set. The data splitting ratio is defined by $r' \cdot n / r \cdot n$, where $r + r' = 1$. Let $\hat{w}$ be the weight vector which minimizes the training error function, and let $\tilde{w}$ the weight vector which minimizes the validation error function. Since the training examples and cross-validation examples are independent, both $\hat{w}$ and $\tilde{w}$ are asymptotically subject to independent normal distributions with mean w_0 and covariance matrices $G^{-1}/(r \cdot n)$ and $G^{-1}/(r' \cdot n)$, respectively. A new sphere can be composed in Figure 12 which is centered at $(\hat{w} + \tilde{w})/2$. The trajectory A enters $\hat{w}$ linearly in the neighborhood of $\hat{w}$. The point w^* on the ray A which minimizes the cross-validation error is the point on A that is closest to $\tilde{w}$. When ray A' is approaching $\hat{w}$ from the opposite side of $\tilde{w}$, the optimal stopping point is $w^* = \hat{w}$.

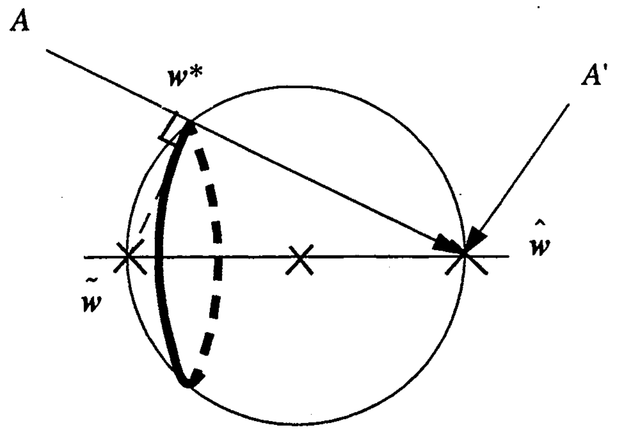


Figure 12 Geometric Picture of Cross-Validated Early Stopping

The average generalization error that can be achieved by the optimal early stopping can be expressed as

$$\langle R_{\text{gen}}(w^*, r)\rangle \approx H_0 + \frac{2N-1}{4r \cdot n} + \frac{1}{4r' \cdot n}. \tag{88}$$

The proof is given in [AmMu97]. Minimizing Eq. (88) with respect to r, we can obtain r_{opt} which is the optimal ratio of the training samples to the total samples

$$r_{\mathrm{opt}} = 1 - \frac{\sqrt{2N-1}-1}{2(N-1)}. \tag{89}$$

When N is large, r_{opt} can be simplified as

$$r_{\mathrm{opt}} \approx 1 - \frac{1}{\sqrt{2N}}, \tag{90}$$

which says that if the total number of samples is n, only $n/\sqrt{2N}$ samples are needed for the validation set to minimize the average generalization error. The optimal data splitting rartio is

$$\frac{r'_{\mathrm{opt}} \cdot n}{r_{\mathrm{opt}} \cdot n} = \frac{1}{\sqrt{2N}-1}. \tag{91}$$

If the number of network parameters N is large, the splitting ratio is small, meaning that a large percentage of samples should be put in the training set. Substituting Eq. (90) into Eq. (88) and making further simplification, we obtain the average generalization error

$$\langle R_{\mathrm{gen}}(w^*, r_{\mathrm{opt}})\rangle \approx H_0 + \frac{N}{2n}\left(1 + \sqrt{\frac{2}{N}}\right). \tag{92}$$

Recall that the generalization error given by Eq. (82) without the use of cross-validation (r=1) is

$$\langle R_{\mathrm{gen}}(\hat{w})\rangle \approx H_0 + \frac{N}{2n}.$$

This implies that the generalization error increases slightly when using cross-validated early stopping, compared with using all the examples for training.

Amari's study of cross-validation shows that cross-validated early stopping is not asymptotically effective in reducing the generalization error. However, this theory is not valid for applications where the number of samples n is not asymptotically large. To explain why cross-validated early stopping is effective in practice, the distribution of the initial $w(0)$ and the nonlinear learning trajectories should be considered. The complete theory is difficult to construct due to the complexity introduced by the limited data size. The optimal use of cross-validated early stopping is problem dependent.

Application Trade-off

We discussed the asymptotic cross-validation properties in the previous section. Now let us go back to practical applications.

Training neural networks with cross-validated early stopping has two advantages. First, no assumptions on the system, the model parameter distribution and the noise distribution are required. Secondly, compared with the computational cost in adapting regularization parameters, cross-validated early stopping is more efficient. In Bayesian regularization, for instance, updating α and β requires calculation of the trace of the inverse of the Hessian matrix. In early stopping, only the additional computation of the validation error function is required in each iteration. In addition, the total number of training epochs needed in early stopping is usually smaller than that required for adaptive regularization, especially when the noise levels are low. However, the optimal use of cross-validated early stopping depends on several important factors, which are the pattern/parameter ratio, the initial parameters and the data splitting ratio.

Effect of Pattern/Parameter Ratio

The pattern/parameter ratio is the ratio of the number of training examples n to the number of network parameters N. When n is less than or equal to N, the examples can in principle be memorized, and overfitting is possible. In this case, placing any samples in the validation set severely reduces the training set size. Early stopping does work for this case, but the variation in model performance is very sensitive to the choice of the validation data [KrHe92][ChHa98]. Instead of using a specific hold-out validation set, it is better to train different networks with different data splittings and average the prediction results.

If the pattern/parameter ratio is somewhat larger than 1, then early stopping is effective in general. This can be explained by the analogy between early stopping and regularization. However, there is no general theory (as we presented in the previous section) to fully support it. In practice, the condition $n > N$ is satisfied by many neural network applications, and there are many heuristics for using early stopping in this case.

If the pattern/parameter ratio is greater than 30, then the number of the training examples n can be considered asymptotic large [AmMu97]. In this case, the theory and the simulation results demonstrate that cross-validation is not necessary. The generalization error becomes worse when using cross-validation examples to determine the stopping time. This is not the typical situation encountered in neural network modeling. For the rest of this chapter, our discussion will focus on the intermediate pattern/parameter ratio case.

Effect of Initial Parameters

Recall that in Amari's asymptotic study, the optimal stopping point depends on the direction by which the training ray approaches convergence. Therefore, the distribution of the initial weights may play an important role in the training results. Starting from a specific initial condition, the procedure may stop too early due to a premature increase in the validation error. For a different choice of the initial weights, the trajectory of the validation error may be monotonically descending. This is not critical for the regularization method, if the regularization parameter is properly adapted during the training. In that case, each update of the regularization parameter leads to a new performance function. Since many different performance functions are trained with various conditions before the adaptive regularization converges, the influence of the initial weights on the final result becomes less significant.

To get more consistent results with early stopping, we may put some constraints on the initial weights so that the nonlinear learning trajectory is not isotropic. For example, the initial weights can be randomly selected from a small range to keep all activation values of neurons in the dynamic range of the transfer function [Lars93]. Then these weights might be further scaled to keep the initial variance for all activation values approximately equal [Wan93]. In practice, the necessary preprocessing of the inputs and the targets is performed along with the parameter initialization.

Effect of Data Splitting Ratio

For a fixed pattern / parameter ratio, the division of the available data into training set and validation sets often makes a difference in network performance. An obvious drawback to use cross-validated early stopping is that only part of the available data can be used for parameter estimation. The validation data set is passively withheld to determine the stopping point, but it does not contribute to calculating the weights. This implies a trade off in selecting the data splitting ratio n_v/n_t. On the one hand, the validation size n_v should be as small as possible to ensure a proper weight estimate; on the other hand, n_v should be as large as possible to ensure that E_v^D will be a reliable estimator of the generalization error. In general, there is no single fixed data splitting ratio accepted for all applications. Practically, we can refer to some existing rules to make a nearly optimal selection.

One rule, as recommended by Amari et al.[AmMu97], relates the data splitting ratio to the number of network parameters. It says we can use only $n/\sqrt{2N}$ samples in the cross-validation set. This ratio can be very small in the case of large N. The other rule [Lars93] suggests using a 1 / 1 data splitting ratio if we have no information about the function complexity and the confidence interval on the validation data set. Kearns [Kear97] concluded from his study that if the function complexity is small compared to the sample size, the performance of cross-validation is relatively insensitive to the choice of the data splitting ratio. Any choice between 1 / 9 and 1 / 1 yields nearly optimal generalization error. However, as the function becomes more complex relative to the sample size, the optimal data splitting ratio decreases. For the backpropagation experiments with early stopping, Kearns showed that the optimal data splitting ratio is somewhere between 1 /4 and 3 / 7.

We believe that the optimal data splitting ratio is problem dependent. However, there could be a narrow range for the ratio that works nearly optimally for a wide range of applications. Since only part of the available data can be used for parameter estimation, the resulting networks trained with early stopping generalize worse, on average, for noisy data, than networks trained using all the data with properly adapted regularization. To overcome this drawback, we will propose a new procedure called retrained early stopping.

Retrained Early stopping

In order to improve the performance of networks trained with conventional early stopping, we propose a new procedure which retrains the network after early stopping. We call the training using early stopping the first stage training, and the retraining described in this section the second stage training. The objective of the retraining is to improve the parameter estimation by using the whole data set, without violating the model constraints determined by early stopping.

The new retraining procedure is described below. The retraining starts with the resulting weights obtained from the first stage training, but uses a combined training data set including all data points in the previous training and validation sets. Any standard optimization algorithm with unregularized performance function can be used in the retraining process. The stop criteria of the second stage training are based on the model accuracy and complexity of the first stage. We use the mean-squared-error (MSE) to measure the model accuracy. In addition, we use the sum-squared-weights (SSW) to measure the model complexity. The error goal of the second stage training is slightly less than the value of the MSE at the early stopping point. The weight goal of the second stage training is the value of the resulting SSW of the first stage multiplied by a small incremental constant ($1.05 \sim 1.3$). The retraining will be terminated when either the error goal or weight goal is met.

Now let us explain why it is possible to use retrained early stopping to improve generalization. As we discussed early in this chapter, terminating network training before convergence is similar to regularization. Regularization can use the whole data set for parameter estimation. However, early stopping can only use part of the available data points, because a separate data set has to be selected for validation testing. The selection of the validation data set is somewhat arbitrary. Therefore, it is rational to find a way to adjust the parameter estimation after early stopping by retraining the network using the combined data set. If the network complexity is fixed, training with more data points does not risk overfitting.

Here arises another question. In order to make a full use of the available data set, we would like to perform a combined training. However, since we use the unregularized performance function, the MSE will decrease, and the SSW will increase when retraining proceeds. To avoid overfitting, we add the error goal and the weight goal constraints obtained from the first stage training. The error goal gives a lower bound on the training accuracy, which is the best achieved using cross-validation. The weight goal gives an upper bound on the SSW, which limits the model complexity. These bounds set a new compromise between the model accuracy and the model complexity for the combined training. To justify these settings, let us make a new use of Moody's GPE (generalized prediction error) [Mood92] which was briefly discussed in Chapter 2.

Moody's model selection criterion can be expressed as

$$\mathrm{GPE} = \mathrm{MSE}_{\mathrm{train}} + 2\sigma^2\frac{\gamma}{n} , \qquad (93)$$

where γ is the effective number of parameters, n is the number of training data points, and σ^2 is the variance of the noise on the data. Eq. (93) is essentially a different expression of Eq. (83), which is given in the asymptotic learning section in this chapter and is rewritten below:

$$\langle R_{\text{gen}}(\hat{w})\rangle = \langle R_{\text{train}}(\hat{w})\rangle + \frac{N}{n}.$$

In Eq. (93), the training performance is measured by the mean-squared-error. In Eq. (83), the training performance is measured by the entropy. Also, Eq. (93) enables us to use a regularized performance function to calculate the effective number of parameters.

Since early stopping has an effect that is similar to regularization, we can use Eq. (93) to estimate the GPE obtained using early stopping. From [Mood92], σ^2 in Eq. (93) can be estimated by

$$\sigma^2 = \frac{n}{n-\gamma}\text{MSE}_{\text{train}}. \tag{94}$$

For this choice, the expression for GPE becomes

$$\text{GPE} = \frac{n+\gamma}{n-\gamma}\text{MSE}_{\text{train}} \quad . \tag{95}$$

Now assume that we use the same error goal for retrained early stopping. To make $\text{GPE}_2 < \text{GPE}_1$, where subscripts are the stage index numbers, the following inequality should be held

$$\frac{n_2+\gamma_2}{n_2-\gamma_2} < \frac{n_1+\gamma_1}{n_1-\gamma_1},$$

which indicates

$$\frac{\gamma_2}{\gamma_1} < \frac{n_2}{n_1}.$$

Recall that the $\gamma = 2\alpha\Omega(w)$ in Bayesian regularization, where α is the regularization parameter, and $\Omega(w)$ is the sum-squared-weights (SSW). And suppose that early stopping and retrained early stopping generate models with similar complexity, i.e., $\alpha_1 \approx \alpha_2$ in a rough analogy to regularization. The condition required to get a reduced GPE becomes

$$\frac{\mathrm{SSW}_2}{\mathrm{SSW}_1} < \frac{n_2}{n_1}.$$

Thus, if we limit the increase of the SSW to a small fixed percentage during the retraining, the inequality above will hold for most practical data splitting ratios, and the GPE may be reduced. Even though the SSW goal is reached before the error goal has been met, the inequality above may still compensate for the increase in the MSE and force the GPE to decrease.

Let us assume some retraining situations. When a small data splitting ratio is used, the starting MSE for the retraining is either less than or slightly larger than the error goal. In the former case, the prediction on the validation set was better than the prediction on the training set. Therefore, no further combined training is needed. In the latter case, only a few training epochs are required to reach the error goal. The increase in the SSW will not exceed the permissible range.

When a large data splitting ratio is used, the starting MSE for the retraining may be larger than the error goal by a significant percentage. Since retraining usually requires more iterations to reach the error goal under this circumstance, the weight goal might be met first. Whether or not the generalization error decreases will depend on the balance of the product on the right side of Eq. (95).

Note that calculating the equivalent effective number of parameters for early stopping is very difficult. In general, we cannot locate the lowest point of the GPE curve during the retraining. And, without referring to a new validation data set, we cannot determine the optimal stop point by using the combined training data set only. Therefore, the setting of the stop criteria for the retraining is heuristic. However, as we discussed above, in order to get improved performance, it would be better to perform the limited retraining around the previous stop point.

In the next section, we will compare retrained early stopping with conventional early stopping using a simple problem with different pattern parameter ratios and data splitting ratios. Simulations on a different problem under more complicated conditions will be discussed in Chapter 6.

Training Example

To investigate the features of retrained early stopping (RTES) and conventional early stopping (ES), let's consider the same shifted sinusoidal function that we used in Chapter 3,

$$y = 1 + \sin\left(\frac{\pi}{4}x\right).$$

This is a simple single-input, single-output example, and will be modeled by a 1-15-1 network with 46 parameters. For each training procedure, we run simulations with three pattern / parameter ratios (PPR = 81/46, 161/46 and 481/46), and three data splitting ratios (DSR= 1/7, 1/3 and 1/1). The training targets y_i are evenly generated from the shifted sinusoidal function over the inputs x_i $(-2 \sim 6)$, and corrupted with Gaussian noise (zero mean and 0.04 variance). For each PPR / DSR combination, the generalization error is measured between the network outputs and the true function outputs on a large testing data set (6000 data points), and is averaged over 100 trials with different initial weights and random noise realizations. The computational cost is measured by the average number of floating point operations (FLOPS).

The goals for retrained early stopping are set to $0.95 \times \text{MSE}$ and $1.30 \times \text{SSW}$, where MSE and SSW are the mean-squared-error and sum-squared-weights obtained using conventional early stopping. In addition, we require retraining to take at least one epoch. The results are summarized in Table 2.

PPR	DSR	$\overline{\text{MSE}}$ (ES) ($\times 10^{-3}$)	$\overline{\text{MSE}}$ (RTES) ($\times 10^{-3}$)	$\overline{\text{FLOPS}}$ (ES) ($\times 10^{7}$)	$\overline{\text{FLOPS}}$ (RTES) ($\times 10^{7}$)
81 / 46	1 / 7	6.80	5.66	1.99	2.41
	1 / 3	6.80	5.37	1.80	2.39
	1 / 1	7.57	4.84	1.48	2.15
161 / 46	1 / 7	4.09	3.46	3.54	4.72
	1 / 3	4.21	3.26	3.24	4.46
	1 / 1	4.29	2.80	2.50	3.80
481 / 46	1 / 7	1.72	1.26	11.8	15.6
	1 / 3	1.55	1.08	9.43	13.2
	1 / 1	1.63	1.00	6.75	10.7

Table 2 Comparison of ES and RTES

It is observed that prediction becomes more accurate as the pattern / parameter ratio increases. For each PPR / DSR combination, the performance trained with RTES is better than that trained with ES. However, the quantitative improvement varies with different data splittings. The error ratio of ES over RTES is $1.18 \sim 1.37$ for DSR = 1 / 7, $1.29 \sim 1.43$ for DSR = 1 / 3, and $1.53 \sim 1.63$ for DSR = 1 / 1. This is in good agreement with our previous analysis, which says that using a relatively large DSR will make the conditional inequality easily satisfied.

Table 2 also shows that the optimal data splitting ratios for retrained early stopping and conventional early stopping are different. The optimal DSR is 1 / 1 for RTES, but is 1 / 7 or 1 / 3 for ES. Therefore, it is better to use a relatively large DSR when retrained early stopping is used, but to use a relatively small DSR if the network is only trained with conventional early stopping.

In this simulation experiment, the computational cost of RTES is $21 \sim 56$ percent higher than ES. The larger percentage corresponds to the larger DSR, and the smaller percentage corresponds to the smaller DSR. However, since training with ES using a larger DSR is less computationally expensive, the lowest cost for RTES is when the DSR of 1 / 1 is used. If we compare the best ES performance with the best RTES performance under each PPR, the generalization error for ES is $40 \sim 55$ percent larger, and the computational cost for RTES is only $7 \sim 19$ percent higher.

In summary, this example demonstrates the potential to retrain a network using the combined data set after early stopping. In order to improve model performance, we can use a relatively large validation data set to determine reliable estimators of the optimal model accuracy and complexity. We then can perform a limited full-data retraining to obtain better parameter estimation. The simulation shows that the networks produced by retrained early stopping generalize significantly better than conventional early stopping, with only slightly increased computational cost.

Here arises an interesting question. As we discussed in this chapter, the validation data set is used to monitor the network training. However, the training follows its own path before early stopping. The minimum validation error at the stop point is a function of the initial training parameters, and cannot be adapted during the training. This is called passive validation. This problem is inherent in using the unregularized training performance function. In the next chapter, we will combine regularization with the active use of the validation data set, and we will propose a new adaptive regularization algorithm to minimize validation error.

CHAPTER 5

ACTIVE VALIDATION AND THE SDVR ALGORITHM

Objectives

This chapter introduces the concept of validation-set-based regularization and proposes a new SDVR framework, which consists of a basic algorithm and two variations. The development of the SDVR framework is one of the main contributions of this study.

Introduction

In previous chapters, we have shown how early stopping and regularization techniques can be applied to improve generalization in neural network modeling. In most applications, early stopping and regularization are used separately. When the early stopping procedure is employed, we terminate the training process as soon as the validation error increases. In that approach, the validation data set is passively withheld to determine the training stop point. When Bayesian regularization is used, we do not use a separate validation data set. The regularization parameter is statistically updated to match the training data.

In this chapter we will combine regularization with the active use of the validation data set. Instead of using the validation set to determine when to stop training, we will choose the regularization parameter α so as to minimize the error on the validation set.

The main purpose of this chapter is to develop a new algorithm to iteratively update the regularization parameter through an adaptive training process. We will propose a second derivative of validation error based regularization method (SDVR), and investigate its application in depth.

We will begin with a brief review of validation set based regularization, and introduce a convergent gradient descent approach which was proposed by Larsen et al. [LaHa96].

We then introduce our SDVR algorithm. We will start with the mathematical derivation of the basic algorithm, which recalculates the regularization parameter in each training epoch. Then, we will apply the basic SDVR algorithm to a training example and will show how the SDVR algorithm improves the model generalization performance. The specific parameter settings in the application recipe will be explained briefly.

Additional features of the SDVR algorithm will be investigated through a comparison test with the gradient descent method. The simulations will be performed under a variety of initial conditions. The model performance and the computation cost will be evaluated accordingly.

The basic SDVR algorithm updates the regularization parameter incrementally. However, this is not the only possible implementation. We will propose two variations of the SDVR algorithm that enable the regularization parameter to be updated over a variable interval according to the specified control criteria. All of implementations presented in this chapter can be placed into a common SDVR framework. We will test this framework with several numerical experiments and show how to make optimal use of the SDVR framework with a variety of different problems.

Background Review

Many applications have shown that training a feedforward neural network with the regularized performance function $F = e^T e + \alpha w^T w$ can improve the generalization performance of the network, if the regularization parameter α is appropriately estimated. However, how to determine the parameter α is still an open question. There are several different approaches to this problem. MacKay's Bayesian framework automatically adapts the regularization parameter to maximize the evidence of the training data [MacK92]. The computation overhead in updating the regularization parameter can be reduced when the Gauss-Newton approximation to the Hessian matrix is employed [FoHa97].

A different approach to adaptive regularization is to minimize validation error. In this case, a validation data set, which is independent of the training data, is withheld for decision making. The network weights are estimated from the training data, and the amount of regularization is optimized for the validation data. This approach is based on the assumption that the selected validation set is a good representative of new data. Therefore, the model with minimum validation error will have a better chance to generalize well on novel inputs. The simplest application of this method is to train the neural network with a number of different α values, and then choose the model having the smallest validation error.

A more attractive approach to validation-set-based regularization is to use an optimization algorithm to adapt the regularization parameter α automatically. Consider that validation error is a function of the network weights, and the network weights are affected by the α value through the regularized performance function. Therefore, the validation error is an implicit function of α. These inherent relations can be used to solve the optimization problem. A simple gradient descent algorithm was proposed by Larsen et al. [LaHa96] using a single validation set, and then extended to multi-fold validation sets [LaSv97]. In both approaches, an updated regularization parameter is calculated after the network has been trained to convergence with the previous regularization parameter. After each parameter update, the network is again trained to convergence.

Theoretically speaking, validation-set-based regularization is not constrained to a single validation set and a single regularization parameter. Multi-fold validation sets and multiple regularization parameters can be used under the same framework. However, some problems are associated with these options ([Gout97], [ZhRo96]). If the regularizer consists of many parameters, there is a potential risk of overfitting on the cross-validation data. Also, the computation burden and complexity of the data splitting may prohibit their practical use. In the rest of the chapter, we will use a single validation set and a single regularization parameter.

Gradient Descent Approach

We have mentioned that Larsen's gradient descent approach to adaptive regularization is implemented with a convergent updating method; the regularization parameter is only updated after the network has been trained to convergence. Larsen's approach is commonly used for validation-based regularization, so we will introduce this scheme first, in order to provide a basis for comparison for the new methods introduced in this chapter.

Let's start with a preselected regularization parameter α_k and assume that a network model has been trained with fixed α_k to convergence. The resulting weight vector w is a function of α_k, which can be denoted as $w(\alpha_k)$. The validation performance F_v is measured using w,

$$F_v(w) = E_v^D = (e_v(w))^T e_v(w) . \tag{96}$$

Calculating the new regularization parameter along the gradient descent direction $\nabla_\alpha F_v(w(\alpha_k))$ leads to an updating equation

$$\alpha_{k+1} = \alpha_k - \eta \nabla_\alpha F_v(w(\alpha_k)) \ , \tag{97}$$

where η is a leaning rate, and

$$\nabla_\alpha F_v(w(\alpha_k)) = \frac{\partial}{\partial \alpha_k} F_v(w) . \tag{98}$$

In order to find the gradient $\nabla F_v(w(\alpha_k))$, we use the chain rule

$$\frac{\partial}{\partial \alpha_k} F_v(w) = \frac{\partial}{\partial \alpha_k}(w)^T \cdot \frac{\partial}{\partial w} F_v(w), \qquad (99)$$

since $F_v(w)$ is an implicit function of α_k.

Now, let's consider the first term on the right side of Eq. (99). To calculate the partial derivative of the weight vector with respect to the fixed α_k, we need to use some properties of the training performance function. Recall that when the weight vector is trained to convergence, the training performance reaches its local minimum, i.e.,

$$\frac{\partial}{\partial w} F_t(w) = 0, \qquad (100)$$

where the regularized training performance index is a function of w and α_k:

$$F_t(w) = E_t^D + \alpha_k \Omega(w) = (e_t(w))^T e_t(w) + \alpha_k w^T w. \qquad (101)$$

Eq. (100) implies

$$\frac{\partial^2}{\partial w \partial \alpha_k} F_t(w) = 0. \qquad (102)$$

We know that the first derivative of the training performance index is also a function of w and α_k,

$$\frac{\partial}{\partial w} F_t(w) = \nabla F_t(w) = 2 J^T(w) e(w) + 2\alpha_k w, \qquad (103)$$

Differentiating Eq. (103) with respect to α_k can be expanded as:

$$\frac{\partial}{\partial \alpha_k} \nabla F_t(w) = \frac{\partial}{\partial w} \nabla F_t(w) \cdot \frac{\partial w}{\partial \alpha_k} + \frac{\partial}{\partial \alpha_k} \nabla F_t(w) \cdot \frac{\partial \alpha_k}{\partial \alpha_k} \qquad (104)$$

where the second derivative of the training performance function with respect to the weight vector is the Hessian matrix,

$$\frac{\partial}{\partial \boldsymbol{w}} \nabla F_t(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w}), \tag{105}$$

and from Eq. (103),

$$\frac{\partial}{\partial \alpha_k} \nabla F_t(\boldsymbol{w}) = 2\boldsymbol{w}. \tag{106}$$

Substituting Eq. (105) and Eq. (106) into Eq. (104), and setting it to zero, we have

$$\boldsymbol{H}(\boldsymbol{w}) \cdot \frac{\partial \boldsymbol{w}}{\partial \alpha_k} + 2\boldsymbol{w} = 0. \tag{107}$$

Then we obtain the derivative of the resulting $\boldsymbol{w}$ with respect to α_k,

$$\frac{\partial \boldsymbol{w}}{\partial \alpha_k} = -2\boldsymbol{H}(\boldsymbol{w})^{-1}\boldsymbol{w}. \tag{108}$$

The second term on the right side of Eq. (99) is the gradient of the validation performance function with respect to the resulting $\boldsymbol{w}$, which can be calculated from Eq. (96),

$$\frac{\partial}{\partial \boldsymbol{w}} F_v(\boldsymbol{w}) = 2\boldsymbol{J}_v^T(\boldsymbol{w})\boldsymbol{e}_v(\boldsymbol{w}). \tag{109}$$

Substituting Eq. (108) and Eq. (109) into Eq. (99), we obtain the required gradient

$$\frac{\partial}{\partial \alpha_k} F_v(\boldsymbol{w}) = -4[\boldsymbol{H}(\boldsymbol{w})^{-1}\boldsymbol{w}]^T \boldsymbol{J}_v^T(\boldsymbol{w})\boldsymbol{e}_v(\boldsymbol{w}). \tag{110}$$

The regularization parameter α_{k+1} then can be recalculated using Eq. (97).

In the next iteration, the network is again trained to convergence with fixed α_{k+1}. After each updating of the regularization parameter, the training is initialized with the previous resulting weight vector. If the validation error increases with current α, then, the learning rate is reduced by bisection, and the regularization parameter is recalculated from the previous α. This retraining may be attempted several times in Larsen's approach until the validation error decreases. Then the next estimate of α is computed from the current value. The first stage training is usually terminated if the gradient of the validation error with respect to the regularization parameter is small enough, or the relative change of validation errors between two adjacent training cycles is below a predetermined threshold. Finally, the parameter α corresponding to the minimum validation error is used to perform the second stage training using all available data.

Limitations

The gradient descent approach is simple in implementation. However, it has several inherent limitations.

Convergent updating using the gradient descent method is computationally inefficient. If the training starts from a small α, and the initial learning rate is also small, more than a hundred retrainings are usually needed to find the appropriate value of α. Reaching convergence for each regularization parameter, for most α values, would require many training epochs. The total computation cost would be very high.

Another problem with the gradient descent scheme is the determination of the learning rate. Usually, a small learning rate is used for α to ensure stability. As a consequence, the gradient descent approach has a tendency to converge to the first local minimum point. For example, when α increases gradually, it often converges to the first local minimum at a small α; and it never has a chance to reach the global minimum point with a larger α. On the other hand, since the increment in α is the product of the learning rate and the gradient, an unexpectedly large jump in α may occur even with a small learning rate if gradient is large. As we will see later in this chapter, the proper choice of the learning rate for gradient descent depends on the initial weights and the initial regularization parameter, and the variation in training results is usually large when different initial conditions are tested with same learning rate. This shortcoming makes gradient descent difficult to apply.

In addition, training to convergence is controversial in practice, since there are many possible convergence criteria. If we use the Levenberg-Marquardt training algorithm, as we discussed in previous chapters, convergence is defined when the training performance function does not decrease with a large value of μ_w. However, the regularization parameter will almost remain unchanged if the maximum μ_w is reached, because the gradient calculated from the Larsen's equation is too small. In general, we have to update the regularization parameter before the μ_w value becomes very large, when the Levenberg-Marquardt algorithm is used.

In the next section, we will propose a more efficient and easy-to-apply training algorithm using validation-set-based regularization.

Basic SDVR Algorithm

In this section we will introduce a new algorithm for active regularization that addresses some of the limitations of Larsen's approach. There are two main innovations of this algorithm. First, the algorithm uses second order information to set the optimal learning rate. Second, the algorithm updates the regularization parameter at each weight update, rather than after the weights have converged. Tests will show that the new algorithm produces networks with better generalization and requires less computation.

Before introducing our proposed new algorithm, let's assume that the network is trained with a batch training algorithm. Its regularized performance index at iteration k is

$$F_t(\mathbf{w}_k) = (\mathbf{e}_t(\mathbf{w}_k))^T\mathbf{e}_t(\mathbf{w}_k) + \alpha_k \mathbf{w}_k^T \mathbf{w}_k \tag{111}$$

where $\mathbf{w}_k$ is an $N \times 1$ weight vector, $\mathbf{e}_t(\mathbf{w}_k)$ is an $n \times 1$ error vector and the subscript t denotes the training data set. For the next training epoch, the weight vector $\mathbf{w}_{k+1}$ is computed with fixed α_k to minimize the performance index by using the Gauss-Newton method

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{H}^{-1}(\mathbf{w}_k)\nabla F_t(\mathbf{w}_k) \tag{112}$$

where $\nabla F_t(\mathbf{w}_k)$ is the gradient vector of $F_t(\mathbf{w}_k)$ with respect to $\mathbf{w}_k$,

$$\nabla F_t(\mathbf{w}_k) = 2\mathbf{J}^T(\mathbf{w}_k)\mathbf{e}(\mathbf{w}_k) + 2\alpha_k \mathbf{w}_k . \tag{113}$$

$\mathbf{H}(\mathbf{w}_k)$ is the Gauss-Newton approximation to the Hessian matrix,

$$\nabla^2 F_t(\mathbf{w}_k) = \mathbf{H}(\mathbf{w}_k) \approx 2\mathbf{J}^T(\mathbf{w}_k)\mathbf{J}(\mathbf{w}_k) + 2\mathbf{I}_N(\alpha_k + \mu_w) , \tag{114}$$

in which $\mathbf{J}(\mathbf{w}_k)$ is an $n \times N$ Jacobian matrix, $\mathbf{I}_N$ is an $N \times N$ identity matrix, and μ_w is a tunable parameter, as in the Levenberg-Marquardt implementation [HaDe96], [Fore96].

Now we propose a new algorithm to update parameter α using a second derivative of validation error based regularization (SDVR). The updating equation for the parameter

α uses an approximation of Newton's method to minimize the validation error:

$$\alpha_{k+1} = \alpha_k - H_\alpha^{-1}(w_{k+1}(\alpha_k))\nabla_\alpha F_v(w_{k+1}(\alpha_k)) \quad . \tag{115}$$

In Eq. (115), the validation error

$$F_v(w_{k+1}(\alpha_k)) = (e_v(w_{k+1}))^T e_v(w_{k+1}) \tag{116}$$

is a function of w_{k+1}, hence it is an implicit function of α_k. $\nabla_\alpha F_v(w_{k+1}(\alpha_k))$ and $H_\alpha(w_{k+1}(\alpha_k))$ are the gradient and Hessian of the validation error with respect to the regularization parameter α_k. Thus

$$\nabla_\alpha F_v(w_{k+1}(\alpha_k)) = \frac{\partial}{\partial \alpha_k} F_v(w_{k+1}(\alpha_k)), \tag{117}$$

and

$$H_\alpha(w_{k+1}(\alpha_k)) = \frac{\partial}{\partial \alpha_k} \nabla_\alpha F_v(w_{k+1}(\alpha_k)). \tag{118}$$

In order to update the parameter α_{k+1}, we assume that the weight vector w_{k+1} is updated first, using a fixed α_k. After w_{k+1} is computed, then α_{k+1} is updated. In the following derivation, we will refer to updating w_{k+1} as the inside loop training, and will refer to calculating α_{k+1} as the outside loop updating.

Incremental Gradient Descent

As we see in Eq. (115), the mathematical implementation of the SDVR algorithm requires the gradient $\nabla_\alpha F_v(w_{k+1}(\alpha_k))$ and the Hessian $H_\alpha(w_{k+1}(\alpha_k))$. Since validation error is not an explicit function of α_k, we can use chain rule:

$$\frac{\partial}{\partial \alpha_k} F_v(w_{k+1}(\alpha_k)) = \frac{\partial}{\partial \alpha_k}(w_{k+1})^T \cdot \frac{\partial}{\partial w_{k+1}} F_v(w_{k+1}(\alpha_k)). \tag{119}$$

Note that Eq. (119) is different from Eq. (99). In Eq. (99), the chain rule was applied to find the convergent gradient. In Eq. (119), the chain rule is used to compute the incremental gradient. The first partial derivative term on the right side of Eq. (119) can be calculated from Eq. (112). Recall that w_k was computed before α_k was updated. From this view, w_k is not a function of α_k. Therefore differentiating Eq. (112) with respect to α_k becomes

$$\frac{\partial}{\partial \alpha_k}(w_{k+1}) = -\left[\left[\frac{\partial}{\partial \alpha_k} H^{-1}(w_k)\right] \nabla F_t(w_k) + H^{-1}(w_k) \frac{\partial}{\partial \alpha_k} \nabla F_t(w_k)\right]. \tag{120}$$

To find the derivative matrix $\partial H^{-1}(w_k)/\partial \alpha_k$, we use the fact that

$$H(w_k) H^{-1}(w_k) = I_N. \tag{121}$$

Since the derivative of the identity matrix I_N with respect to α_k is zero, we have

$$\left[\frac{\partial}{\partial \alpha_k} H(w_k)\right] H^{-1}(w_k) + H(w_k) \frac{\partial}{\partial \alpha_k} H^{-1}(w_k) = 0 . \tag{122}$$

Now let's examine derivative $(\partial H(w_k))/(\partial \alpha_k)$. In Eq. (114), the Jacobian matrix $J(w_k)$ is a function of w_k only. Since w_k is not a function of α_k,

$$\frac{\partial}{\partial \alpha_k} H(w_k) = 2 I_N. \tag{123}$$

Then $\partial H^{-1}(w_k)/\partial \alpha_k$ in Eq. (122) can be obtained by

$$\frac{\partial}{\partial \alpha_k} H^{-1}(w_k) = -H^{-1}(w_k)\left[\frac{\partial}{\partial \alpha_k} H(w_k)\right] H^{-1}(w_k) = -2[H^{-1}(w_k)]^2. \tag{124}$$

Similarly, $\partial \nabla F_t(w_k)/\partial \alpha_k$ in Eq. (120) can be calculated from Eq. (113) directly, with the result

$$\frac{\partial}{\partial \alpha_k} \nabla F_t(w_k) = 2 w_k . \qquad (125)$$

Substituting Eq. (124) and Eq. (125) into Eq. (120), we get

$$\frac{\partial}{\partial \alpha_k}(w_{k+1}) = -2 H^{-1}(w_k)[-H^{-1}(w_k) \nabla F_t(w_k) + w_k] . \qquad (126)$$

Note that the term in the bracket above is the new updated weight vector w_{k+1} from Eq. (112), therefore, we can rewrite Eq. (126) as:

$$\frac{\partial}{\partial \alpha_k}(w_{k+1}) = -2 H^{-1}(w_k) w_{k+1} . \qquad (127)$$

Now, let's go back to Eq. (119). After w_{k+1} is obtained from the inside loop training, the validation error is evaluated. The gradient of the validation error with respect to w_{k+1} can be calculated directly:

$$\frac{\partial}{\partial w_{k+1}} F_v(w_{k+1}(\alpha_k)) = 2 J_v^T(w_{k+1}) e_v(w_{k+1}) , \qquad (128)$$

where $J_v(w_{k+1})$ is the Jacobian matrix of the validation data. The incremental gradient now can be obtained by substituting Eq. (127) and Eq. (128) into Eq. (119):

$$\frac{\partial}{\partial \alpha_k} F_v(w_{k+1}(\alpha_k)) = -4 [H^{-1}(w_k) w_{k+1}]^T J_v^T(w_{k+1}) e_v(w_{k+1}) . \qquad (129)$$

Comparing Eq. (129) with Larsen's gradient expression in Eq. (110), we can see that Larsen's expression is a special case of Eq. (129) when the weight vector converges, at which time $w_{k+1} = w_k = w$. However, our approach and Larsen's approach are based on different assumptions. In Eq. (110), the converged weight vector w is a function of fixed α_k, the derivative $\partial w/\partial\alpha_k$ is derived using the convergent condition of the inside training loop, i.e., $\partial F_t/\partial w = 0$ and $\partial(\partial F_t/\partial w)/\partial\alpha_k = 0$. While in our approach, the regularization parameter is updated in each training epoch. Only the weight vector w_{k+1}, rather than w_k, is a function of α_k. The derivative $\partial w_{k+1}/\partial\alpha_k$ is computed from the weight update equation directly. Treating w_k and α_k as independent variables is important in implementing the SDVR algorithm. As we will see next, this assumption also makes it convenient to calculate the incremental Hessian.

Incremental Hessian Approximation

The SDVR algorithm is characterized by the incremental Hessian $H_\alpha(w_{k+1}(\alpha_k))$, which can be computed by differentiating Eq. (129) with respect to α_k.

$$\begin{aligned} H_\alpha(w_{k+1}(\alpha_k)) &= \frac{\partial}{\partial\alpha_k}\nabla_\alpha F_v(w_{k+1}(\alpha_k)) \\ &= -4\frac{\partial}{\partial\alpha_k}([\boldsymbol{H}^{-1}(w_k)w_{k+1}]^T\boldsymbol{J}_v^T(w_{k+1})\boldsymbol{e}_v(w_{k+1})) \end{aligned} \tag{130}$$

Since $[\boldsymbol{H}^{-1}(w_k)w_{k+1}]^T\boldsymbol{J}_v^T(w_{k+1})\boldsymbol{e}_v(w_{k+1})$ is a scalar, it is equal to its transpose. Therefore,

$$[\boldsymbol{H}^{-1}(w_k)w_{k+1}]^T\boldsymbol{J}_v^T(w_{k+1})\boldsymbol{e}_v(w_{k+1}) = \boldsymbol{e}_v^T(w_{k+1})\boldsymbol{J}_v(w_{k+1})[\boldsymbol{H}^{-1}(w_k)w_{k+1}]. \tag{131}$$

Using the rule for differentiation of a product, we can rewrite Eq. (130) as:

$$\frac{\partial}{\partial \alpha_k}[-4(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T \boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})] \qquad . \tag{132}$$
$$= -4\left[\frac{\partial}{\partial \alpha_k}[(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T]\right]\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})$$
$$-4\left[\frac{\partial}{\partial \alpha_k}[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})]\right]\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1}$$

The first partial derivative of the product within the bracket on the right side of Eq. (132) can be expanded as

$$\frac{\partial}{\partial \alpha_k}[(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T] = \left[\left[\frac{\partial}{\partial \alpha_k}\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\right]\boldsymbol{w}_{k+1} + \boldsymbol{H}^{-1}(\boldsymbol{w}_k)\frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1})\right]^T . \tag{133}$$

Note that both $\partial \boldsymbol{H}^{-1}(\boldsymbol{w}_k)/\partial \alpha_k$ and $\partial(\boldsymbol{w}_{k+1})/\partial \alpha_k$ in Eq. (133) were calculated previously in deriving the incremental gradient. Substituting Eq. (124) and Eq. (127) into Eq. (133), we get

$$\frac{\partial}{\partial \alpha_k}[(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T] = -4[[\boldsymbol{H}^{-1}(\boldsymbol{w}_k)]^2\boldsymbol{w}_{k+1}]^T \tag{134}$$

The derivative vector $\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})$ in Eq. (132) is not an explicit function of α_k. Its derivative with respect to α_k can also be calculated using the chain rule:

$$\frac{\partial}{\partial \alpha_k}[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})] = \frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1})^T \cdot \frac{\partial}{\partial \boldsymbol{w}_{k+1}}[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})] , \tag{135}$$

In Eq. (135), only $\partial[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})]/\partial \boldsymbol{w}_{k+1}$ is unknown, which is the Hessian matrix of the validation data. As with the Hessian matrix of the training data, we use the Gauss-Newton method to obtain the approximate expression

$$\frac{\partial}{\partial \boldsymbol{w}_{k+1}}(\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})) \cong \boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1}) , \tag{136}$$

Substituting Eq. (127) and Eq. (136) into Eq. (135), we have

$$\frac{\partial}{\partial \alpha_k}[e_v^T(w_{k+1})J_v(w_{k+1})] = -2[H^{-1}(w_k)w_{k+1}]^T J_v^T(w_{k+1})J_v(w_{k+1}). \tag{137}$$

The approximate incremental Hessian can be obtained by putting Eq. (134) and Eq. (137) into Eq. (132):

$$\begin{aligned} H_\alpha(w_{k+1}(\alpha_k)) \equiv\; & 16(H^{-1}(w_k)H^{-1}(w_k)w_{k+1})^T J_v^T(w_{k+1})e_v(w_{k+1}) \\ & + 8(H^{-1}(w_k)w_{k+1})^T J_v^T(w_{k+1})J_v(w_{k+1})(H^{-1}(w_k)w_{k+1}) \end{aligned} . \tag{138}$$

As we see from Eq. (138), the incremental Hessian is eventually approximated as an algebraic manipulation of the characteristic matrices and vectors of the training data set and the validation data set. It appears to involve significant computation. However, after the incremental gradient is calculated, the extra computation associated with the incremental Hessian is very limited. In addition, since we use the Gauss-Newton approximation to both the Hessian matrix of the training data and the Hessian matrix of the validation data, calculating the incremental Hessian is convenient. For instance, $H^{-1}(w_k)$ and w_{k+1} in Eq. (138) can be obtained from the inside loop training directly. The validation gradient vector $J_v^T(w_{k+1})e_v(w_{k+1})$ and the validation Hessian matrix $J_v^T(w_{k+1})J_v(w_{k+1})$ can be calculated simply by passing the validation data through the same conventional subroutine as used in computing the gradient and Hessian matrix for the training data.

One more comment should be made on Eq. (138). Recall that in the inside loop training, the Hessian matrix given in Eq. (114) can be made positive definite with the Levenberg-Marquardt implementation, which guarantees that the weight increment is always in a descent direction. However, the situation is different as we compute the incremental

Hessian in the outside loop. Let's look at the two sequential product terms in Eq. (138). The second sequential product term $(H^{-1}(w_k)w_{k+1})^T J_v^T(w_{k+1})J_v(w_{k+1})(H^{-1}(w_k)w_{k+1})$ has a quadratic form with $J_v^T(w_{k+1})J_v(w_{k+1})$ being a real symmetric and positive semi-definite matrix, but the first product term $(H^{-1}(w_k)H^{-1}(w_k)w_{k+1})^T J_v^T(w_{k+1})e_v(w_{k+1})$ is non-quadratic which can be either positive or negative. Therefore, by using Eq. (138) directly, it is possible to get a negative incremental Hessian $H_\alpha(w_{k+1}(\alpha_k))$, which is not desirable in practice. In order to keep the increment of the regularization parameter in a descent direction, we will force $H_\alpha(w_{k+1}(\alpha_k))$ to be equal to the quadratic term when the value obtained from Eq. (138) is negative. In this case, we get a larger Hessian, which corresponds to a reduced learning rate and will not cause any problem during the training.

In addition, a tunable positive parameter μ_α can be added to Eq. (138) to make the incremental Hessian invertible in any case and can be used to adjust the effective learning rate. This is similar to the use of μ_w in Eq. (114) with the Levenberg-Marquardt implementation. A small μ_α corresponds to a second derivative dominated approach, while a large μ_α indicates a transition to the gradient descent method. Another purpose of μ_α is to provide additional stability control. When we use a regularized performance function to train a neural network model, the α value is usually restricted to be positive. However, a decrement of the regularization parameter suggested by the updating equation may lead to a negative α, which may cause a large increase in weight values. To avoid this problem, we can force the updated α to be positive by increasing μ_α if the decrement of the regularization parameter is too large. Other usages of μ_α will be discussed in the next section. Adding μ_α into Eq. (138), we have

$$\begin{aligned} &H_\alpha(w_{k+1}(\alpha_k)) \\ &\cong 16(H^{-1}(w_k)H^{-1}(w_k)w_{k+1})^T J_v^T(w_{k+1})e_v(w_{k+1}) \\ &+ 4(H^{-1}(w_k)w_{k+1})^T H_v(w_{k+1})(H^{-1}(w_k)w_{k+1}) + \mu_\alpha \end{aligned} \quad , \qquad (139)$$

with $H_v(w_{k+1}) = 2J_v^T(w_{k+1})J_v(w_{k+1})$.

Now we have completed the derivation of both the gradient and the Hessian in Eq. (115) for the incremental updating of the SDVR algorithm. In the next section, we will apply the basic SDVR algorithm to a training example, and explain the relevant parameter settings.

Method of Application

In this section, we will use a two-stage training method to apply the SDVR algorithm. In the first stage, our purpose is to determine an optimal regularization parameter. In the second stage, we will use a fixed α and perform the final training on a combined data set consisting of the previous training and validation data. Since the resulting α is optimal, we will limit the complexity of the neural network so that it has less risk of overfitting. Our discussion will be mainly concentrated on the first stage training. When the optimal regularization parameter is determined, the final combined training is straightforward.

Recipe of Application

Here are the general steps required for optimization of the regularization parameter with the incremental SDVR algorithm:

1. Divide the available data set into training and validation subsets using a proper splitting ratio. Initialize w_0 and α_0.
2. Use the weight update equation (Eq. (112)) to get w_{k+1}.
3. Compute the validation error.
4. Check the relative change of the validation error. If the change is small enough, terminate the first stage of training, and go to step 5. Otherwise, use the α update equation (Eq. (115)) to obtain α_{k+1} and go back to step 2.
5. Put the training data set and the validation data set together for the final training, using the latest updated regularization parameter.

Training Example

For a demonstration of how the SDVR algorithm improves neural network model generalization, consider the parabolic function used in the previous chapter. The function output is computed by using the following equation:

$$Z = 2.5 + 5.0X(2-X)Y(2-Y),$$

where the variables X and Y both range from 0 to 2. We chose to explore the regularization with a two-input, single-output function, because in real world applications many regression problems use multi-input, single-output models. The function targets are corrupted with normally distributed noise with zero mean and 0.04 variance. A 2-10-10-1 feedforward neural network model with hyperbolic tangent activations on the hidden layers and linear activations on the output layer is used to learn the function. The pure data and the noisy data are displayed in Figure 13.

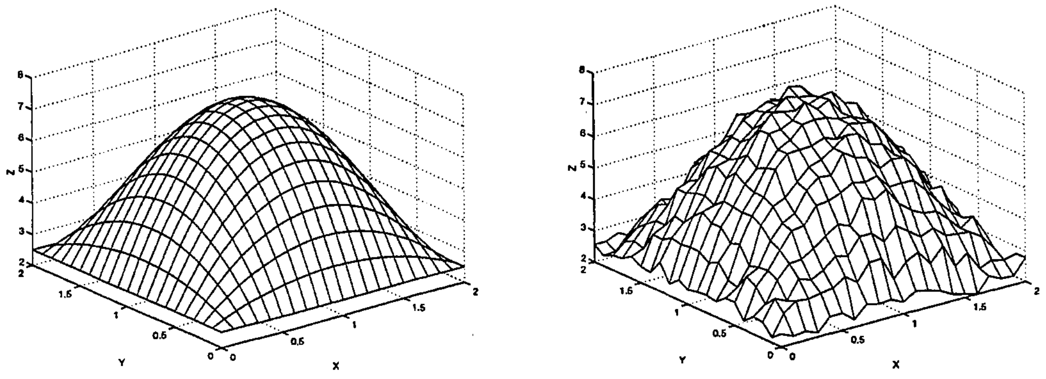


Figure 13 Pure and Noisy Data

In this example, the X-Y plane is sampled along a 21 by 21 grid. We use 231 data points for training and 110 data points for validation. The data splitting ratio is 110/231. The data splitting grid is given in Figure 14, where the circle-mark is used to represent the training sample and the x-mark is used to indicate the validation sample.

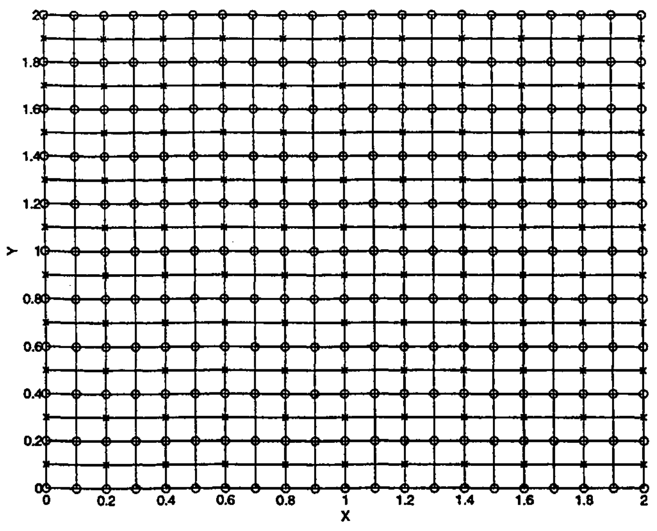
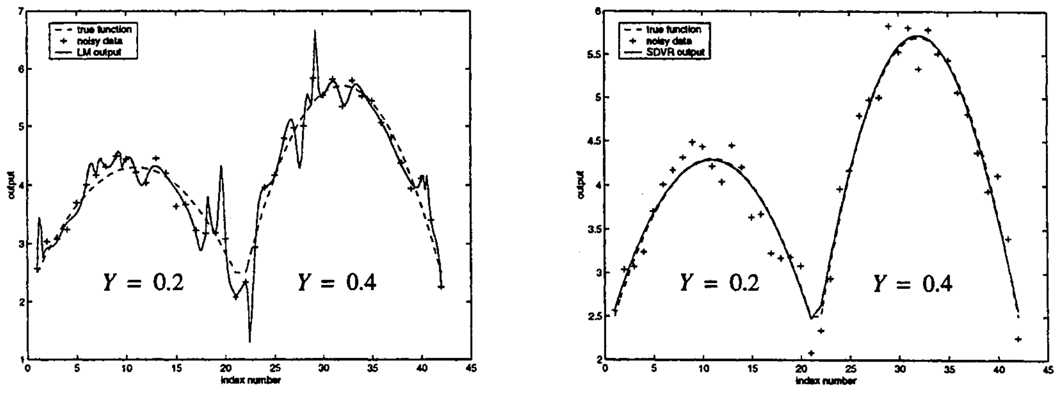


Figure 14 Data Splitting Grid

Note that training this complicated network (151 weight parameters in total) will overfit the noise training data if the unregularized performance function is used with the Levenberg-Marquardt algorithm. This is demonstrated in Figure 15 (a), where we show two cross-sections of the fitted function, at $Y = 0.2$ and $Y = 0.4$. Obviously, the model performance is poor if we do not use regularization, since the model fits the noise. As over-fitting occurs, some prediction errors for inputs between the training data points are very large.

In comparison, results obtained using the SDVR algorithm are displayed in Figure 15 (b). No overfitting occurs because the appropriate regularization is determined during the adaptive training. Therefore, with the use of the SDVR algorithm, we can choose a network with adequate complexity and apply a fast training algorithm with less worry of over-fitting.



(a). LM results (b). SDVR results

Figure 15 Model Comparison

A more complete evaluation of the SDVR results can be made by testing the model over the whole input data space, as shown in Figure 16. Recall that the training and validation data are selected from a 21 by 21 sampling grid on the X-Y plane, but now we use a 51 by 51 entry (2601 points) for testing. More than 2000 new points are contained in this test set. As we see from Figure 16, the learned function is much smoother than the noisy training data, and closely approximates the noise-free data shown in Figure 13.

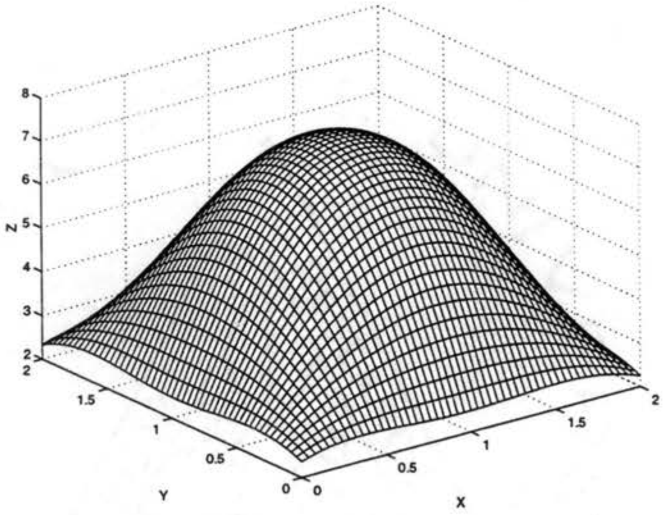


Figure 16 Model Generalization

Parameter Setting

Now, let's go back to the application recipe for the SDVR algorithm, and look at specific parameter settings at each training step.

Data Splitting Ratio

The data splitting ratio in step 1 is the number of the validation data points divided by the number of training data points. Since the SDVR algorithm is derived under the assumption that the validation data set is a good representation of new data, the validation data set should have a wide coverage. For this reason, a splitting ratio from 0.5 to 1 is preferred. However, when the target function is complex compared to the sample size, and the data is noiseless, a smaller data splitting ratio can also be used. In practice, the SDVR algorithm is best suitable to the situations where the data samples are adequate to be divided into two groups without losing generality. The effect of the data splitting ratio on the generalization performance will be discussed in the next chapter.

Initial Conditions

The effect of the initial conditions on the SDVR training results will be discussed later in this chapter. In the above example, the initial weights are normally distributed with zero mean and 0.01 variance. The initial regularization parameter is set to 0.01 which is far from the optimal value obtained by the end of the first stage training.

Tunable Parameter μ_w

In step 2, Hessian matrix $\boldsymbol{H}(\boldsymbol{w}_k)$ must be computed in the inside loop to obtain the network weight vector $\boldsymbol{w}_{k+1}$. Using the Levenberg-Marquardt implementation of Eq. (114), we usually set a small initial μ_w (e.g., 0.005), which corresponds to a fast learning rate. For the fixed regularized performance function characterized by $\boldsymbol{w}_k$ and α_k, the goal of the inside loop optimization is to search for a new weight vector $\boldsymbol{w}_{k+1}$ to reduce the performance index on the training data. This can be done by iteratively adjusting μ_w until the right $\boldsymbol{w}_{k+1}$ is found. If the performance is not improved, μ_w is incremented. After the performance is improved, μ_w is decremented. Normally, the increment constant of μ_w can be set to 10, and the decrement constant can be set to 0.1. Performing inside loop training in this way, we can guarantee that improved training performance is obtained before the regularization parameter is updated.

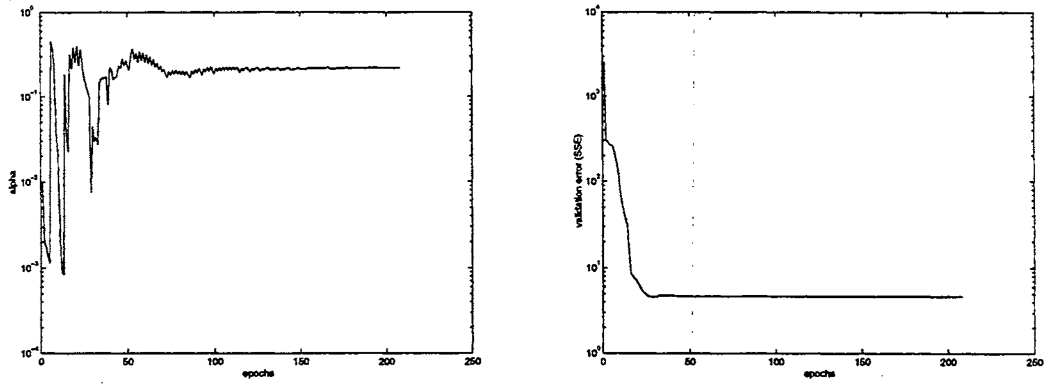
Tunable Parameter μ_α

Updating the regularization parameter in step 4 requires the computation of the parameter increment using Eq. (115). Similar to the weight calculation within the inside loop, we can preselect a small μ_α in Eq. (139) to get the new estimate of α_{k+1}. The adjustment of μ_α is dependent on the next validation error, which is measured by using the resulting model from the inside loop training. Note that the dynamic weight update within the inside loop is based on the training data set only, and the effect of the regularization parameter on the validation error is indirect. Therefore, the trajectory of the validation error may not be monotonicaly descending during the training. When the validation error increases with the α_{k+1}, instead of going back to the previous α_k and giving a different increment, we still accept the current update but compensate in the next iteration by multiplying μ_α by an increment coefficient. This increment coefficient can be set to 1.05 for many applications, and no decrement coefficient is required in general. With this implementation, the parameter μ_α will not contribute significantly in computing the incremental Hessian $H_\alpha(w_{k+1}(\alpha_k))$ if the validation error keeps going down, but gradually affects the learning rate if the validation error oscillates. This is the advantage of the SDVR algorithm, which allows a transition from the second derivative dominated approach to the gradient descent method as the value of μ_α becomes large.

Stop Criteria

The goal of the first stage training using the SDVR algorithm is to determine an optimal regularization parameter. Therefore, it is reasonable to terminate the first stage training in step 4 when the regularization parameter converges. This corresponds to the condition $\partial F_v/\partial\alpha \cong 0$, which indicates that the gradient of the validation error with respect to the regularization parameter is close to zero. Since this stop criterion is not easy to apply in practice, we can use an equivalent condition instead. As the training continues around the optimal regularization parameter, the relative change of the validation error (RCVE), which is equal to $|(F_v(w_{k+1}) - F_v(w_k))/F_v(w_{k+1})|$, should be small. In the previous example, the stop criterion is considered to be reached when the relative change of the validation error between any two adjacent iterations within a sliding 15-epoch window is less than 0.00005. The trajectories of the parameter α and the validation error versus the training epochs are presented in Figure 17.

Since the optimal regularization parameter is determined by the end of the first stage training, the combined training with fixed α is straightforward. Starting from the latest updated weight vector, the number of training iterations required in the second stage is very limited.



(a). Parameter α (b). Validation error

Figure 17 Incremental Updating

Simulation Testing

We have discussed the methods for applying of the SDVR algorithm in the previous section. Recall that our motivation for proposing the SDVR algorithm is to obtain a more efficient algorithm for adaptive regularization than Larsen's method. In this section, we will compare these two approaches through simulation. For convenience, we will refer to Larsen's method as the FDVR (first derivative of validation error based regularization) algorithm. One of our measures of algorithm performance is test set error. The other measure is computational complexity. In addition, we will also consider how easy the algorithm is to apply.

Method and Procedure

Let's consider the same example as we used in the previous section. The example is a two-input parabolic function which has its single output corrupted with zero mean and 0.04 variance noise. The objective is to approximate the true function with neural network models trained from the noisy data. We will compare the results obtained using two different adaptive regularization methods.

In order to make a fair comparison, let's specify some conditions which are common to both algorithms under investigation. First, since the SDVR algorithm and the FDVR algorithm both adapt the regularization by minimizing the error of the single validation set, it is convenient for us to use the same data splitting ratio of 110/231.

The other condition is the choice of initial weights. Thirty different sets of initial weights are used for each run of the simulation. Therefore, it is the statistic results that will be compared in the simulation testing.

Two empirical tests are designed to compare the average model performance and the computation load of the two algorithms. In the first test, we are interested in the effect of the initial regularization parameter on the training results. Five different initial $\alpha's$, which cover a practical range of normal settings, are tested with each algorithm. For the FDVR method, the initial learning rate η is set to 0.0001, and will be multiplied by 1.2 if the validation error decreases, or by 0.5 if the validation error increases. For the SDVR algorithm, the learning rate is determined by the iterative Hessian, and μ_α is initialized to 0.005. A multiplying constant, equal to 1.05, will be used as the validation error increases. In the inside loop training, since both algorithms employ the Gauss-Newton method with the Levenberg-Marquardt implementation, there is no need to choose different initial settings for μ_w.

In the second test, we use the same initial regularization parameter, but change the initial learning rate η in the FDVR algorithm, and the μ_α value in the SDVR algorithm. Through this test, we will investigate how these initial parameters affect the training results.

A large testing data set, 2601 samples from a 51 by 51 grid over the input plane, is used to test the model performance. The model output is calculated using the resulting weights from the combined training, and the model performance is measured by the mean squared error (mse) between the model outputs and the true function outputs. The results are averaged over 30 trials under different initial weights.

To measure algorithm complexity, we use the number of float point operations (FLOPS). Since there is no significant difference in computation complexity between the two algorithms during the final combined training, we only measure the FLOPS during the first stage search. The computational load is averaged over 30 trials.

We also measure the variation in resulting regularization parameter and use it as an additional index of the model sensitivity to the initial conditions.

Results and Discussion

The results of the simulation testing are presented from Table 3 to Table 6. In Table 3, measurements obtained using the FDVR algorithm are summarized under five different initial regularization parameters with initial learning rate of 0.0001. The item marked by the overline is the mean value averaged over 30 trials, and the prefix 'std' is the notation for the standard derivation of the item after the underscore.

Results	α_0=1.0	α_0=0.1	α_0=0.01	α_0=0.001	α_0=0.0001
$\overline{\text{mse}}$ $\times 10^{-3}$	5.23	2.59	7.15	3.54	105
std_mse $\times 10^{-3}$	0.06	0.32	4.26	1.00	4.59
$\overline{\text{flops}}$ $\times 10^{10}$	0.87	1.25	6.40	4.41	3.55
std_flops $\times 10^{10}$	0.28	0.76	4.23	4.00	2.45
$\overline{\text{alpha}}$ $\times 10^{-1}$	9.99	0.99	0.46	0.99	58.8
std_alpha $\times 10^{-1}$	0.002	0.83	0.27	0.62	216

Table 3 FDVR Results with Different Initial α Value (η_0= 0.0001)

We can see from Table 3 that the effect of the initial regularization parameter on the training results is dramatic for this training example. The model performance is best for the condition $\alpha_0 = 0.1$, and becomes worse as α_0 changes in either directions. Also, the mean values of the resulting parameter α in the first and second columns are almost equal to the corresponding initial value. As α_0 decreases, the regularization is adapted during the training. However, since there is no relation between the convergent gradient and the learning rate, updating the regularization parameter is less systematic. Even a small learning rate may cause a large change in α if the gradient at convergence is too large. The same learning rate may only cause a small change if the gradient is too small. As a consequence, the training might be improperly terminated under many different circumstances. For the given range of α_0 in Table 3, the maximum ratio is about 40 for $\overline{\text{mse}}$, about 7 for $\overline{\text{flops}}$, and the variation in $\overline{\text{alpha}}$ is very large.

Results	$\alpha_0=1.0$	$\alpha_0=0.1$	$\alpha_0=0.01$	$\alpha_0=0.001$	$\alpha_0=0.0001$
$\overline{\text{mse}}\ \times 10^{-3}$	2.83	2.88	2.89	2.97	2.94
std_mse $\times 10^{-3}$	0.57	0.61	0.58	0.66	0.53
$\overline{\text{flops}}\ \times 10^{10}$	1.15	1.07	1.14	1.03	1.09
std_flops $\times 10^{10}$	0.21	0.31	0.28	0.36	0.26
$\overline{\text{alpha}}\ \times 10^{-1}$	1.65	1.68	1.58	1.61	1.49
std_alpha $\times 10^{-1}$	0.78	0.76	0.77	0.76	0.75

Table 4 SDVR Results with Different Initial α Value ($\mu_{\alpha 0} = 0.005$)

Table 4 summarizes the SDVR training results for the same initial regularization parameters. The initial μ_α equals 0.005, which is a typical setting for the SDVR algorithm. Although the parameter α_0 changes over a wide range, the training results are less sensitive. Moreover, the simulation results of the SDVR algorithm are not only more consistent, but are also better than those of the FDVR algorithm. The model performance under each condition of Table 4 is comparable to the best performance of Table 3, and the computation load summarized in Table 4 is also much lower than that in Table 3, on the average.

Table 5 is used to evaluate how the initial learning rate affects the training results for the FDVR algorithm. In this test, we set $\alpha_0 = 0.01$, which would not provide adequate smoothness of the model if regularization was not adapted during the training. The different initial learning rates range from 0.0001 to 0.1.

Results	η_0=0.1	η_0=0.01	η_0=0.001	η_0=0.0001
$\overline{\text{mse}} \times 10^{-3}$	3.60	3.23	3.92	7.15
std_mse $\times 10^{-3}$	0.85	1.62	0.40	4.26
$\overline{\text{flops}} \times 10^{10}$	8.13	3.01	5.01	6.40
std_flops $\times 10^{10}$	2.62	2.38	1.70	4.23
$\overline{\text{alpha}} \times 10^{-1}$	1.11	2.05	0.69	0.46
std_alpha $\times 10^{-1}$	1.04	1.40	0.09	0.27

Table 5 FDVR Results with Different Initial η Value (α_0= 0.01)

As shown in Table 5, using the initial learning rate of 0.0001 almost doubles the error index compared with using other initial learning rates. The best results for both model performance and computation load are achieved with $\eta_0 = 0.01$. Further increasing the initial learning rate does not improve the model performance, but causes serious oscillation in parameter α,which leads to an extreme increase in computation load.

The situation is quite different in Table 6. For the SDVR algorithm, the learning rate is determined automatically. The choice of initial μ_α in a reasonable range does not make a significant difference in model performance, but it may affect the convergence speed to some extent. This can be seen from Table 6, in which the value of initial μ_α is changed from 0.0005 to 0.5. Comparing the results in Table 6 with the results in Table 5, the SDVR algorithm produces a $10 \sim 150$ percent mean error reduction over the FDVR algorithm, and needs only $10 \sim 40$ percent as much computation.

Results	$\mu_{\alpha 0}$=0.0005	$\mu_{\alpha 0}$=0.005	$\mu_{\alpha 0}$=0.05	$\mu_{\alpha 0}$=0.5
$\overline{\text{mse}} \times 10^{-3}$	2.97	2.89	2.90	2.83
std_mse $\times 10^{-3}$	0.61	0.58	0.59	0.54
$\overline{\text{flops}} \times 10^{10}$	1.27	1.14	8.93	8.36
std_flops $\times 10^{10}$	0.32	0.28	0.37	0.36
$\overline{\text{alpha}} \times 10^{-1}$	1.52	1.58	1.54	1.52
std_alpha $\times 10^{-1}$	0.81	0.77	0.72	0.66

Table 6 SDVR Results with Different Initial μ_α Value (α_0= 0.01)

We can conclude from the simulation testing in this section that the basic SDVR algorithm generalizes better and converges faster than the FDVR algorithm. Without prior information about the right choice for parameter α, the FDVR algorithm has to be attempted under many different initial conditions. In contrast, we almost always get good results the first time when using the SDVR algorithm.

Variations of SDVR Algorithm

The discussions so far on the SDVR algorithm have concentrated on the basic scheme, which incrementally updates the regularization parameter in each training epoch. We have tested the algorithm on examples with a moderate data size and a moderate number of network parameters. We have shown that the basic SDVR algorithm works better than the FDVR algorithm does on these particular tests. In practice, the problem complexity may vary from case to case. For some applications we may need to process a large amount of data using networks with thousands of parameters. Under those circumstances, the computational trade off between the inside loop training and the outside loop updating may become an important concern. To reduce the total computation cost, we are interested in different implementations of the SDVR algorithm which are distinguished by the way in which the parameter α is updated.

Although the SDVR algorithm was derived using the incremental iteration method, the updating of the regularization parameter in the outside loop can be implemented over a variable. In this section, we will propose two variations of the SDVR algorithm. The first variation is called convergent updating. This method is similar to the FDVR algorithm in training procedure, but uses a second derivative method instead of gradient descent. The second variation is called conditional updating. This method recalculates the regularization parameter only if the validation error increases, or if the trajectory of the validation error is descending very slowly. With these variations, the SDVR algorithm can be more efficiently applied to a variety of different problems.

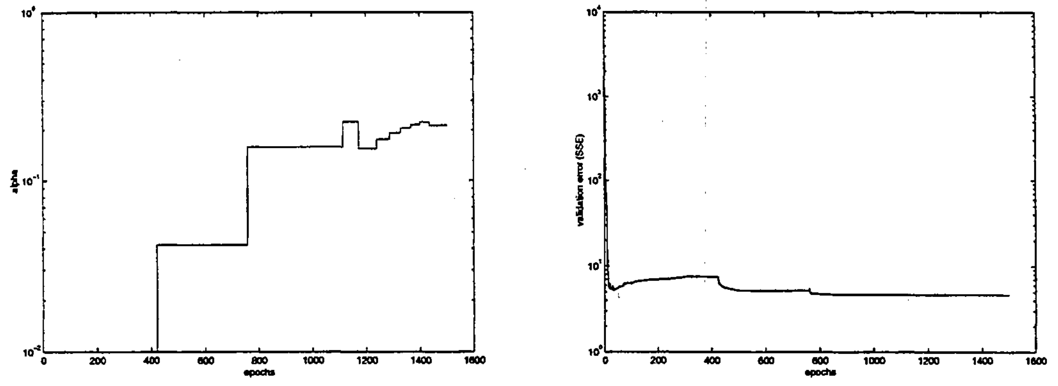
Convergent Updating

Convergent updating is the oppsite of incremental updating. While incremental updating recalculates the regularization parameter in each training epoch, convergent updating keeps training with a fixed α until the optimization algorithm converges. After convergence, a new α will be estimated by using Eq. (115), as with incremental updating. Although the algorithm is implemented with convergent updating, the Hessian in Eq. (115)

is still computed from Eq. (139). This procedure will be repeated several times during the first stage training. The tunable parameter μ_α is set to a small value at the beginning, but will be multiplied by a relatively large constant if the validation error increases at each switching point of α. The first stage training is usually terminated when the RCVE between two subsequent updates is smaller than a predetermined threshold. Since we train the network to convergence for each fixed α, the final choice of the optimal regularization parameter will be the one with smallest validation error among all updatings.

Training Example

Figure 18 shows the trajectories of α and validation error for convergent updating on our previous problem. This algorithm needs only a few iterations in the outside loop, but



(a). Parameter α (b). Validation error

Figure 18 Convergent Updating

requires more training epochs in the inside loop. Starting from $\alpha = 0.01$, the training takes several hundred epochs to reach convergence for the first few updatings of the regularization parameter. As the parameter α approaches the optimal value, the required number of training epochs with fixed α become smaller, since each retraining starts from the previous weight vector. The minimum validation error is reached at the end of training.

In this example, the stable state of the inside loop training is controlled by a 30-epoch sliding window with 0.00005 RCVE threshold. The other specific settings include 0.05 for initial α, and 10 for increment constant of μ_{α}.

Framework Justification

One important feature of the convergent updating is using the basic SDVR framework to compute the new parameter α. Recall that the basic SDVR algorithm was designed for incremental updating. In determining the incremental gradient and Hessian, we assumed that $\boldsymbol{w}_k$ is not a function of α_k. This is not true for convergent updating, because there is an accumulative effect of fixed α on the intermediate weight vector. However, each updating of the network weights in the inside loop is mathematically meaningful only within its own iteration. To compute the new estimate $\boldsymbol{w}_{k+1}$ using Eq. (112), the vector $\boldsymbol{w}_k$ and parameter α_k can always be considered as independent initial conditions, no matter how they were determined previously. From this view, convergent updating is just incremental updating initialized with the previous parameters. As proved early in this chapter, Eq. (110) for calculating the convergent gradient is a special case of Eq. (129) for computing the incremental gradient at the convergent state $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k$. Therefore, without further proof, we can apply the incremental Hessian updating equation to compute the convergent Hessian. In fact, the basic SDVR algorithm derived for incremental updating can be implemented with any updating interval.

Feature Discussion

Now we have the SDVR algorithm implemented with convergent updating. Recall that the FDVR algorithm, which we introduced early in this chapter, was also derived under the condition of the convergent updating. The FDVR algorithm applies a gradient descent method to calculate parameter α, while the convergent SDVR algorithm employs a second derivative method. We compared the FDVR algorithm with the incremental SDVR algorithm in the previous section (from Table 3 to Table 6). Here, using the same example, we provide more simulation results in Table 7 that reflect the features of the convergent SDVR algorithm.

Table 7 is divided into five columns, which represent five different initial regularization parameters. This table is analagous to Table 3, which presents the training results under the same initial conditions by for the FDVR algorithm. A comparison of these two tables allows us to see the difference between the first derivative and the second derivative approaches, because in both tests parameter α is updated only when the optimization algorithm converges in the inside training loop.

Results	α_0=1.0	α_0=0.1	α_0=0.01	α_0=0.001	α_0=0.0001
$\overline{\text{mse}} \times 10^{-3}$	2.57	2.80	3.36	3.18	3.43
std_mse $\times 10^{-3}$	0.48	0.57	0.61	0.72	0.69
$\overline{\text{flops}} \times 10^{10}$	2.22	1.58	4.24	4.71	6.30
std_flops $\times 10^{10}$	1.37	0.90	2.16	1.54	2.31
$\overline{\text{alpha}} \times 10^{-1}$	1.75	1.44	1.01	0.99	0.87
std_alpha $\times 10^{-1}$	0.61	0.69	0.53	0.39	0.35

Table 7 SDVR Results with Convergent Updating

On average, the model performance in Table 7 is better than that in Table 3. Although the FDVR results with $\alpha_0 = 0.1$ and $\alpha_0 = 0.001$ are comparable to the corresponding SDVR results, the among-group variance of $\overline{\text{mse}}$ is larger in Table 3 than in Table 7. No matter what initial condition it starts from, the training with the SDVR algorithm converges properly with little variation. This demonstrates that under the same implementation of convergent updating, the SDVR algorithm is still superior to the FDVR algorithm.

One interesting phenomenon can be observed from Table 7. For this particular example, there is an applicable range of parameter α under which the validation error surface is flat. When the training starts from a small value of α_0, the chance to converge to the left bound of the applicable range is high. Therefore, over 30 trials, the mean of the parameter α is relatively low. When the search begins from the opposite direction of α, the training usually stops near the right bound of the applicable range, and the mean of parameter α is relatively high. This phenomenon is not obvious for incremental updating (see Table 4) because several hundred recalculations for α are made during the first stage of training. After that, the final results are less sensitive to the initial condition. Later in this chapter, we will see that the SDVR results using convergent updating are consistent with SDVR results using incremental updating with respect to the model performance when the data splitting ratio is close to 1. In that case, the validation data set and the training data set are both likely to be representative, and therefore the effect of the local variation of the validation data set on the regularization may be smaller.

Also, as shown in Table 7, the computational load is sensitive to the value of the initial regularization parameter for the convergent SDVR algorithm. If the noise level of the data set is underestimated at beginning, by setting the initial regularization parameter too small, then reaching convergence in the inside loop will be computationally expensive, and the convergence of the parameter α will take many training epochs. This problem is common with convergent updating and cannot be avoided.

Next we will investigate a different implementation which can improve the efficiency of the SDVR algorithm by reducing the unnecessary computation in both the inside loop and the outside loop.

Conditional Updating

Conditional updating is motivated primarily by computational simplicity. Since adapting regularization with the SDVR algorithm requires both inside loop training and outside loop calculation, the total computational load is an important concern in selecting the appropriate implementation. Incremental updating and convergent updating have limitations for some applications. On the one hand, updating α too frequently may increase computational overhead in the outside loop. On the other hand, training a network to convergence with fixed α involves unnecessary computation in the inside loop if the validation error has increased long before.

As a solution to these problems, conditional updating is proposed as a compromise between incremental updating and convergent updating. In this implementation, the updating of the regularization parameter is made over a variable interval controlled by some performance constraints. This variable interval, measured by the number of training epochs in the inside loop, is at least several epochs long; thus, conditional updating does not require as many iterations as incremental updating in the outside loop. In addition, since conditional updating uses a less expensive stop control in the inside loop, it does not require as many training epochs as convergent updating with fixed α.

Method of Implementation

The conditional updating can be implemented with the same SDVR framework by making a simple modification to the stop criteria of the basic scheme. The general training procedure for using conditional updating can be described as follows:

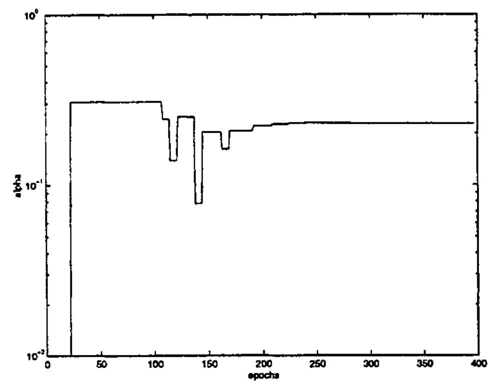
1. Divide the available data set into training and validation subsets using a proper splitting ratio.
2. Initialize the network weights and the regularization parameter.
3. Optimize the training performance function with fixed α by iteratively updating the weight vector using Eq. (112) until the validation error increases, or the RCVE within a sliding window is sufficiently small.
4. Evaluate the performance on the validation data set. If the stop criterion is not satisfied, update the regularization parameter using Eq. (115).
5. Go back to step 3 if the stop criterion is not met. Otherwise, terminate the first stage training.
6. Put the training data set and the validation data set together for combined training using the last updated regularization parameter.

Comparing conditional updating with the other two implementations, the main difference in the application recipe is in step 3. If the network training starts from an improper initial parameter α, then an increase of the validation error will be observed before training converges. Recalculating the regularization parameter at this switching point will avoid the further unnecessary training with the initial α, as occurs with convergent updating.

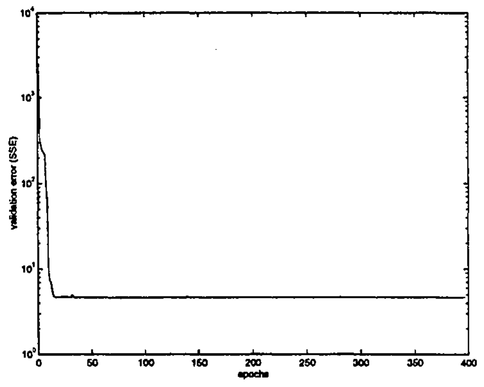
During the transition of α, the validation error may increase after a number of training epochs, or its trajectory may be descending but very flat. We stop the inside loop training in either case. To test these conditions, we use a sliding window for RCVE that is shorter than that for convergent updating (e.g., 10 epochs). This guarantees that the computation cost using conditional updating with fixed α never exceeds the computation cost using convergent updating.

Training Example

Figure 19 (a) illustrates the switchings of α during the first stage training with conditional updating. Beginning at $\alpha = 0.01$, the regularization is adapted quickly due to the increase of the validation error. The total inside loop training takes about 400 epochs and about 30 updates of the regularization parameter. Compared with Figure 18 (a), conditional updating uses about 20 more iterations than convergent updating in the outside loop, but saves about 1200 training epochs in the inside loop. The most significant saving is observed for the initial setting. In Figure 19 (b), we can see that the regularization is adapted at each increase of the validation error just occurs. The final adjustment of the parameter α is small, since the validation error is almost unchanged.



(a). Parameter α



(b). Validation error

Figure 19 Conditional Updating

There is also an apparent difference in the α trajectory between conditional updating and incremental updating. This can be understood by comparing Figure 19 (a) with Figure 17 (a). While incremental updating recalculates the parameter α more than 200 times during the training, conditional updating requires much fewer computations in its nearly 30 iterations. Moreover, the transition of α in Figure 19 (a) is smoother than that in Figure 17 (a), especially during the early stage.

In this training example, the initial μ_α is the same as that used in the other two cases, but the incrementing constant for μ_α was changed to 2. The stop criterion in the outside loop for conditional updating is similar to that for incremental updating, but uses a shorter 5-point sliding window.

Feature Discussion

As with incremental updating and convergent updating, we investigate the sensitivity of the training results to the initial condition through a simulation test using conditional updating. The results are summarized in Table 8.

Results	$\alpha_0=1.0$	$\alpha_0=0.1$	$\alpha_0=0.01$	$\alpha_0=0.001$	$\alpha_0=0.0001$
$\overline{\text{mse}}\ \times 10^{-3}$	2.62	2.68	2.96	2.88	2.77
std_mse $\times 10^{-3}$	0.50	0.55	0.65	0.66	0.59
$\overline{\text{flops}}\ \times 10^{10}$	1.69	0.95	0.81	1.12	1.06
std_flops $\times 10^{10}$	0.74	0.63	0.50	0.73	0.80
$\overline{\text{alpha}}\ \times 10^{-1}$	1.67	1.54	1.28	1.40	1.38
std_alpha $\times 10^{-1}$	0.75	0.69	0.76	0.79	0.77

Table 8 SDVR Results with Conditional Updating

Now, we are interested in comparing the results in Table 8 with the results in Table 7 and Table 4. Recall that Table 7 is for convergent updating and Table 4 is for incremental updating. These results are based on the same example, modeled with the same neural network architecture, investigated under the same initial weights and initial regularization parameters, trained with the same SDVR framework but with different implementations.

Considering the model performance in Table 8, the conditional SDVR algorithm works as well as the other two SDVR schemes. Each column in Table 8 has almost the same error mean and variance, which indicates that conditional updating is also less sensitive to the initial setting of the regularization parameter. Starting from very small α_0, the training with the conditional SDVR algorithm is three times faster than the training with the covergent SDVR algorithm. The average computational cost over the five groups in Table 8 is similar to that in Table 4. Thus, for this example, the conditional updating seems as efficient as the incremental updating.

Optimal Use of the SDVR Algorithm

We have investigated three implementations of the SDVR algorithm, incremental updating, (which is the basic scheme,) convergent updating and conditional updating. These three implementations are based on the same SDVR framework, but are distinguished by the way in which the parameter α is updated.

The incremental updating method recalculates the parameter α in each training epoch. Therefore, the number of the iterations required to update the weight vectors in the inside loop equals the number of iterations used to reestimate the regularization parameter in the outside loop. The convergent updating method optimizes the training performance function with fixed parameter α and then determines the next estimate of the parameter α. It takes many more training epochs in the inside loop but only a small number of calculations in the outside loop. The conditional updating method usually needs more iterations in the outside loop than the convergent updating method does, but much less computation in the inside loop.

Now the question is how to make optimal use of the SDVR algorithm. How do we choose the most suitable implementation for the specified problem, and what are the optimal parameter settings for each implementation? Definitely, for the first task, the model performance is the main concern in evaluating the algorithm. However, it has been shown that there are no significant variations in model performance among the three SDVR implementations. Since the computation load involved in the inside loop training and the outside loop updating is different for each implementation, the total computation cost may become a dominating factor in decision-making.

Previously, we discussed the training results of the three implementations for a particular example and a fixed data size. In the following, we will investigate how the training results change as the problem complexity varies. A reasonable index for the problem complexity is the size of the Jacobian matrix, which is the product of the number of training patterns and the number of network parameters.

Using the same parabolic function training example and the same neural network structure, we can vary the training complexity by changing the number of patterns. In this empirical investigation, data is evenly sampled over the input space with three different data set sizes. Normally distributed random noise with zero mean and 0.04 variance is added to the function outputs. The data splitting ratio (the number of validation samples to the number of training samples) is 220/221 for the first data set, 480/481 for the second, and 1300/1301 for the third. The initial regularization parameter is set to 0.01 for all cases, and 30 trials with different initial weights are averaged to obtain the final results. The results are summarized in Table 9, where INC represents incremental updating, CVG refers to convergent updating ,and CDT is for conditional updating.

It can be concluded from the table that the three SDVR implementations work equally well with respect to the model performance. However, the computation costs are quite different among them. Incremental updating is as efficient as conditional updating for the first two data sets, but conditional updating is more cost-effective for the third data set. Convergent updating is much slower than the other two methods for moderate data size, but for large data size, its computation cost is close to incremental updating. Therefore, as a guide to the user, we suggest using either incremental updating or conditional updating if the size of the Jacobian matrix is not too large, and using conditional updating method otherwise. If the computational load is not a big concern, then convergent updating is always a useful method. The other factors, like data splitting ratio and noise level, may affect the comparison results, but they were not considered in this chapter.

Method	No. Data	$\overline{\text{mse}}$ $\times 10^{-3}$	std_mse $\times 10^{-3}$	$\overline{\text{flops}}$ $\times 10^{10}$	std_flops $\times 10^{10}$	$\overline{\text{alpha}}$ $\times 10^{-1}$	std_alpha $\times 10^{-1}$
INC	441	2.86	0.05	1.28	0.18	1.20	0.08
CVG	441	2.49	0.41	3.03	1.06	1.27	0.15
CDT	441	2.79	0.42	1.60	0.77	1.20	0.15
INC	961	1.82	0.06	2.16	1.22	6.42	0.55
CVG	961	1.89	0.32	4.50	0.86	6.53	0.64
CDT	961	1.83	0.06	1.49	0.75	6.79	0.45
INC	2601	1.00	0.12	5.32	0.72	2.50	0.98
CVG	2601	1.03	0.08	6.87	2.00	2.61	0.79
CDT	2601	1.04	0.05	2.65	1.10	2.71	0.79

Table 9 Comparison of Variations of SDVR Algorithm

The second task in the optimal use of the SDVR algorithm is making a proper parameter setting with each implementation. We have discussed the insensitivity of the training results with each implementation to the initial regularization parameter. We have also shown that training results with incremental updating are insensitive to the initial μ_α within a normal range. The effect of initial μ_α on the training results of convergent updating and conditional updating is similar to that for incremental updating.

Now let's consider the incrementing constant for μ_α, which is usually set to different values for different implementations. Recall that in Eq. (139), we multiply μ_α by a incrementing constant each time the validation error increases. The parameter α will be forced to be stable if the value of μ_α becomes very large. Therefore, in incremental updating, this constant is usually set to 1.05~1.2, which allows over a thousand updatings before μ_α dominates the learning rate. In convergent updating, α is recalculated only at the convergent condition, therefore not too many iterations are expected. Thus, a larger incrementing constant, 10 for example, can be used. The best value for conditional updating is around 2, which is greater than the value for incremental updating, but less than the value for convergent updating.

Summary

This chapter presented a thorough investigation of methods to adapt regularization during the training of feedforward neural networks through actively using a single validation data set. The FDVR algorithm has suffered from the problem that both the resulting model performance and the computational cost are sensitive to the initial α and the initial learning rate. To improve the generalization performance of neural network models, we proposed a new SDVR algorithm by treating α_k and w_k as independent variables. We have shown how the basic SDVR algorithm, as well as its two variations, can be applied to adapt the regularization parameter in order to minimize the validation error.

In the SDVR algorithm, we use a second derivative algorithm in both the inside loop training and the outside loop updating to achieve quick convergence. The Hessian matrices of the training data and the validation data are both approximated with the Gauss-Newton approximation. We have shown that the additional computational overhead in the

SDVR algorithm is limited, since intermediate results and conventional routines are common to updating the weight vector and calculating the regularization parameter.

We have placed the three SDVR implementations (incremental updating, convergent updating, and conditional updating) into a common mathematical framework. The tests on numerical examples demonstrate that the three SDVR implementations work better than the FDVR algorithm with respect to the resulting model performance, sensitivity to the initial conditions, and training efficiency. The tests also indicate how to choose the best implementation to reduce the computational load according to the problem complexity.

Note that simulations and comparisons made in this chapter are concentrated on the validation set based regularization only. We are interested to know if the SDVR algorithm works better than other widely used techniques in improving generalization. This comparison is more difficult to make because each algorithm may have its own best application scope. However, gaining an insight into it will help us to make better use of each technique for future applications.

In the next chapter, we will compare the SDVR algorithm with other previously discussed algorithms under more complicated conditions from an application point of view.

CHAPTER 6

EMPIRICAL ALGORITHM COMPARISON

Objectives

In this chapter, simulation experiments are designed to compare the generalization capabilities of the SDVR algorithm, Bayesian regularization, cross-validated early stopping, and retrained early stopping. The relative advantages and limitations of each algorithm are discussed, based on simulation results.

Introduction

We proposed a new RTES procedure in Chapter 4, and a new SDVR algorithm in Chapter 5. In this chapter, we will compare the generalization capability of these new methods with Bayesian regularization and cross-validated early stopping using a sufficiently complicated neural network structure. Bayesian regularization and cross-validated early stopping were described in Chapter 3 and Chapter 4, and are procedures which have been found to have good performance in a wide range of applications. We are attempting to determine if the proposed methods have any advantage over these two algorithms. However, different algorithms may perform best on different problems, and it is therefore not possible to recommend a single universal optimization algorithm. Instead, we highlight the relative advantages and limitations of each algorithm.

The algorithm comparison presented in this chapter will be based on simulation experiment results only, so that the generalization error can be computed with respect to the true function output. We will describe the specific experiment design in the first section, and will discuss the simulation results in the second section. This chapter analyzes the effects of many influential factors, such as the pattern/parameter ratio, the data splitting ratio and the noise level. The conclusions drawn from this study will help us to better understand each training algorithm from an application point of view.

Experiment Design

We have proposed and demonstrated a new RTES procedure, and a new SDVR algorithm in the previous chapters. It is a common practice to compare a new algorithm with some currently well-used algorithms to understand its advantages and limitations. In this chapter, we compare the RTES and SDVR methods with Bayesian regularization and cross-validated early stopping. (The latter two are implemented with the Levenberg-Marquardt algorithm in the Matlab Neural Network Toolbox.) The goal of the comparison is to investigate the generalization capability of each candidate algorithm. In addition, the computational cost required for each candidate is also considered.

The algorithm comparison presented in this chapter will be based on simulation experiment results only. We will use the parabolic function defined in Chapter 5 as the true function, and will approximate this function with neural network models. From an application point of view, neural network users want to know how to select a suitable algorithm according to the sample size. They want to know how to divide total samples into the training set and the validation set. They want to know which algorithm generalizes better with noisy data. In order to answer these basic questions, we will investigate the effects of the pattern / parameter ratio (PPR), the data splitting ratio (DSR), and the noise level. In practice, these factors interact with each other, and the effect of these factors on the training results may not be same with different candidate algorithms. In order to make a fair comparison, the experiment should be designed using many combinations of the selected influential factors.

In our experiments, we use a fixed and complicated network structure to run simulations. This does not mean that simpler network structures cannot be used. However, our main concern is the ability of an algorithm to control the effective complexity of the model in the case when a complicated network structure is used.

In the rest of this section, we will define the specific options of the experiments and will describe the comparison method used in this empirical study.

Pattern/Parameter Ratio

Consider the previously used parabolic function

$$Z = 2.5 + 5.0X(2-X)Y(2-Y),$$

and a three-layer 2-10-10-1network architecture with 151 parameters. The three basic data sets, $n = 961$, 441 and 225, are evenly sampled along a 31 by 31, 21 by 21, and 15 by 15 grid across the X-Y plane ranging from 0 to 2. The pattern / parameter ratio with respect to each data size is 961 / 151, 441 / 151, and 225 / 151. The smallest ratio is slightly larger than 1, which is representative of the cases with sparse data. The largest ratio is greater than 6, which represents situations with a large number of samples within the non-asymptotic region, where the computation cost is not too high for simulation tasks. By using different training algorithms, the challenge is how to manipulate the available samples in each data set to minimize the generalization error measured on a new large testing set.

Note that the effective pattern / parameter ratio is different during the training among the four candidate algorithms. Cross-validated early stopping uses only the training patterns to calculate the network weights, but uses the validation patterns to terminate the training as soon as the validation error increases. The RTES and SDVR methods utilize a two stage training method. The training patterns and the validation patterns play different roles during the first stage training, but the final results are optimized during the second stage training by putting the two data sets together. Bayesian regularization uses all available data to update the network weights and the performance function parameters during the whole training process. The difference created by the pattern / parameter ratio will be discussed, along with the data splitting ratio.

Data Splitting Ratio

Within each data set, we divide the samples into the training set and the validation set. The data splitting ratio (DSR) is defined as the number of validation examples divided by the number of training examples. The DSR has no effect on Bayesian regularization, but may affect the training results with cross-validated early stopping, the RTES and SDVR methods.

Our experiments are simulated under three different data splitting ratios, 1 / 1, 1 / 3 and 1 / 7. In each case, the validation patterns are evenly sampled in the input space. The largest ratio is Larsen's frequent setting for validation-set-based regularization [LaHa96], under which the number of training samples is about as same as the number of validation samples. The smallest ratio is larger than, but close to, Amari's criterion for cross-validated early stopping [AmMu97]. Since Amari's criterion is derived with the asymptotic assumption, we slightly increase the ratio for non-asymptotic use. The intermediate ratio, which is about 1 / 3, is also a common setting in many applications. These settings cover a wide range of data splitting. The best ratios for the candidate algorithms should fall within these settings.

Noise Level

We use four noise levels in our numerical experiments. The random noise added to the outputs of the parabolic function has a Gaussian distribution with zero mean and standard derivation of 0.3, 0.2, 0.1 and 0.05 respectively. Figure 20 illustrates typical examples of noise corrupted data with different noise levels.

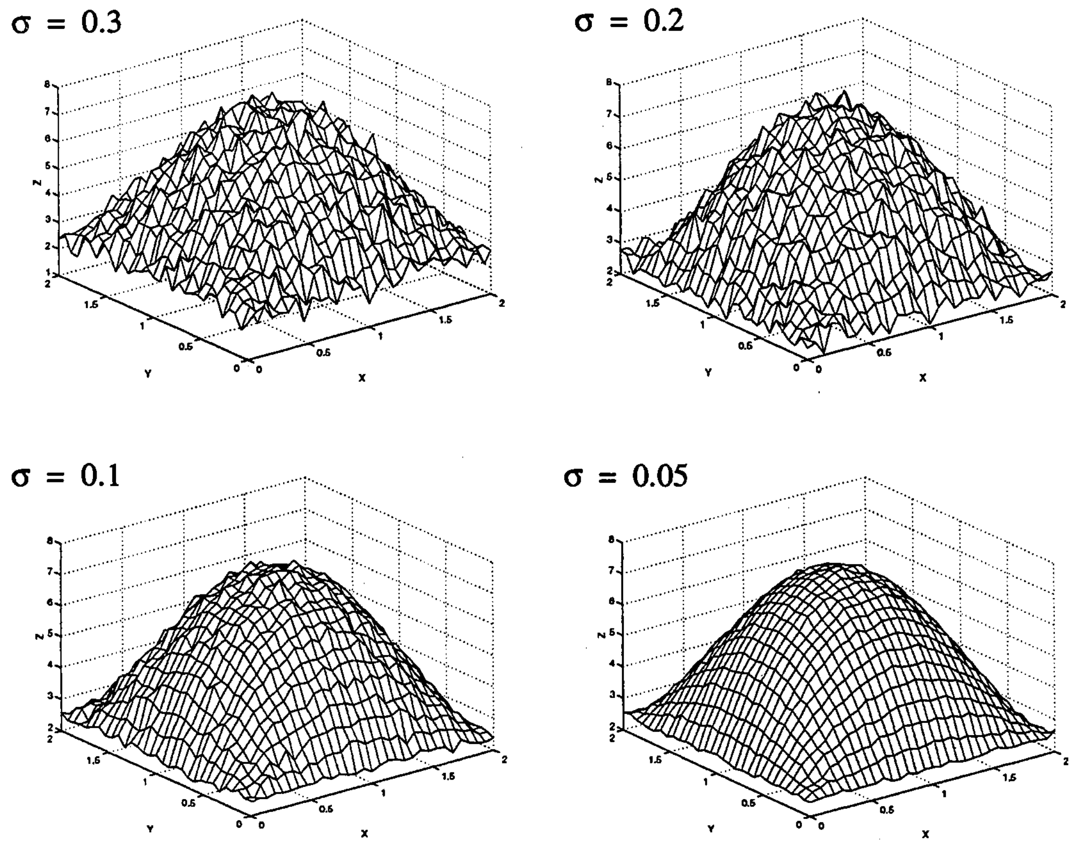


Figure 20 Noise Corrupted Samples

We can see from this example that $\sigma = 0.3$ is a representative of very noisy data, while $\sigma = 0.05$ is almost noise free. The simulation results from these different noise levels will help us to know the target reconstruction capability of each candidate algorithm.

Average Generalization Error

The purpose of using regularization and cross-validated early stopping is to improve the generalization performance of the model. Therefore, after each individual trial, we measure the generalization error (mean-squared-error) between the model outputs and the true function outputs for a given large testing set. For neural network models, different initial weights and sample realizations may lead to different results. Thus, a rational algorithm comparison should not be based upon the training result from only a single realization of the noisy set and fixed initial weights. Instead, it should rest on the training results of statistically adequate resamplings from the population space. Given the data splitting ratio and the noise level, we average the generalization error over 30 trials with different initial weights and random noise realizations. This measurement is a reasonable estimate of the expectation of the generalization error with respect to the initial weight distribution and the noise distribution. In the next section, we will use average generalization error as the algorithm performance index throughout the discussion.

In summary, simulations in this chapter use three pattern /parameter ratios, three data splitting ratios, and four noise levels. For each combination, the testing results are averaged over 30 trials with different initial weights and random noise. The number of trials for Bayesian regularization is $3 \times 4 \times 30 = 360$, because the data splitting ratio does not apply to the Bayesian method. The number of trials for each of other algorithms is $3 \times 3 \times 4 \times 30 = 1080$. The total number of simulations required for these tests is 3600! We will see in the next section that these trials provide adequate information for us to evaluate the effect of the different influential factors on the training results, and to draw meaningful conclusions for future applications.

Results and Discussions

Now let us compare the four candidate algorithms under the different conditions which were specified in the experiment design. We provide the empirical results in a number of tables. These tables are summarized with respect to the different influential factors, and the results are discussed with respect to each factor. We will present a general comparison first, which is made under the relatively optimal testing conditions for each candidate algorithm. Then, the effect of the data splitting ratio on the training results of the relevant algorithms will be evaluated. These results give us a basic idea of the generalization capability of neural network models, and highlight the relative advantages of each algorithm.

General Comparison

The comparisons made here cover average generalization error, variability over different trials and computation cost.

Average Generalization Error

The general results of the empirical algorithm comparison is summarized in Table 10, where PPR represents pattern / parameter ratio, BR refers to Bayesian regularization, and ES refers to cross-validated early stopping. The three pattern / parameter ratios are separated by the double lines in the table, and the four noise levels are summarized in different columns.

Table 10 shows that the pattern / parameter ratio and the noise level of the data play a fundamental role in neural network modeling. For a fixed noise level, the average generalization error decreases as the pattern / parameter ratio increases. Also, for the fixed pattern / parameter ratio, the predictions become worse if the samples used for the learning are corrupted with the higher level of noise. This is true in general for each of the training algorithms. In order to generate a good neural network model, it is important to get adequate training patterns to learn the underlying function, and to improve the accuracy of measurements.

PPR	Training Algorithm	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.3$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.2$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.1$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.05$)
961 / 151	SDVR	30.41	17.93	6.18	1.62
	BR	39.86	18.42	5.05	1.42
	ES	37.78	20.36	5.99	1.85
	RTES	32.60	17.68	5.27	1.50
441 / 151	SDVR	55.23	29.90	10.71	2.88
	BR	69.78	34.73	11.40	3.13
	ES	73.62	40.85	10.98	3.69
	RTES	64.51	32.98	9.02	2.74
225 / 151	SDVR	100.27	49.97	18.58	6.05
	BR	125.59	61.17	20.25	6.30
	ES	140.60	67.63	21.02	6.15
	RTES	123.30	55.38	16.65	5.25

Table 10 General Empirical Algorithm Comparison

However, using different algorithms can make a difference in generalization error. To make this clear, let's divide each entry in Table 10 by its corresponding SDVR error term. The normalized results are listed in Table 11, in which, the average generalization error for the SDVR algorithm equals 1.00.

It is observed from Table 10 and Table 11 that the SDVR algorithm is best suitable to the high noise data ($\sigma = 0.3, 0.2$). It wins competition for different pattern/parameter ratios under those conditions. When the data become less noisy and the PPR is not high, the

SDVR algorithm is ranked in the second place (following the RTES method) according to the generalized performance.

PPR	Training Algorithm	$\overline{\text{MSE}}$ (norm) ($\sigma = 0.3$)	$\overline{\text{MSE}}$ (norm) ($\sigma = 0.2$)	$\overline{\text{MSE}}$ (norm) ($\sigma = 0.1$)	$\overline{\text{MSE}}$ (norm) ($\sigma = 0.05$)
961 / 151	SDVR	1.00	1.00	1.00	1.00
	BR	1.31	1.03	0.82	0.88
	ES	1.24	1.14	0.97	1.14
	RTES	1.07	0.99	0.85	0.92
441 / 151	SDVR	1.00	1.00	1.00	1.00
	BR	1.26	1.16	1.06	1.09
	ES	1.33	1.37	1.03	1.28
	RTES	1.17	1.10	0.84	0.95
225 / 151	SDVR	1.00	1.00	1.00	1.00
	BR	1.25	1.22	1.09	1.04
	ES	1.40	1.35	1.13	1.02
	RTES	1.23	1.10	0.90	0.87

Table 11 Normalized Results for General Algorithm Comparison

The RTES method has obvious advantage for the low noise data ($\sigma = 0.05, 0.1$). It generalizes almost equally well as the Bayesian method when the PPR is high, and produces the best networks when the PPR decreases. Even for high noise data, the RTES method has the second best performance among the candidate algorithms. It is superior to conventional early stopping under each simulation condition.

The Bayesian method predicts better than the other algorithms when the noise is low and PPR is high. This is not surprising if we consider the assumptions that the Bayesian method rests on. In Bayesian regularization, the Gaussian approximation to the posterior probability density function is used to make analysis easy, which is rational in the case of the large data size. However, with small data size, especially for high noise data, the simple Bayesian model may not work well, even though we put all available data in the training set. Under this circumstance, performing regularization to minimize validation error or using retrained early stopping is more effective in reducing the generalization error.

Cross-validated early stopping does not work well for the high noise data. However, as can be seen from Table 10 and Table 11, the networks trained with the ES method are comparable to the SDVR algorithm and Bayesian method for some low noise cases.

Variability in Generalization Error

The previous discussion concentrated on the average generalization error. Now let's consider the variance of the generalization error.

PPR	Training Algorithm	std $\times 10^{-4}$ ($\sigma = 0.3$)	std $\times 10^{-4}$ ($\sigma = 0.2$)	std $\times 10^{-4}$ ($\sigma = 0.1$)	std $\times 10^{-4}$ ($\sigma = 0.05$)
961 / 151	SDVR	8.75	3.69	1.91	0.34
	BR	18.10	5.39	1.35	0.36
	ES	13.91	4.94	1.32	0.53
	RTES	7.30	4.45	1.26	0.37
441 / 151	SDVR	16.72	9.62	2.44	0.69
	BR	29.0	12.36	4.45	0.92
	ES	25.28	10.52	2.44	0.99
	RTES	21.22	8.75	2.18	0.84
225 / 151	SDVR	33.20	14.23	7.65	1.58
	BR	53.83	25.78	8.24	2.08
	ES	58.16	30.78	7.94	1.44
	RTES	41.15	18.59	5.65	1.06

Table 12 Standard Deviation of Generalization Error over 30 Trials

The standard deviations of the generalization errors are summarized in Table 12. It can be concluded that the SDVR and RTES methods generalize more consistently than the Bayesian method and cross-validated early stopping for the high noise data. For the low noise data, the RTES method still shows a smaller diversity in the training results than cross-validated early stopping, but the SDVR algorithm has no obvious advantage over the other algorithms with respect to the variance. Note that the comparison made in this chapter is based on 30 trials for each testing condition, in order to have a reasonable computation load. With this number of trials, one particular case with a large error may contribute significantly to the variance calculation. In a future study, we may increase the number of trials to some extent, and combine future and current results in our statistical analysis.

Computation Cost

The computation cost summarized in Table 13 is measured with the average floating point operations ($\overline{\text{flops}}$) over the 30 trials in each testing condition.

PPR	Training Algorithm	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.3$)	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.2$)	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.1$)	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.05$)
961 / 151	SDVR	7.81	8.26	15.73	15.0
	BR	6.90	7.72	10.70	13.3
	ES	2.92	3.18	4.66	7.33
	RTES	1.96	2.41	4.97	4.87
441 / 151	SDVR	5.11	5.84	8.70	11.8
	BR	3.70	4.65	8.92	11.1
	ES	1.17	1.36	2.42	3.67
	RTES	1.31	1.12	2.62	2.49
225 / 151	SDVR	3.98	3.81	5.52	7.46
	BR	2.40	3.37	5.83	7.76
	ES	0.73	0.77	1.28	1.90
	RTES	0.78	0.68	1.44	2.17

Table 13 Computation Cost with Different Algorithms

Compared with the SDVR algorithm and Bayesian regularization, the RTES method and cross-validated early stopping require fewer computations. This is because the latter two procedures are normally used along with the simple error function. The SDVR algorithm and the Bayesian method both need to update the regularization parameter many times, which usually requires more calculations. In most applications, however, we are more interested in accurate results than in the time required to computer the answer. Even the SDVR and Bayesian algorithms can train practical networks in an acceptable amount of time.

The computational cost of the SDVR algorithm is close to the computational cost of the Bayesian method, especially for the low noise data. We set similar stop criteria to control the relative change of the validation error with the SDVR algorithm and to control the relative change of the training error in Bayesian regularization. Even though the computation time is not too critical for off-line training, cost-effective training algorithms are always desired. To reduce the computational load, we changed the α and β update interval from one training iteration to five iterations for both algorithms, and found that this modification makes adaptive regularization more efficient.

Note that for some entries in Table 13, the RTES method is more computationally efficient than the ES method. This is true since each of them may use its own optimal data splitting ratio in applications. The computational cost for the RTES method with a large DSR might be lower than the cost for the ES method with a small DSR. The effect of the data splitting ratio on the model performance when cross-validation is used will be discussed next.

Effect of Data Splitting Ratio

It is believed that the data splitting ratio is important to the final model performance if cross-validation techniques are used. However, the best data splitting ratio is hard to theoretically determined due to the complexity of different problems. In general, selecting a relatively optimal data splitting ratio depends on what training algorithm is used, which specifies the role played by the validation set during the training. In this subsection, we discuss the effect of the data splitting ratio on the SDVR method, cross-validated early stopping and retrained early stopping, and recommend the relatively optimal ratios for each algorithm.

Data Splittings with SDVR Algorithm

To determine the relatively optimal data splitting for the SDVR algorithm, we divide the total samples in each data set into the training set and the validation set with three different data splitting ratios (DSR), which are close to 1 / 1, 1 / 3 and 1 / 7. The simulation results are summarized in Table 14.

It is observed from Table 14 that among the 12 testing conditions, which are specified by different PPR and noise combinations, the 11 largest average generalization errors appear with the data splitting ratio of 1 / 7, and the 11 smallest average generalization errors are obtained with the data splitting ratio of 1 / 1. The errors arising from these two data splittings are different by a scale factor of $1.14 \sim 1.77$. The simulation results using the intermediate data splitting ratio are generally better than those using the small DSR and are worse than that using the large DSR. Without losing generality, we can conclude that equally dividing the total samples into the training set and the validation set is the relatively optimal way to split the data for the SDVR algorithm.

PPR	DSR	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.3$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.2$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.1$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.05$)
961 / 151	480 / 481	30.41	17.93	6.51	1.62
	241 / 720	39.12	18.29	6.18	1.87
	121 / 840	39.43	19.31	8.46	2.41
441 / 151	220 / 221	55.23	29.90	10.71	2.88
	111 / 330	63.42	30.87	15.91	4.79
	56 / 385	78.84	34.14	16.98	4.73
225 / 151	112 / 113	100.27	49.97	18.58	6.05
	57 / 168	104.61	50.87	18.97	8.02
	28 / 197	153.38	57.60	19.19	10.69

Table 14 Effect of Data Splitting Ratio on SDVR Training Results

Some explanations can be given for these results. Recall that the SDVR algorithm uses a two-stage training method. The objective of the first stage training is to determine the optimal regularization parameter which minimizes the validation error. However, if the validation set is not a good representative of future new data, the regularization selected to minimize the error of the validation set may still have a risk to overfit the new data set. Since we use only a single hold-out validation set, it is desired that the validation data cover a wide range to reflect the properties of the potential novel data as close as possible. When the data splitting ratio is small, it is difficult to determine which portion of the data is the best candidate for the validation set that can be used to estimate the generalization error. However, when the data splitting ratio is close to 1 / 1, the validation data set and the training data set are both likely to be representative, and therefore the effect of the local variation of the validation data set on the regularization will be smaller. In that case, even though the model performance at the end of the first training stage is not good enough, due to the lack of training patterns, it will always be improved during the combined training of the second stage if the proper regularization parameter is selected.

Data Splittings with Cross-Validated Early Stopping

Similarly, the DSR related difference in generalization error for cross-validated early stopping is shown in Table 15. It can be seen that the data splitting ratio which favours the SDVR algorithm does not work well for cross-validated early stopping. While the SDVR algorithm achieved its best training results with the highest DSR of 1 / 1, cross-validated early stopping works more successfully on the same problems with DSR = 1/3 (9 cases from 12) and DSR = 1/7 (3 cases from 12).

PPR	DSR	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.3$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.2$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.1$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.05$)
961 / 151	480 / 481	46.22	26.76	8.64	2.44
	241 / 720	42.04	20.36	5.99	1.85
	121 / 840	37.78	22.53	7.16	1.92
441 / 151	220 / 221	107.20	46.22	14.81	4.22
	111 / 330	73.62	40.85	10.98	3.69
	56 / 385	81.36	41.67	11.44	4.35
225 / 151	112 / 113	194.53	90.82	26.32	9.01
	57 / 168	145.14	70.86	21.02	6.15
	28 / 197	140.60	67.63	23.67	9.69

Table 15 Effect of Data Splitting Ratio on ES Training Results

This can be partially explained by Amari's rule, which establishes the asymptotic relationship between the number of the network parameters and the data splitting ratio [AmMu97]. Amari's rule says the more parameters are used in the network, the larger the portion of the total samples that are needed in the training set. When the network parameters vary over a wide range, from 2 to 200 for example, the asymptotic data splitting ratio decreases from 1 / 1 to 1 / 19. Therefore, as long as the number of parameters is greater than 2, more than half of the total samples should be used in the training set in order to make the learning curve as accurate as possible.

Amari's rule does not explain how to modify the data splitting ratio when the number of samples is not asymptotically large. To keep a reasonable validation data size, it is usually best to select a somewhat larger DSR than that determined by Amari's rule. As can be seen from Table 15, the difference in average generalization errors between DSR's of 1 / 7 and 1 / 3 are not significant for most testing conditions.

Data Splittings with retrained Early Stopping

The effect of the data splitting ratio on retrained early stopping can be evaluated from Table 16.

PPR	DSR	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.3$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.2$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.1$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.05$)
961 / 151	480 / 481	32.60	17.68	5.87	1.50
	241 / 720	36.33	18.31	5.27	1.56
	121 / 840	36.64	21.48	6.58	1.75
441 / 151	220 / 221	70.54	32.98	9.96	2.74
	111 / 330	64.51	33.53	9.02	2.80
	56 / 385	75.42	40.53	10.68	4.07
225 / 151	112 / 113	132.48	55.38	17.80	6.21
	57 / 168	123.30	60.64	16.65	5.25
	28 / 197	137.47	66.30	21.76	8.45

Table 16 Effect of Data Splitting Ratio on RTES Training Results

Compared to the ES method, the RTES method requires a relatively large DSR to produce more promising networks. Under the 12 PPR / noise conditions, 6 of them are best trained with DSR = 1 / 1, the others are best trained with DSR = 1 / 3. For the Gaussian noise with $\sigma = 0.2$, the optimal DSR obtained in this example is 1 / 1, which is in agreement with the other example discussed in Chapter 4. But for different noise levels and model complexity, the optimal DSR varies over a range which is problem dependent.

Note that in this study we employ a two stage training method for both the RTES and SDVR applications. It seems best to use a relatively large data splitting ratio during the first stage if the final parameter estimation will be refined by combined training during the second stage. As can be seen from Table 16, using $25 \sim 50$ percent of the total samples for cross-validation might be a nearly optimal choice for this RTES application.

Conclusions

In this chapter, a large number of numerical simulations were performed to investigate the relative advantages and limitations of the SDVR algorithm, Bayesian regularization, cross-validated early stopping and retrained early stopping.

It can be concluded from this study that cross-validated early stopping is computational efficient. It works reasonably well with a relatively small data splitting ratio for the low noise data. For the high-noise data, cross-validated early stopping generates worse models than the SDVR algorithm and Bayesian regularization and should not be recommended in applications.

Bayesian regularization produces the best results when the pattern / parameter ratio is large and the noise level is not too high. It also generates better models than cross-validated early stopping in general for high-noise data. The computational cost of the Bayesian method is slightly lower than the SDVR algorithm, but higher than cross-validated early stopping.

The SDVR algorithm wins the competition for high-noise data. It also performs better than the Bayesian method and cross-validated early stopping for low-noise data when the pattern / parameter ratio is not high. However, the SDVR algorithm needs more computations, and its performance is partially affected by the data splitting ratio. It is found that using a data splitting ratio of about 1 / 1 is relative optimal for the SDVR algorithm. This ratio not only works well for the large data size, but also works well for the small data size.

The RTES method works well for the low noise data. It performs better than cross-validated early stopping in each testing condition. Moreover, when noise is low and the pattern/parameter ratio is not high, the RTES method produces networks which demonstrate smaller generalization error than the SDVR algorithm and the Bayesian method. Since the RTES method is also computationally efficient, it has advantages over other algorithms in low noise applications.

These conclusions can be summarized in the following table which recommends the optimal algorithm selection based on our simulation experiments.

	high-noise data	low-noise data
high PPR	SDVR	BR, RTES
mid. PPR	SDVR	RTES
low PPR	SDVR	RTES

Table 17 Optimal Algorithm Selection

In the next chapter, we will extend the concepts of validation-set-based regularization and retrained early stopping to the Bayesian probabilistic networks, and propose several new methods of validation-incorporated Bayesian learning. We will see how those methods can improve generalization performance and training efficience of the standard Bayesian regularization algorithm.

CHAPTER 7

VALIDATION-INCORPORATED BAYESIAN LEARNING

Objectives

In this chapter, several new training methods, which use validation-incorporated Bayesian learning, are proposed. The objective of these methods is to improve generalization performance and training efficiency of the standard Bayesian regularization algorithm. Simulation results obtained with these new methods are compared with the standard method.

Introduction

In Chapter 3, we discussed the Bayesian regularization technique. The network weights using the Bayesian method are updated to minimize the regularized performance function $F = \beta e^T e + \alpha w^T w$, where the parameters α and β are adapted to maximize the evidence presented in the training data set. In this approach, the validation data set is not needed during the training. The model with the largest evidence on the training data would be expected to generalize well on unseen data. Many applications which demonstrate this correlation have been reported [MacK92] [FoHa97]. For some other applications, however, this correlation is far from perfect.

In this chapter, we will first investigate the correlation between the evidence framework and generalization error, and discuss the advantages and limitations of the standard Bayesian regularization algorithm.

In order to improve the network generalization performance and the training efficiency in the cases when the standard Bayesian method shows its limitations, we propose several novel approaches using validation-incorporated Bayesian learning.

GPE-validated Bayesian regularization (GPBR) links Moody's model selection criterion [Mood92], the generalized prediction error (GPE), to standard Bayesian learning. Instead of training networks to convergence, we terminate the training as soon as the GPE value increases. Since the GPE value can be computed by using the training data only, no validation set is required in this approach. The network parameters can be optimized with the conventional one-stage training method using the whole data set.

The other two methods use a two-stage training method. In validation-set-based Bayesian regularization (VBBR), the total samples are divided into a training set and a validation set. The network weights are updated by using the training data only, but the parameters α and β are recalculated to maximize the generalized probabilistic evidence on the validation data. In this approach, we assume that validation evidence may have a more direct correlation with generalization performance. After the optimal α and β are determined, the combined training is performed to refine the estimation of the network parameters.

Bayesian regularization with retrained early stopping (BRES) also uses a validation data set to determine the optimal α and β. However, in the VBBR implementation, the validation data are actively used to update the regularization parameters. In the BRES implementation, the update of α and β are based on the training data only during the first stage training. The validation data are passively used to terminate the first stage training as the validation error increases. With the BRES method, the optimal α and β are the values which generate the smallest validation error. During the second stage training, the network parameters are reestimated using the combined data set with fixed α and β.

We will compare these validation-incorporated Bayesian methods with standard Bayesian method through simulation experiments. The real-world applications of these methods, as well as other new algorithms proposed in the previous chapters, will be discussed in the next chapter.

Evidence Framework and Generalization Error

As we introduced in Chapter 3, network training with Bayesian regularization is performed in a hierarchical fashion. The first level involves the determination of the most probable network weights for the given α and β, which is equivalent to minimizing the regularized performance function. At the second level, the parameters α and β are optimized to maximize the evidence on the training data [MacK92]. These two-level operations can be performed alternately during the training. Let us rewrite the evidence in Eq. (140)

$$P(D_t|\alpha, \beta) = \frac{P(D_t|\mathbf{w}, \beta)P(\mathbf{w}|\alpha)}{P(\mathbf{w}|D_t, \alpha, \beta)} \qquad (140)$$

where $P(D_t|\alpha, \beta)$, the probability of the model with given α and β which fits the training data D_t, is called the evidence. $P(D_t|\mathbf{w}, \beta)$ is the likelihood function. $P(\mathbf{w}|\alpha)$ is the prior density of the weight vector $\mathbf{w}$, and $P(\mathbf{w}|D_t, \alpha, \beta)$ is the posterior density function of $\mathbf{w}$.

Now consider a single weight parameter w. Figure 21 illustrates the prior density $P(w|\alpha)$ and the posterior density $P(w|D_t, \alpha, \beta)$. As illustrated in Figure 21, if the posterior distribution is sharply peaked in weight space around the most probable value w_{MP}, then we can approximate the area under the curve $P(w|D_t, \alpha, \beta)$ by the value at the maximum times the width $\Delta w_{posterior}$ of the peak. If we take the prior $P(w|\alpha)$ to be uniform over some large interval Δw_{prior}, then Eq. (140) becomes

$$P(D_t|\alpha, \beta) \cong P(D_t|w_{MP}, \beta)\left(\frac{\Delta w_{posterior}}{\Delta w_{prior}}\right). \qquad (141)$$

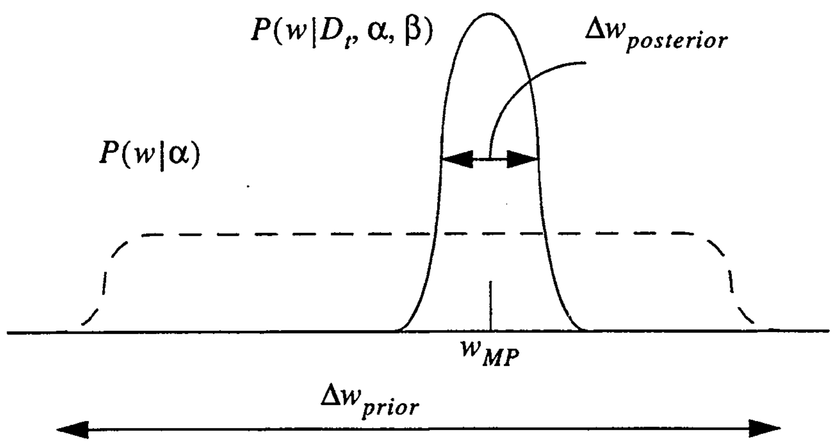


Figure 21 Prior and Posterior of w

The ratio of $\Delta w_{posterior}/\Delta w_{prior}$ is referred to as an Occam factor, which is less than 1. Note that in Eq. (141), the evidence is a product of the likelihood evaluated for the most probable parameter value and the Occam factor. To achieve the largest model evidence, we search for a trade off between fitting the data well and having a relatively wide posterior density spectrum. The value of $\Delta w_{posterior}$ indicates the uncertainty of w_{MP}. A relative large $\Delta w_{posterior}$ means the model performance will not be too sensitive to w_{MP}. For a model with many parameters, each will generate a similar Occam factor. The sequential product of the individual Occam factor will be correspondingly reduced. If two models have the same likelihood evaluated for the most probable parameters, the Bayesian approach favours the simple model because of its larger evidence. From this view, the evidence criterion has a good agreement with other model selection methods. We would expect that the model with the largest evidence would give the smallest generalization error.

However, from other points of view, the distinctions between the evidence and generalization error can be identified [Bish95] from several respects. Basically, the evidence is not measuring the same thing as generalization performance. Generalization performance is calculated with fixed network weights, while the evidence takes account of the complete posterior distribution around the most probable value. The correlation between these two quantities might be strong in some cases, but weak in other cases.

Now let us consider single-input / single-output regression problems. We usually start from Bayesian learning with an adequately complex network structure. Suppose that there are two networks, both with sufficient complexity. The resulting generalization performance would be necessarily similar regardless of the network size if the training converges to the almost same effective number of parameters. However, the calculated evidence may favour the simple model with a relatively small number of network parameters, and reject the more complicated model, which has comparable generalization performance.

For multiple-input / single-output regressions, we often have different problems. Although extensively training a complicated network with the Bayesian framework may overfit the unseen data, the evidence measured on the training data may still be improved, indicating an acceptance of a model with less optimal generalization performance.

In practice, the evidence is difficult to measure accurately. This is because, as we discussed in Chapter 3, calculating the evidence requires the computation of the determinant of the inverse of the Hessian (Eq. (62) and Eq. (66)). Since the product of the eigenvalues of the Hessian is very sensitive to the approximation and the numerical error, the evaluation of the evidence is often inaccurate, especially when the dimension of the parameter space is high. In contrast, the calculation of the effective number of parameters for updating the regularization parameters only needs the computation of the trace of the inverse Hessian. This is the sum of the eigenvalues and is less sensitive to the approximation accuracy and the numerical error. Therefore, it is advantageous to adapt regularization through computing the effective number of parameters in the Bayesian framework. Since our goal in this study is to achieve the optimal generalization performance, even with redundant parameters, we rarely compute the evidence in model comparison.

Here arises a problem. An imperfect correlation between evidence and generalization might be detected for some multiple-input applications, which infers the presence of limitations in the models. In these cases, training a network with the standard Bayesian method to convergence may lead to an increase in generalization error. To overcome these limitations, we propose several novel approaches which apply some generalization-correlated controls to ensure the model validity during the training while still keep using the main framework of Bayesian regularization.

GPE-Validated Bayesian Regularization (GPBR)

GPE-validated Bayesian regularization (GPBR) links Moody's model selection criterion [Mood92], the generalized prediction error (GPE), to standard Bayesian learning. With the GPBR approach, we terminate the training as soon as the GPE value increases, instead of training networks to convergence.

Moody's GPE was briefly introduced in Chapter 2 and Chapter 4. The estimation of GPE from the training examples can be expressed as (see Eq. (95), Chapter 4)

$$\mathrm{GPE} = \frac{n+\gamma}{n-\gamma}\mathrm{MSE}_{\mathrm{train}} ,$$

where n is the number of training examples, γ is the effective number of parameters, and $\mathrm{MSE}_{\mathrm{train}}$ is the mean-squared training error. The GPE is a nonlinear function of γ and $\mathrm{MSE}_{\mathrm{train}}$. After training is completed, a large effective number of parameters is usually associated with a small training error, and vice versa.

GPE was originally developed as an objective guide for architecture selection problems, such as choosing between various classes of models for a specific problem, determining the number of hidden neurons, finding the best value of the regularization parameter. It is rarely used to decide when the training should be terminated. In Chapter 4, we made a new use of GPE to justify retrained early stopping. Here we find an additional use of GPE -- to check the validity of a specific model.

In the Bayesian method, the parameter γ is calculated in each iteration by using the following equation (see Eq. (69), Chapter 3)

$$\gamma = N - \alpha \mathrm{tr}(\boldsymbol{H}_{MP}^{-1}),$$

This is a function of the number of network weights N, regularization parameter α, and Hessian matrix $\boldsymbol{H}_{MP}$ evaluated at the most probable weight vector. Since the effective number of parameters is less sensitive to approximation accuracy and numerical error, we can use it to estimate the GPE directly. We can then monitor the GPE to determine the training stop point. In this approach, we suppose that the GPE is a better generalization performance indicator than the training evidence, and the increase of the GPE may indicate the risk of overfitting.

Figure 22 presents several GPE trajectories for Bayesian learning on a two-input / single-output model with a sufficiently large number of parameters. The training data are corrupted with Gaussian noise of different variances.

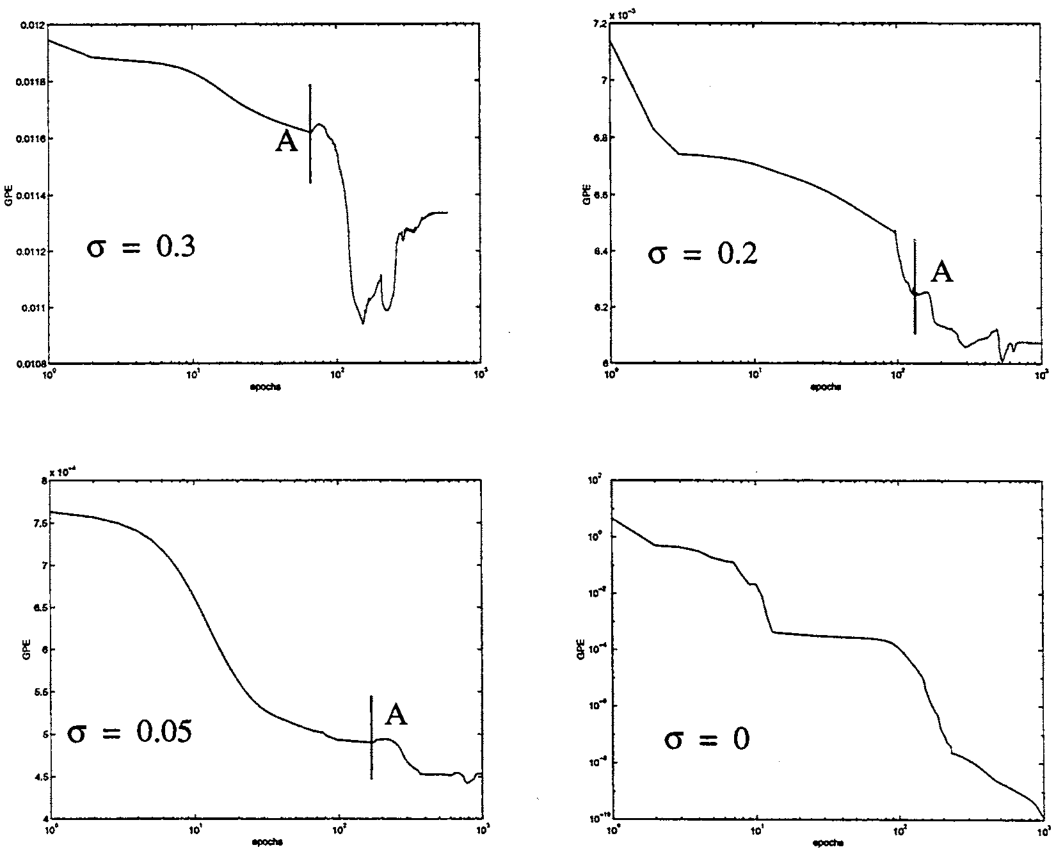


Figure 22 GPE Trajectories with Bayesian Learning

Now let's examine these trajectories in more detail. For noise-free data, the GPE is monotonously descending during the training. However, for noisy data, which is our main concern in this study, the GPE decreases first, then oscillates. Here arises a question: Where is the best stop point based on the given GPE trajectory? As a general stop criterion, we propose to terminate the training at the point A where the GPE reaches its first local minimum. There are several reasons to justify this choice. First, Based on Figure 22, the computational cost up to the point A is inversely proportional to noise variance. This is reasonable, because noise-free data might be trained accurately without overfitting, adapting regularization usually takes longer to obtain high accuracy. Noisy data require models with appropriate smoothness. Shorter training is therefore expected as noise level increases. Secondly, after point A, the change of GPE is not significant. The job of finding point A is simpler than the determination of the global minimum GPE. Furthermore, the GPE trajectory changes in a complicated way after point A, which may reflect the uncertainty of the model validity. The global minimum GPE may not be the best one if it appears after the point A. This is similar to cross-validated early stopping. With early stopping experiments, validation error may occasionally decrease again if we overtrain a network after the validation error first increases. However, the testing error in this case is usually larger than the error tested at the early stopping point.

There is another convenience of using the GPBR method. Since the GPE value can be estimated from the training data only, no validation set is required in this approach. The network parameters can be optimized with the conventional one-stage training method using the whole data set.

Validation-Set-Based Bayesian Regularization (VBBR)

In validation-set-based Bayesian regularization (VBBR), the total samples are divided into a training set and a validation set. The network weights are updated by using the training data only, but regularization is adapted based on the validation data. This concept is similar to the SDVR algorithm. However, in the SDVR algorithm, the parameter α is selected by minimizing the validation error. In the VBBR algorithm, α and β are chosen to maximize the generalized probabilistic evidence on the validation data. Using the VBBR approach, we assume that the validation evidence might be more highly correlated to the generalization error.

Now, let us show how α and β are calculated in the VBBR implementation. If we assume that α and β are random variables, then Bayes' rule to optimize these two parameters, conditional on the given validation data D_v has the form

$$P(\alpha, \beta | D_v) = \frac{P(D_v | \alpha, \beta) P(\alpha, \beta)}{P(D_v)}. \tag{142}$$

As we explained in Chapter 3, the following proportional relation can be identified between the two probabilistic quantities

$$P(\alpha, \beta | D_v) \propto P(D_v | \alpha, \beta), \tag{143}$$

where $P(\alpha, \beta | D_v)$ is the conditional posterior density function, and $P(D_v | \alpha, \beta)$ is the validation evidence for α and β. Eq. (143) says that optimizing the posterior probability of α and β is equivalent to maximizing the validation evidence $P(D_v | \alpha, \beta)$. Substituting the validation data D_v for the training data D_t in Eq. (140), we have validation evidence expressed as:

$$P(D_v | \alpha, \beta) = \frac{P(D_v | \boldsymbol{w}, \beta) P(\boldsymbol{w} | \alpha)}{P(\boldsymbol{w} | D_v, \alpha, \beta)}. \tag{144}$$

In order to evaluate the validation evidence, we need to calculate the weight posterior density $P(\boldsymbol{w} | D_v, \alpha, \beta)$ and the likelihood function $P(D_v | \boldsymbol{w}, \beta)$ based on the validation data. Instead of retraining the validation data to get these quantities, we assume that the weight posterior probabilities of the training and validation sets are both Gaussian distributed and centered at the same most probable weight vector w_{MP}. The uncertainties $P(D_v|w,\beta)$ and $P(w|D_v,\alpha,\beta)$ therefore can be approximately computed by substituting validation data (denoted by subscript v) into the corresponding equations for calculating the uncertainties of training data which were given in Chapter 3. The generalized likelihood function of the validation set can be expressed as

$$P(D_v|w, \beta) = \frac{1}{Z_{D_v}(\beta)}\exp(-\beta E_v^D), \tag{145}$$

where $E_v^D = e_v^T(w)e_v(w)$, and $Z_{D_v}(\beta) = (\pi/\beta)^{n_v/2}$. The uncertainty of the posterior weights measured on the validation set has the form

$$P(w|D_v, \alpha, \beta) = \frac{1}{Z_{F_v}(\alpha, \beta)}\exp(-F_v) , \tag{146}$$

where $F_v = \beta E_v^D + \alpha\Omega(w)$, and $Z_{F_v} = (2\pi)^{N/2}(\det(H_v^{-1}))^{1/2}\exp(-F_v(w_{MP}))$.

Note that the weight prior density function $P(w|\alpha)$ in Eq. (144) is same for both the training and validation sets. As given in Eq. (54) of Chapter 3,

$$P(w|\alpha) = \frac{1}{Z_W(\alpha)}\exp(-\alpha\Omega(w)),$$

with $\Omega(w) = w^T w$, and $Z_W(\alpha) = (\pi/\alpha)^{N/2}$. Substituting $P(D_v|w,\beta)$, $P(w|\alpha)$ and $P(w|D_v,\alpha,\beta)$ into Eq. (144), we have

$$P(D_v|\alpha, \beta) = \frac{Z_{F_v}(\alpha , \beta)}{Z_{D_v}(\beta)Z_W(\alpha)} = \frac{(2\pi)^{N/2}(\det(H_v^{-1}))^{1/2}\exp(-F_v(w_{MP}))}{(\pi/\beta)^{n_v/2}(\pi/\alpha)^{N/2}}. \tag{147}$$

If we take the log of Eq. (147), we obtain

$$\log P(D_v|\alpha, \beta) = \frac{N}{2}\log(2\pi) - \frac{1}{2}\log\det(H_v) - F_v(w_{MP}) - \frac{n_v}{2}\left(\frac{\pi}{\beta}\right) - \frac{N}{2}\left(\frac{\pi}{\alpha}\right) . \quad (148)$$

The optimal α and β which maximize the validation evidence can be determined by seeking the extreme value of Eq. (148). The solutions are given below:

$$\alpha^{MP} = \frac{\gamma_v}{2\Omega(w^{MP})} \quad (149)$$

$$\beta^{MP} = \frac{n_v - \gamma_v}{2E_v^D(w^{MP})} \quad (150)$$

where γ_v is the effective number of parameters assessed from validation data.

$$\gamma_v = N - \alpha \mathrm{tr}((H_v^{MP})^{-1}) \quad (151)$$

Comparing the final equations obtained from the validation evidence framework with the corresponding equations from the training evidence framework, we can see that they have the exactly same form. However, the training evidence is a straightforward measure following a hierarchical fashion. It is evaluated on the same data as that used to determine the network weights. In contrast, the validation evidence is a generalized comprehensive measure made on the validation data, which is disjoint to the training data, to evaluate the validity of the training probabilistic model about the weight and error distributions. The return values based on this evaluation are α and β, which maximize the generalized validation evidence. They produce constraints on the new training performance function.

Using the VBBR algorithm, α and β are adapted in each iteration. Initially, we can set β to be 1 and α to be a small value (e.g., 0.1). The change of their trajectory is usually fast during the early stage of training, and gradually becomes slow without significant oscillation as the training continues. After the training converges to the required accuracy, the optimal values of α and β are saved. The combined training (using both training and validation data) then will be performed to refine the estimation of the network parameters using the fixed α and β.

Bayesian Regularization with Retrained Early Stopping (BRES)

Now let us turn to another implementation of validation-incorporated Bayesian learning which is called Bayesian regularization with retrained early stopping (BRES). As with the VBBR algorithm, the approach of BRES also employs a two-stage learning method, and uses a disjoint training set and validation set in the first stage. However, unlike the VBBR algorithm, network learning for BRES starts with the standard Bayesian method to alternately update the network weights and regularization parameters using the training data. The model validity, measured with sum-of-squared error on the validation data, is checked at each iteration. The first stage training of BRES is terminated when the validation error increases. Then, it is followed by a second stage retraining, as in the VBBR algorithm, using a combined data set and the fixed α and β obtained from the early stopping point.

It can be seen that the BRES implementation also has some features in common with the previously discussed retrained early stopping (RTES). Both of them monitor validation error to determine the optimal stop point of the first stage training, and perform a retraining using all of the data. However, several distinctions can be identified between these two implementations. In the RTES algorithm, conventional cross-validated early stopping is applied to the unregularized performance function during the first stage. In the BRES method, the early stopping procedure is applied to the regularized performance function. For the unregularized performance function, the increase of validation error simply means that further training may learn the noise, or overfit the data. For the regularized performance function, the increase of validation error indicates that the specific model assumptions for the weight and error distribution may no longer apply if further training follows the Bayesian framework. In the retraining stage, the RTES algorithm needs to specify the error goal and the weight goal to control the final model complexity. The BRES method can use the GPE criterion to determine the stop point, since the effective number of parameters can be calculated directly from the Bayesian framework.

Now we can summarize what we have proposed so far in this chapter. In the standard Bayesian method, the training is always performed to capture the specified prior beliefs. The model validity is hard to justify when all available data are used to produce the network. In order to reduce the potential risk of overfitting, we implemented several new approaches using validation-incorporated Bayesian learning. The GPBR and BRES methods presented new ways to determine the early stopping points which infer the presence of limitations in models if the further training is performed. The VBBR algorithm uses validation evidence to update regularization parameters which might result in better generalization performance than using training evidence. Note that we employed a two-stage training method in both VBBR and BRES implementations. To avoid the overuse of Bayesian learning, the second stage training does not follow the standard two-level hierarchical procedure. The performance function with optimal regularization parameters remains unchanged to the end. Since the retraining combines previous training data and validation data, we are able to use all available data, but in a different way than the standard Bayesian method.

In the next section, we will compare these new proposed approaches with the standard Bayesian framework through simulation experiments.

Simulations and Discussions

In this section, we will show the testing results of the validation-incorporated Bayesian learning algorithms on the same example that we used in Chapter 6, and will compare them with those obtained using the GNBR (Gaussian Newton Approximation to Bayesian Learning) algorithm [Fore96][FoHa97], which has been implemented in the Matlab Neural Network Toolbox.

The algorithms under testing competed in parabolic function approximation with noisy data employing a sufficiently complicated neural network structure. The results are compared under three pattern / parameter ratios (PPR = 961 / 151, 441 / 151, 225 / 151) and four noise levels($\sigma = 0.3, 0.2, 0.1, 0.05$). The data splitting ratio is 1 / 1 for both VBBR and BRES methods at the first training stage. The performance comparison is summarized in Table 18. In which, each performance entry is averaged over 30 trials with different initial weighs and random noise realizations, as we explained in Chapter 6.

PPR	Training Algorithm	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.3$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.2$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.1$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.05$)
961 / 151	GNBR	49.01	20.47	6.60	1.81
	GPBR	34.48	16.35	5.02	1.44
	VBBR	28.72	15.64	4.71	1.36
	BRES	28.40	14.76	4.76	1.34
441 / 151	GNBR	84.26	40.80	13.79	4.17
	GPBR	62.97	30.51	8.97	2.62
	VBBR	58.54	27.75	7.89	2.35
	BRES	57.57	27.67	7.28	2.47
225 / 151	GNBR	138.79	63.52	23.31	7.52
	GPBR	112.71	50.39	15.26	4.69
	VBBR	95.96	44.31	13.49	4.23
	BRES	99.23	46.69	13.66	4.26

Table 18 Generalization Performance with Variations of Bayesian Learning

PPR	Training Algorithm	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.3$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.2$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.1$)	$\overline{\text{MSE}} \times 10^{-4}$ ($\sigma = 0.05$)
961 / 151	GNBR	28.93	5.17	1.96	0.55
	GPBR	14.02	3.84	1.14	0.38
	VBBR	9.21	3.85	1.13	0.34
	BRES	8.22	3.57	1.11	0.37
441 / 151	GNBR	50.59	20.58	5.95	1.51
	GPBR	22.55	8.92	3.07	0.91
	VBBR	19.16	8.14	1.72	0.75
	BRES	19.89	7.47	1.59	0.77
225 / 151	GNBR	58.22	27.15	12.74	2.27
	GPBR	30.71	14.49	5.50	1.25
	VBBR	28.36	12.15	4.74	0.83
	BRES	26.98	13.44	4.33	0.81

Table 19 Standard Deviation in Performance with Variations of Bayesian Learning

Table 19 shows the standard deviation in generalization error over 30 trials. It can be concluded from these tables that VBBR and BRES methods produce the best results under each testing condition. The prediction error with the BBR-trained / BRES-trained model is 54 ~ 74 percent of the GNBR-trained model. Although GPBR results are not as good as VBBR and BRES results, they are still significantly better than those we obtained by using the standard GNBR algorithm (0.63 ~ 0.81 in error ratio). For the given problem, these simulation experiments demonstrate that the validation-incorporated Bayesian implementations proposed in this chapter have distinct advantages over the training-evidence-based Bayesian method.

PPR	Training Algorithm	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.3$)	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.2$)	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.1$)	$\overline{\text{flops}} \times 10^9$ ($\sigma = 0.05$)
961 / 151	GNBR	21.5	23.8	33.9	47.8
	GPBR	10.2	13.4	20.2	29.8
	VBBR	4.94	7.87	10.8	18.4
	BRES	3.84	6.34	11.2	15.6
441 / 151	GNBR	12.3	14.5	20.7	32.5
	GPBR	3.94	6.25	8.21	14.9
	VBBR	3.07	3.38	5.87	8.12
	BRES	1.98	3.04	6.75	7.99
225 / 151	GNBR	10.9	12.2	22.2	31.0
	GPBR	2.26	2.73	5.40	6.02
	VBBR	1.90	2.07	3.81	5.17
	BRES	1.13	1.64	3.73	4.99

Table 20 Computational Costs of Variations of Bayesian Learning

In Table 20, comparisons are made on the computational costs with different Bayesian implementations. We can see that computation efficiency can be dramatically improved when validation-incorporated Bayesian learning is used. The cost ratio of GNBR over VBBR (BRES) is between $3 \sim 9$. And the ratio is between $1.5 \sim 5$ for GNBR over GPBR.

Note that the GNBR results summarized in this chapter are different from the BR results presented in Chapter 6. In this chapter, training with GNBR is extended until convergence (SSE $< 10^{-6}$, or $\mu_W > 10^{10}$) is reached. While in Chapter 6, to make the training efficient, we heuristically terminated the learning process when the relative change of SSE within a sliding window was below a predetermined value(5.0×10^{-4}). The results presented in Chapter 6 using BR are better than GNBR results, but not as good as the validation-incorporated Bayesian implementations we proposed in this chapter.

One more comment should be made on the use of GPE-validated training. The GPE criterion is not just limited to the GPBR implementation. It can also be used to determine the stop point of the second stage training required by VBBR and BRES algorithms with fixed α and β. Even for the SDVR algorithm, the simulation experiments show that the GPE-validated retraining can result in better generalization performance. There might be a limitation for using GPE in noise free data. However, if optimal α and β are determined from the first stage learning, the monotonic decrease of the GPE in the second stage may infer the very low probability of overfitting. In this case, training can be terminated whenever the required mapping accuracy is reached.

CHAPTER 8

REAL-WORLD APPLICATIONS

Objectives

In this chapter, we will apply the newly developed methods to real-world problems, and compare them with some currently used methods. Five representative problems with realistic complexity are selected to test the algorithms. The relevant data analysis and pre-processing are also addressed in this chapter.

Introduction

In the previous chapters, we proposed several new methods to improve generalization performance of neural networks by optimal use of regularization and cross-validation techniques. The simulation experiments showed promising results for these innovative approaches.

In this chapter, we will test our new algorithms on several real-world problems. These problems represent a diverse cross-section of applications with a variety of complexity, and they will require the application of a variety of networks. The number of network parameters will vary from a few tens to several hundreds. Each of the problems has distinct noise levels.

Real-world applications are different from numerical simulations in several respects. First, in the simulation experiments the generalization performance is evaluated by the error between the model predictions and the true function outputs. In real-world applications, the true function is unknown. The generalization performance is estimated by the error on a test set, which is not used for training. This estimate can be quite noisy. Therefore, the model with smallest test error on a specific data set may not be the model with smallest generalization error. Secondly, the data availability is usually not a problem in simulation experiments. The training examples can be appropriately sampled over the input/output space. In real-world applications, the information provided by the finite data set is often limited. The model prediction credibility is seriously affected by the sparsity of data. This increases the difficulty of algorithm comparison, which is based on the test error. To make a reasonable comparison, we will calculate the test error in different ways. When a large number of examples are available, the test error will be measured from a fixed large testing set, and averaged over several trials with different initial weights. When data are limited, however, the test error will be averaged over several trials with different data splittings.

Although the objective of this chapter is to test different algorithms through real-world applications, the relevant data analysis and preprocessing are also discussed. Each example presented in this chapter can be considered as a case study.

Soft Sensors for Diesel Engines

This example is selected from the OSU Report to Cummins Engine Company [ChMa97]. The emission levels of carbon monoxide, unburned hydrocarbons, and nitrogen oxides from diesel engines are restrictively legislated with the increasing concern about environmental pollution. Although the emissions can be measured in the raw undiluted exhaust, the sampling and analysis process can be expensive for some cases. In this example, we use a neural network as a soft sensor to indirectly compute nitrogen oxide (NOx) emissions from measurements of speed and fueling, which can be obtained more easily and inexpensively. The data were collected from a diesel engine, which was operated over a transient test cycle for two minutes. The transient cycle involves steady states, accelerations, decelerations and overrun conditions which are supposed to represent engine maneuvers in an urban environment.

Figure 23 shows part of the sample sequences of the speed and fueling with a sampling interval of one second. The speed and fueling in the transient cycle change frequently, and the time series data appear nonstationary. Since the frequent shift of speed and fueling will change the state of flame diffusion in the combustion chamber, the emission level of the engine at a given instant can be modeled as a function of the current and previous measurements of the speed and fueling.

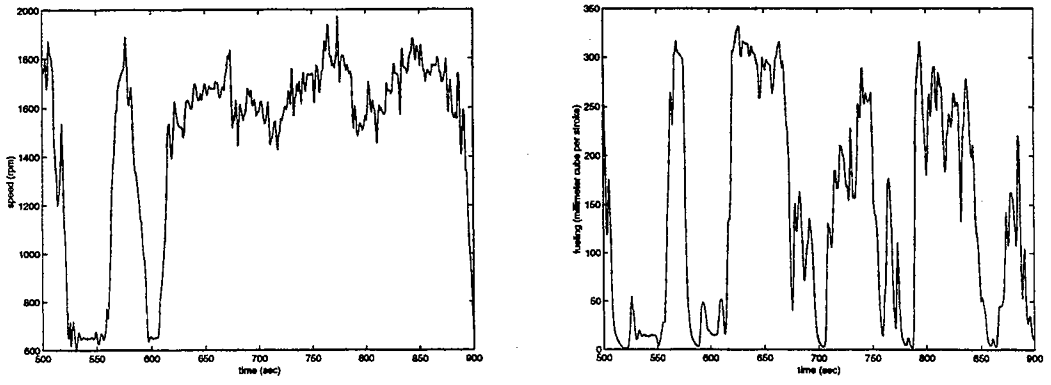


Figure 23 Speed and Fueling in Transient Cycle

In this application, the NOx prediction system is represented by a four-input / single-output neural network model which is defined in Figure 24.

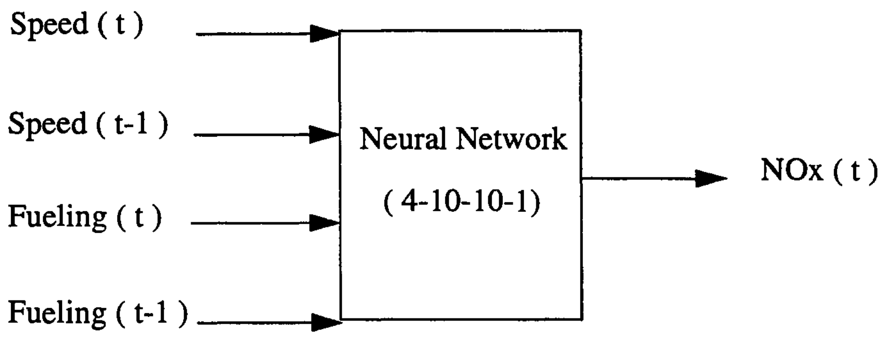


Figure 24 NOx prediction Model

The total input/output patterns are randomly resampled five times. In each resampling, the available data are divided into the training, validation, and testing groups. The data splitting ratio (training size / validation size / testing size) is about 3 / 3 / 2. Figure 25 presents partial measurements of the nitrogen oxide, which are used as training targets, and the NOx prediction using a trained neural network model. Although using speed and fueling alone to predict emission levels is a simplified approach, the neural network prediction is reasonably accurate. The obvious prediction errors occurred at the time instants when the measured NOx is zero. Apparently, the sensor was unable to register NOx levels below approximately 200, therefore any emission below this level registered as zero. It is difficult for the neural network model to capture this kind of nonlinearity without overfitting the data in other regions.

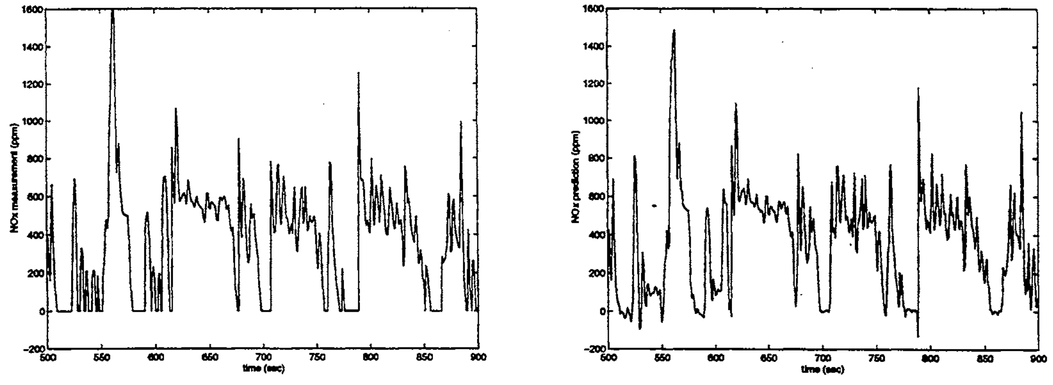


Figure 25 NOx Measurement and Prediction

The testing results used for algorithm comparison are averaged over five trials. Each trial is run under different data resampling. Table 21 presents the statistical parameters obtained from the error measurements. It is observed from this example that the testing error of the neural network models trained with the GNBR algorithm is larger than the testing error of the models trained with the other algorithms. This might be a case which shows the limitation of the Bayesian method when the assumptions about the noise and the weight distribution do not apply as the network is trained to convergence. The other methods performed almost equally well indicating that the nature of the data is easy to capture by using either active validation or passive validation.

Algorithm	S1	S2	NumParam	MSE $\times 10^{-2}$	STD $\times 10^{-2}$
GNBR	10	10	171	1.68	0.42
ES	10	10	171	1.31	0.33
SDVR	10	10	171	1.26	0.37
VBBR	10	10	171	1.29	0.39
BRES	10	10	171	1.32	0.42
RTES	10	10	171	1.22	0.32

Table 21 Testing Results on Engine Data

Prediction of Chaotic Intensity Pulsation of an NH_3 Laser

This example is from the Santa Fe Institute Time Series Prediction and Analysis Competition [Wan93]. The chaotic intensity pulsation of an NH_3 laser was distributed as part of the competition. The contestants were given only 1000 points of data, which are shown in Figure 26, and invited to send in solutions predicting the next 100 points. This is a long-term time series prediction problem, and the network structure will consist of a tapped delay line preceding a multilayer network.

For this problem, we assume that the time series $y(n)$ will be modelled as a nonlinear function of its past values, i.e.,

$$y(n) = f(y(n-1), y(n-2), \dots y(n-m)) + e(n), \qquad (152)$$

where $e(n)$ is prediction error, and m is the order of tapped delay, which is also the input dimension of the neural network. Selection of the input dimension was based mostly on trial-and-error along with various heuristics. In this example, a high-order tapped delay ($m = 35$) was used.

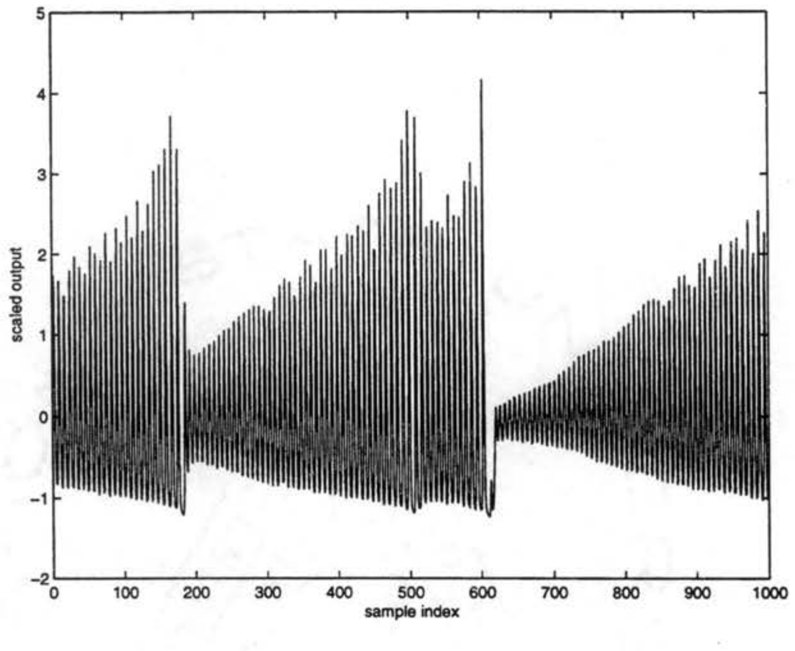


Figure 26 Chaotic Intensity Pulsation of an NH_3 Laser

To model the given time series, the original data were first scaled to zero mean, unit variance. Since the true task involved long-term prediction, we chose the last 100 points of the given 1000 samples as the validation set. A three-layer 35-12-12-1network with 601 parameters was selected to approximate the underlying function of the time series. Five trials with different initial weights were run by using each training algorithm. The error minimized in the training objective function was based on the one-step prediction.

After the training was completed, the generated model was used to make the iterative prediction, in which the previous predicted values were put into the input vector to forecast future output. This process was repeated several times. After 35 iterated predictions, no true sample of the time series would be included in the input filter. The whole tapped delay line would be filled with forecasted data!

The 100-point predictions obtained in this way then were compared with withheld measurements to compute the testing error. The predictions from two trials and the measurements are plotted in Figure 27. It was clear from the nature of the data that predicting downward intensity collapses would be the most important and difficult aspect of the series to learn. The prediction accuracy can be as good as that shown in the left plot of Figure 27 if the network weights are appropriately determined. Also, if overfitting occurs, the prediction accuracy can be much worse than that shown in the right plot of Figure 27.

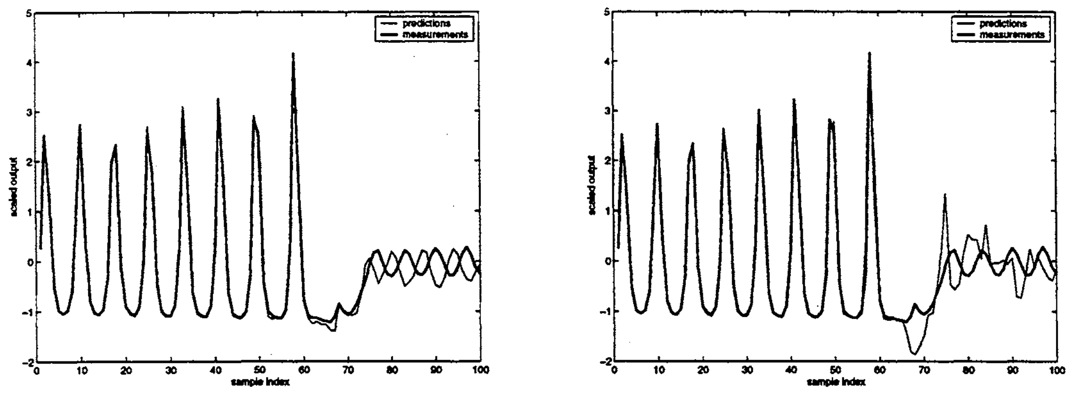


Figure 27 Iterative Long-Term Prediction for NH_3 Laser Data

The testing results with different algorithms are given in Table 22. Apparently, the variation of the prediction accuracy among the candidate algorithms is large due to the difficulty to forecast the downward intensity collapses. It can be concluded from the table that using the validation-set-based VBBR and SDVR algorithms to determine the optimal regularization parameter is very effective in reducing the generalization error in this time series example. While training with the GNBR algorithm to high accuracy on the whole data set may risk overfitting on unseen data.

Algorithm	S1	S2	NumParam	MSE $\times 10^{-1}$	STD $\times 10^{-1}$
GNBR	12	12	601	3.17	3.90
ES	12	12	601	1.64	0.80
SDVR	12	12	601	0.70	0.35
VBBR	12	12	601	0.59	0.49
BRES	12	12	601	2.03	1.88
RTES	12	12	601	1.64	1.01

Table 22 Testing Results on NH_3 Laser Data

Neural Network Inversion of Induction Logs

This application reflects our current research on well logging analysis [ChHa98b]. Formation resistivity is one of the primary formation parameters that is highly relevant to the presence of oil and/or gas. Modeling formation resistivity has been widely used as a means to design electrical logging tools, to predict their responses and to extract resistivities and structural geometries from resistivity logs. The complexity of the problem can be represented by the 1D, 2D and even 3D models, depending on the versatility of the formation geometries.

In this application, we will use a neural network model to determine the relationship between the induction logging tool response and the formation resistivity. Induction logging uses a bank of coils to make resistivity measurements. Some of these coils, referred to as the transmitter coils, are energized by an alternating current at a frequency between 8 and 40 kHz. The oscillating magnetic field produced by this arrangement results in the induction of currents in the earth formations which are nearly proportional to the conductivity (reciprocal of the resistivity). These currents in turn contribute to the voltage induced in the receiver coils. The receiver responses, the real signal and the imaginary signal, are usually used in signal processing. The real signal is the portion of the received voltage that is in phase with the transmitter current, and the imaginary signal is the portion 90 degrees out of phase with the transmitter current. In general, the receiver response, which is also called the tool response, is a nonlinear function of the formation resistivity. Since the imaginary signal is too noisy and not reliable, we often only use the measured real signal as the induction logging tool response.

The traditional method of modeling formation resistivity is usually used to predict the tool response. In this example, we have an inverse modeling problem, in which the tool responses are used as the inputs to the model, and the model outputs will approximate the formation resistivities. We assume that the formation geometry is one dimensional and horizontal, i.e., the resistivity changes only along the depth of the borehole (without radial variation).

In the previous example, the current value of a time series can be predicted with its past values. In this example, the resistivity at the certain position in depth is modelled as a function of the tool responses both below and above that position. We use a filter input window which is centered at the same depth index as that of the bed resistivity to be predicted. The window length is 25 feet, and is stretched evenly above and below the index point. The window symmetry may not be necessary, the motivation of this design is simplicity.

The tool response/resistivity profiles used in developing the neural network model are subsets of typical Oklahoma profiles generated from a single receiver with 8 kHz excitation frequency. One example of an Oklahoma profile is shown in Figure 28.

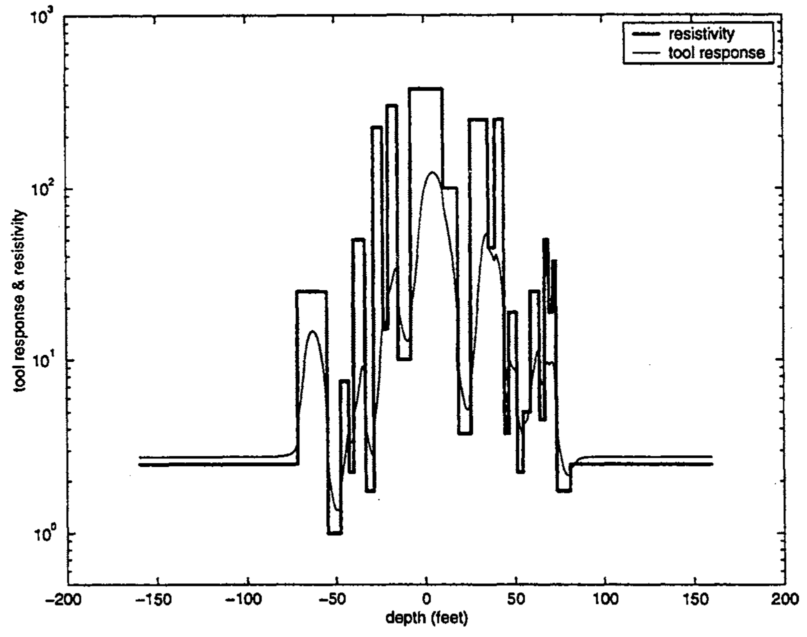


Figure 28 Formation Resistivity and Tool Response of Oklahoma Profile

We can see from Figure 28 that the relation between the tool response and the resistivity is nonlinear. In this example, several Oklahoma formation subsets with similar shapes but different bed values are available. We use 8 subsets (3928 patterns) for training, and 4 subsets (1964 patterns) for validation. The samples are collected at a 0.5 feet interval. For the 25-foot window, the input dimensionality is 51!

Since the values of tool responses and resistivities cover a wide range, a logarithmic transformation is made first on both inputs and targets. Then we train a 51-10-20-1 network with each algorithm by using the normalized log data ranging from -1 to +1. After that, a large number of novel patterns with a variety of formation shapes are used to test the applicability of the neural network inversion of induction logs. Figure 29 presents some of the testing profiles.

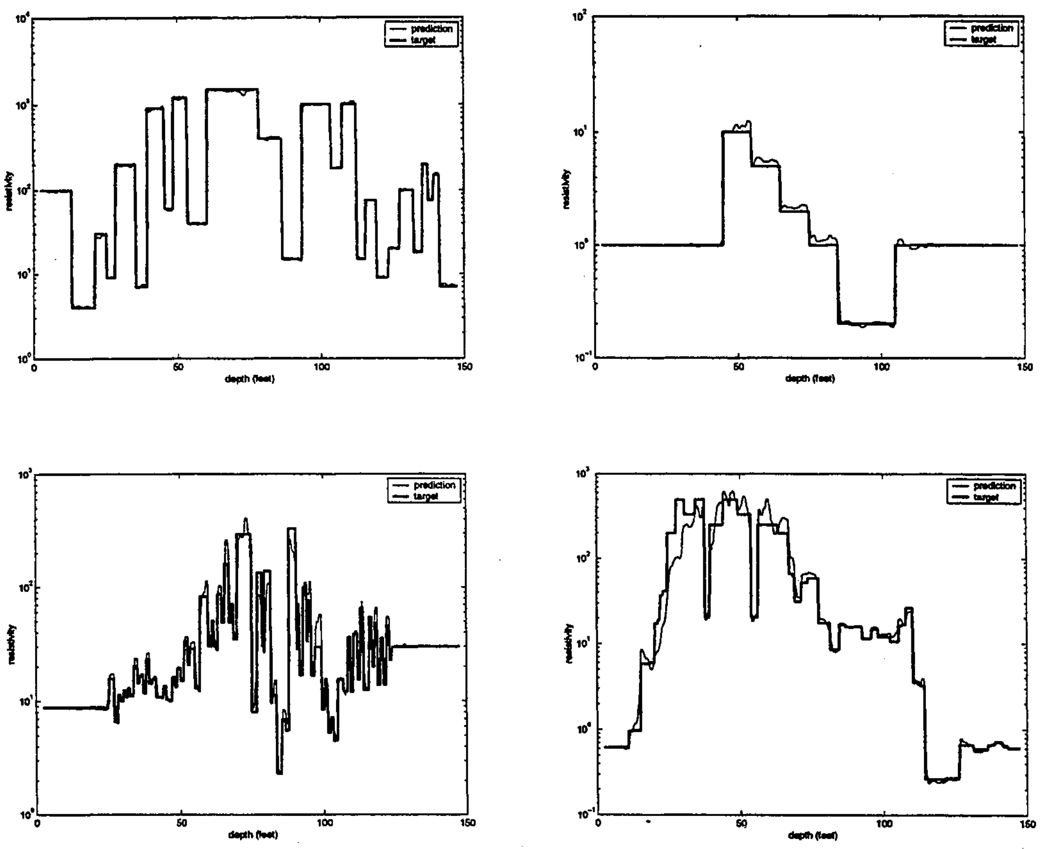


Figure 29 Neural Network Inversion of Induction Logs

For the Oklahoma like profile, the prediction is very accurate. Even for the totally different profiles, the neural network inversion also gives reasonably good prediction. The algorithm comparison based on the testing results are given in Table 23.

Algorithm	S1	S2	NumParam	MSLE $\times 10^{-2}$
GNBR	10	20	761	8.24
ES	10	20	761	11.67
SDVR	10	20	761	8.02
VBBR	10	20	761	8.74
BRES	10	20	761	9.40
RTES	10	20	761	10.04

Table 23 Testing Results on Induction Logs

Note that the testing performance measured in Table 23 is MSLE, which is mean squared log error averaged over about 1200 testing patterns. In this single trial, the SDVR algorithm wins the competition, followed by the GNBR and VBBR algorithms. The expensive computation cost for this example limits the multiple trials with different initial conditions. However, since the data are noiseless, and the pattern/parameter ratio is high, low variation in testing results with further experiments is expected.

Environmental Corrections for Neutron Tools

For this example, a different type of well logging tool, called the neutron tool, is used to determine the formation parameters. While the responses of the induction logging tool are highly correlated with formation resistivity, the responses of the neutron tool are highly correlated with formation porosity. The neutron logging tool comprises a fast neutron source, shielding materials, and two thermal neutron detectors. The fast neutrons spread out and slow down by elastic scattering, approach thermal equilibrium, diffuse at thermal energy, and are absorbed by nuclei in the borehole/formation environment. During the thermal diffusion process, some neutrons are detected by either the near or the far detector, and count rates are developed that become the primary logging input variables.

Since the neutron tool responses are sensitive to several environmental factors, the environmental corrections must be performed in order to make the porosity prediction accurate. However, current techniques for environmental corrections involve applying each correction sequentially and independently of other corrections. The neural network models will be designed to apply all corrections simultaneously, accounting for correction interactions. For this example, we will train a seven-input/single-output model. The model output is the formation porosity. The seven input variables are:

1. Log of the near count rate recorded by short spaced neutron detector;

2. Log of the far count rate recorded by long spaced neutron detector;

3. Ratio of near count rate to far count rate;

4. Formation type (1 for limestone and 0 for sandstone);

5. Matrix absorption cross section, or sigma matrix (capture unit);

6. Borehole diameter (inch);

7. Standoff, which is a measure of lost density pad contact with formation (inch).

The main problem of this task is that only very limited data are available, because making more direct experimental observations is expensive [ChHa98a]. Totally, 222 patterns are provided for model development. Before splitting these samples, we first divide the whole data set into two groups. One group comprises 144 patterns on the boundary of the input space. The other consists of 78 patterns which are located within the range bounded by the patterns in the first group. All of the boundary patterns are included in the training data set. The interior patterns are used to form the validation and test sets, and some of them also form a portion of the training set.

The data are divided into the training, validation and test sets. We randomly pick 39 patterns from the interior group to construct a test set. For the remaining 39 patterns in the interior group, 19 of them are randomly selected to form the validation set. The other 20 patterns are combined with 144 boundary patterns to form the training set. The same procedure is repeated five times to produce five different data splittings. The inputs and the targets are scaled from -1 to +1 before entering the network.

The representative model designed to predict the formation porosity is a two-layer, 7-10-1 network with 91 parameters. One testing example using the neutron tool response as input with environment correction is shown in Figure 30. Since the measurement noise is low, the prediction is accurate for this application.

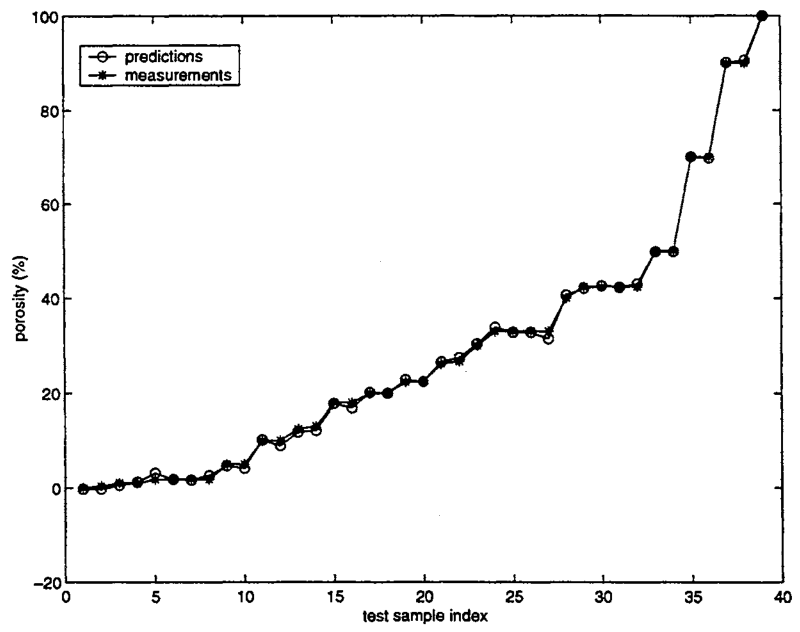


Figure 30 Porosity Prediction Using Corrected Neutron Tool Response

The test results averaged over five trials are summarized in Table 24.

Algorithm	S1	NumParam	MSE $\times 10^{-4}$	STD $\times 10^{-4}$
GNBR	10	91	3.12	0.96
ES	10	91	4.81	1.16
SDVR	10	91	4.13	1.98
VBBR	10	91	3.81	0.85
BRES	10	91	3.35	1.00
RTES	10	91	4.43	1.21

Table 24 Testing Results on Neutron Logs

In this application, the results show that the GNBR algorithm generalizes best with sparse data. The share of the data using cross-validated early stopping (ES) will worsen the existing problems due to the very limited availability of the training data. The other algorithms predict better than the ES algorithm, because the combined training with more samples are finally performed to improve the parameter estimation.

Determination of Cholesterol Levels from Blood Spectral Contents

This is a medical application example [PuLu92]. We are going to design a network which can determine serum cholesterol levels from measurements of spectral content of a blood sample. We have total of 264 patients for which the measurements of 21 wavelengths and the Hdl, Ldl cholesterol levels are collected. This example can be considered as a multiple-input, multiple-output modeling problem, in which the network inputs are 21wavelengths of a blood spectral content, and the network outputs are the Hdl and Ldl cholesterol levels. Figure 31 displays the Hdl and Ldl measurements of patients.

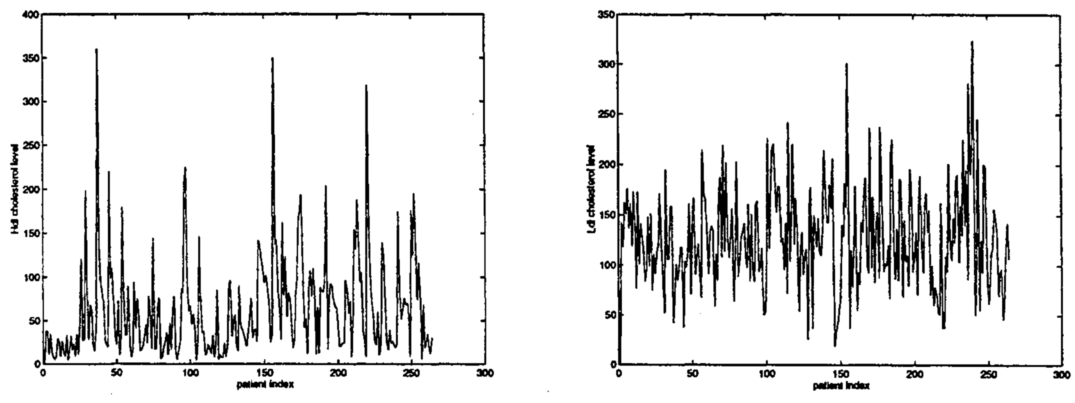


Figure 31 Collected Cholesterol Levels from 264 Patients

To determine the input/output relationships, we use a two-layer network structure with 6 hidden neurons. Five random data splittings with different sample groupings are made. In each splitting, the samples from 44 patients are withheld for the testing, and the samples from 189 and 31 patients are used for the training and validation. The experiments show that the measurements are very noisy in this example. Increasing the number of hidden neurons has little effect on prediction. Figure 32 illustrates how good the prediction can be based on a single testing set.

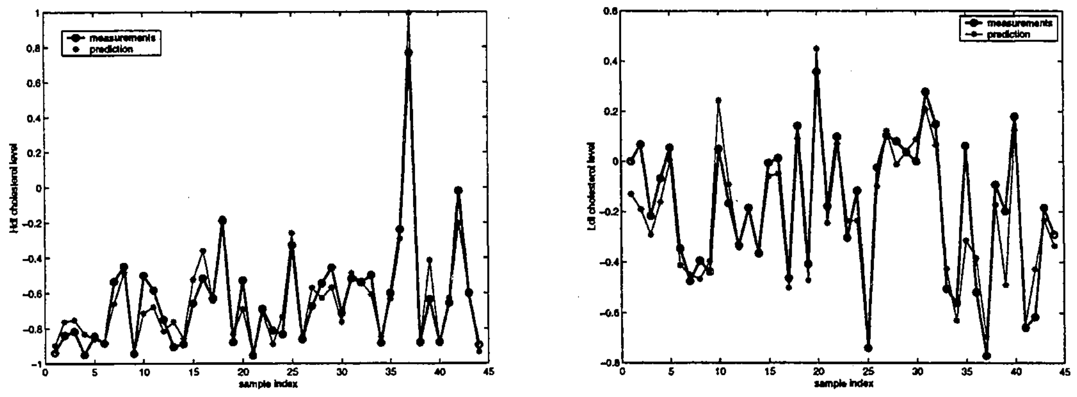


Figure 32 Neural Network Prediction of Cholesterol Levels

The testing results with different training algorithms are summarized in Table 25. Clearly, the newly developed four algorithms work better than the GNBR and ES algorithms with the smaller mean-squared errors and standard deviations.

Algorithm	S1	NumParam	MSE $\times 10^{-2}$	STD $\times 10^{-2}$
GNBR	6	146	3.82	1.82
ES	6	146	2.20	0.77
SDVR	6	146	1.65	0.30
VBBR	6	146	1.90	0.42
BRES	6	146	1.42	0.27
RTES	6	146	2.00	0.63

Table 25 Testing Results on Cholesterol Data

We have already tested six candidate training algorithms on the five real-world problems. The place of each algorithm in the rank of each application is presented in Table 26, where the winner is ranked as the first place.

	Engine	Laser	Induction	Neutron	Cholesterol
GNBR	6	6	2	1	6
ES	4	3	6	6	5
SDVR	2	2	1	4	2
VBBR	3	1	3	3	3
BRES	5	5	4	2	1
RTES	1	4	5	5	4

Table 26 Place in Rank for Real-World Applications

It is interesting to note that the five first places are distributed among the five different algorithms. In order to determine an 'all-round champion', or to quantitatively evaluate the comparison results, we assign points to each place in the competition. First place receives 6 points. The number of points will be reduced by one as the corresponding rank is reduced. In this way, the sixth place will get 1 point. The score in each application and the accumulated points for each algorithm are listed in Table 27.

	Engine	Laser	Induction	Neutron	Choleterol	Σ
GNBR	1	1	5	6	1	14
ES	3	4	1	1	2	11
SDVR	5	5	6	3	5	24
VBBR	4	6	4	4	4	22
BRES	2	2	3	5	6	18
RTES	6	3	2	2	3	16

Table 27 Points Obtained in Real-World Applications

According to this summary, the two new validation-set-based regularization algorithms, SDVR and VBBR, generated the networks which produce the best testing results in real-world applications. Also, the other two new methods, BRES and RTES, demonstrated advantages over the standard GNBR and ES algorithms. The GNBR algorithm performed better than early stopping. These results are consistent with the numerical simulation results presented in the previous chapters.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

Objectives

In this chapter, we present a summary of the research completed, and recommend possible future work drawn from this dissertation.

Summary

The objective of this research has been to effectively improve the generalization performance of neural networks in solving nonlinear regression or function approximation problems. It has been shown that multilayer feedforward neural networks can be used to represent any arbitrary continuous function. Many current training methods are able to produce networks with accurate input/output mappings for the training examples, but may not be satisfactory in generalizing to new data, due to the problem of overfitting. Improving generalization is a major area in neural network research. A variety of different techniques have been developed.

Among the different techniques for improving generalization performance, we have investigated two key methods: regularization and cross-validated early stopping. Training with early stopping is computationally efficient, but only partial data can be used to update network weights, and the resulting models usually show a large variance in generalization error in the presence of noise. Bayesian regularization works well in many situations, but the quality of the produced model depends on whether or not the probabilistic assumptions match the training data in a statistically adequate manner. In order to generate neural network models with better generalization, we are interested in techniques which incorporate adaptive regularization with cross-validation, and refine parameter estimation after early stopping.

One of the main contributions of this dissertation research is the development of the validation-set-based regularization algorithms which combine regularization with the active use of the validation information. We made two innovations which can be applied to

the standard regularization networks and the Bayesian probabilistic networks, respectively. First, In Chapter 5, the SDVR (second derivative of validation error regularization) algorithm was derived to automatically update the regularization parameter by minimizing the validation error. To do this, the algorithm uses second order information to set the optimal learning rate for the regularization parameter. Second, in Chapter 7, the VBBR (validation-set-based Bayesian regularization) algorithm was implemented to calculate the Bayesian regularization parameter by maximizing the validation evidence. The evidence framework of the validation data set is approximately represented in the same Bayesian form as the evidence framework of the training data set. Although the validation error and the validation evidence have different meanings, both are more directly related to the generalization error. We have shown from Chapter 5 to Chapter 8 that the SDVR and VBBR algorithms produce networks with excellent generalization capability.

The other main contribution of this work is the RTES (retrained early stopping) procedure. This procedure retrains a network using the combined data set after early stopping, and therefore overcomes the limitation of conventional early stopping which uses only part of data for parameter estimation. We have shown that Moody's GPE (generalized prediction error), which was developed for regularization, can be functionally reduced by using the RTES if the error goal and weight goal are set appropriately. We have shown in Chapter 4, Chapter 6 and Chapter 8 that the networks produced by retrained early stopping generalize better than conventional early stopping.

As an extension of the RTES method to the Bayesian learning, we proposed and implemented the BRES (Bayesian regularization with early stopping) method in Chapter 7.

While the RTES approach is applied to the unregularized performance function, the BRES approach is applied to the regularized performance function. The BRES algorithm has the advantages of regularization, passive validation, and retraining. In the retraining stage, the BRES algorithm does not need to specify the error goal and the weight goal to control the final model complexity. The Moody's GPE criterion can be used directly to determine the termination point. It has been shown that the BRES algorithm performed better than the RTES algorithm.

Another contribution of this work is the algorithm comparison through extensive simulations and real-world applications with respect to the generalization error. Neural network research should not only develop new techniques, but should also investigate the best techniques for certain types of problems. In numerical experiments, different algorithms were applied to the same function approximation problem under various pattern / parameter ratios, data splitting ratios, and noise levels. We highlighted the relative advantages and limitations for each method, and emphasized the importance of the data splitting ratio when the validation set is used. In real-world applications, five problems were selected for case studies. These problems represent realistic complexity and require the application of a variety of networks. The information provided by this work will assist neural network users in algorithm and model selection, data analysis and preprocessing.

According to the results obtained from the numerical experiments and the real-world applications, the newly developed methods produced networks which generalized better than the standard Bayesian method and early stopping procedure. The two-stage training strategy employed by these methods is effective in improving parameter estimation and is flexible in controlling training accuracy. These methods enrich the regularization and cross-validation techniques, provide network users with practical training tools, and show promise for many applications.

Recommendations for Future Work

The possible future work recommended here is related to the VBBR and SDVR algorithms, which are most promising for potential applications.

One thing which needs to be examined is the relationship between the validated VBBR assumption and the data splitting. In deriving the VBBR algorithm, we assumed that the weight posterior probabilities of the training and validation sets are centered at the same most probable weight vector w_{MP}. This is approximately true if the validation set is a good representative of the training set. In that case, the mean-squared error of the validation set and the training set will be about the same level. When improper data splitting is used, the assumption above may not hold, and the validation error may be considerably larger than the training error. Therefore, monitoring the validation error during the first-stage training by using the VBBR algorithm may help decide whether the data splitting needs to change. With the optimal data splitting, the validation error is expected to be well correlated with the validation evidence, and the model trained with the VBBR algorithm therefore would have good generalization. Since the size of the validation set is also important to the calculation of the effective number of parameters, examining these interactions would provide a better understanding of the VBBR algorithm.

For the SDVR algorithm, the future work concerns the computation cost. One property associated with the SDVR algorithm is the trajectory oscillation during training, especially when the regularization parameter is updated incrementally. We showed in Chapter 5 that the computation cost can be reduced by using a variable update interval for parameter α. It might also be possible to change the incremental coefficient of the tunable parameter μ_α to force a quick convergence as the oscillation is detected. Currently, the incremental coefficient of the tunable parameter μ_α is held constant within each variation of the SDVR algorithm for simplicity. Due to the complicated relationship between the weight update and the regularization update, whether these changes provide any advantages remains to be seen.

REFERENCES

[Akai69] H. Akaike, “Fitting autoregressive models for prediction,” *Annals of the Institute of Statistical Mathematics,* vol. 21, pp. 243-247.

[AmMu93] S. Amari and N. Murata, “Statistical theory of learning curves under entropic loss criterion,” *Neural Computation*, vol. 5, pp. 140-153, 1993.

[AmMu96] S. Amari, N. Murata, K.-R. Müller, M. Finke, and H. Yang, “Statistical theory of overtraining: Is cross-validation effective?,” in *Advances in Neural Information Processing Systems 8,* D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, eds., Cambridge, MA: MIT Press, 1996.

[AmMu97] S. Amari, N. Murata, K.-R. Müller, M. Finke, and H. Yang, “Asymptotic statistical theory of overtraining and cross-validation,” *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp985-996, 1997

[BuWe91] W. L. Buntine and A. S. Weigend, “Bayesian back-propagation,” *Complex Systems*, vol. 5, pp. 603-643, 1991.

[Bish95] C. M. Bishop, *Neural Networks for Pattern Recognition*, New York: Oxford University Press, Inc., 1995.

[ChHa97] D. Chen and M. T. Hagan, “Soft sensors for diesel engines,” *OSU Report to Cummins Engine Company,* Inc., 1997.

[ChHa98a] D. Chen and M. T. Hagan, “Environmental corrections for neutron tools,” *OSU Report to Halliburton Energy Services*, 1998.

[ChHa98b] D. Chen and M. T. Hagan, “Neural network inversion of induction logs (phase I),” *OSU Report to Halliburton Energy Services*, 1998.

[ChHa99] D. Chen and M. T. Hagan, “Optimal use of regularization and cross-validation in neural network modeling,” *Proceedings of the International Joint Conference on Neural Networks (IJCNN99)*, paper no. 323, Washington, DC, July, 1999.

[CoGi95] M. Cottrell, B. Girard, Y. Girard, M. Mangeas, and C. Muller, “Neural modeling for time series: a statistical stepwise method for weight elimination,” *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 303-314, 1995.

[CuDe90] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, ed., pp. 598-605, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.

[Cybe89] G. Cybenko, "Approximations by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303-314, 1989.

[DeSc83] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[Flet87] R. Fletcher, *Practical Methods of Optimization* (Second ed.), New York: John Wiley, 1987.

[Fore96] F. D. Foresee, *Generalization and Neural Networks*, Ph.D. Thesis, Oklahoma State University, 1996.

[FoHa97] F. D. Foresee and M. T. Hagan, "Gaussian-Newton approximation to Bayesian learning," *Proceedings of the IEEE International Conference on Neural Neworks (ICNN'97)*, vol. 3, pp. 1930-1935, Houston, Texas, June, 1997.

[Funa89] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183-192, 1989.

[Gill81] P. E., Gill, W. Murray, and M. H. Wright, *Practical Optimization*, London: Acdemic Press, 1981

[Gout97] C. Goutte, "Note on free lunches and cross-validation," *Neural Computation*,vol. 9, pp. 2111-1215, 1997.

[HaDe96] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*, Boston, MA: PWS Publishing, 1996.

[HaMe94] M. T. Hagan and M. Menhaj, "Training multilayer networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, 1994, pp. 989-993.

[HaSt92] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: optimal brain surgeon," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson et al. eds., pp. 164-171, San Mateo, CA: Morgan Kaufmann Publishers, 1992.

[HeKr91] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Redwood City, CA: Addison Wesley, 1991.

[Horn91] K. Hornik, "Approximation capabilities of multilayer networks," *Neural Networks*, vol. 4, pp. 251-257, 1991.

[HoSt89] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks,* vol. 2, pp. 359-366, 1989.

[Kear97] M. Kearns, "A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split," *Neural Computation*, vol. 9, no. 5, pp. 1143-1161, 1997.

[KrHe92] A. Krogh and J. Hertz, "Generalization in a linear perceptron in the presence of noise," *J. Phys. A*. vol. 25, pp. 1135-1147, 1992.

[Lars93] J. Larsen, *Design of Neural Network Filters*, Ph.D. Thesis, Electronics Institute, Technical University of Denmark, 1993.

[LaHa96] J. Larsen, L. K. Hansen, C. Svarer, and M. ohlsson, "Design and regularization of neural networks: The optimal use of a validation set," in *Proceedinds of IEEE Workshop on Neural Networks for Signal Processing VI*, S. Usui, Y. Tohkura, S. Katagiri, and E. Wilson, eds., pp.62-71, Piscataway, NJ: IEEE, 1996.

[LaSv97] J. Larsen, C. Svarer, L. N. Andersen, and L. K. Hansen, "Adaptive regularization in neural network modeling," in *The Book of Tricks*, G. B. Orr et al., eds., Germany: Springer-Verlag, 1997.

[Ljun87] L. Ljung, *System Identification: Theory for the User*, Englewood Cliffs, NJ: Prentice-Hall,1987.

[Luen84] D. G. Luenberger, *Linear and Nonlinear Programming* (Second ed.), Reading, MA: Addison-Wesley, 1984.

[MacK92] D. J. C. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, pp. 415-447, 1992.

[MacK94] D. J. C. MacKay, "Bayesian methods for backpropagation networks," *Models of Neural Networks III*, E. Domany, J. L. van Hemmen, and K. Schulten, eds., pp. 211-254, New York: Springer-Verlag, 1994.

[MacK97] D. J. C. MacKay and R. Takeuchi, "Interpolation models with multiple hyperparameters," 1997, http://wol.ra.phy.cam.ac.uk/mackay/

[Mood92] J. E. Moody, "The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems," in *Advances in Neural Information Processing System* 4, J. E. Moody, S. J. Hanson and R. P. Lippmann, eds., pp. 847-854, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1992.

[MüFi96] K.-R. Müller, M. Finke, N. Murata, K. Schulten, and S. Amari, "A neumerical study on learning curves in stochastic multilayer feedforward networks," *Neural Computations*, vol. 8, pp. 1085-1106, 1996.

[MuYo94] N. Murata, S. Yoshizawa, and S. Amari, "Network information criterion: determining the number of hidden units for an artificial neural network model," *IEEE Transactions on Neural Networks*, vol. 5, pp. 865-872, 1994.

[Neal96] R. M. Neal, *Bayesian Learning for Neural Networks*, New York: Springer Verlag, 1996.

[Neum98] A. Neumaier, "Solving ill-conditioned and sigular linear systems: A tutorial on regularization," *SIAM Rev.*, vol. 40, no. 3, pp. 636-666, 1998.

[Pola71] E. Polak, *Computational Methods in Optimization: A Unified Approach*, New York: Academic Press, 1971.

[PuLu92] N. Purdie, E. A. Lucas and M. B. Talley, "Direct measure of total cholesterol and its distribution among major serum lipoproteins," *Clinical Chemistry*, vol. 38, no. 9, pp. 1645-1647, 1992.

[RuHi86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, 1986.

[RuMc86] D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, Cambridge, MA: MIT Press, 1986.

[SiDo91] J. Sietsma and R. J. F. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, pp. 67-79, 1991.

[Sjöb95] J. Sjöberg, *Non-Linear System Identification with Neural Networks*, Ph.D. Thesis, Department of Electrical Enginerring, Linköping University, Sweden, 1995.

[TiAr77] A. N. Tikhonov and V. Y. Arsenin, *Solution of Ill-Posed Problems*, Washington, DC: V. H. Winston & Sons, 1977.

[VoMa88] T. P. Vogl, J. K. Mangis, A. K. Zigler, W. T. Zink and D. L. Alkon, "Accelerating the convergence of the backpropagation method," *Biological Cybernetics*, vol. 59, pp. 256-264, 1988.

[Wan93] E. A. Wan, "Time series prediction by using a connectionist network with internal delay lines," in *Times Series Prediction: Forecasting the Future and Understanding the Past*, A. S. Weigend and N. A. Gershenfeld, eds., pp. 195-217, Addison-Wesley, 1993.

[WeRu91] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, “Generalization by weight-elimination applied to currency exchange rate prediction,” in *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, pp. 837-841, Piscataway, NJ: IEEE,1991.

[Will95] P. M. Williams, “Bayesian regularization and pruning using a Laplace prior,” *Neural Computation*, vol. 7, pp. 117-143, 1995.

[ZhRo96] H. Zhu and R. Rohwer, “No free lunch for cross-validation,” *Neural Computation*, vol. 8, pp. 1421-1426, 1996.

2

VITA

Dingding Chen

Candidate for the Degree of

Doctor of Philosophy

Thesis: OPTIMAL USE OF REGULARIZATION AND CROSS-VALIDATION IN NEURAL NETWORK MODELING

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Taiyuan, Shanxi, P. R. China, on December 12, 1956, the son of Wenyong Chen and Xigu Miao.

Education: Graduated from Fifth High School, Taiyuan, Shanxi, in August 1974; received Bachelor of Science degree in Mechanical Engineering from Beijing Agricultual Engineering University, Beijing, China, in May 1982; received Master of Science degree in Agricultural Engineering and Master of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma in May 1993 and May 1996, respectively. Completed the requirements for the Doctor of Philosophy degree in Electrical Engineering at Oklahoma State University in May 2000.

Experience: Employed by Shanxi Agricultural Mechanization Research Institute as a Research and Design Engineer in 1982; invited by Oklahoma State University, Department of Agricultural Engineering as a visiting scholar in 1990; employed by Oklahoma State University as a Research and Teaching Associate from 1991 to present.

Professional Status and Memberships: Institute of Electrical and Electronic Engineers, American Society of Agricultural Engineers.