1
Nonlinear Sequential State Estimation for Solving Pattern-Classification Problems

Simon Haykin and Ienkaran Arasaratnam
McMaster University
Canada

1.1
Introduction

Sequential state estimation has established itself as one of the essential elements of signal processing and control theory. Typically, we think of its use being confined to dynamic systems, where we are given a set of observables and the requirement is to estimate the hidden state of the system on which the observables are dependant. However, when the issue of interest is that of pattern-classification (recognition), we usually do not think of sequential estimation as a relevant tool for solving such problems. Perhaps, this oversight may be attributed to the fact that pattern-classification machines are usually viewed as static rather than dynamic systems. In this chapter, we take a different view:

Specifically, we look to nonlinear sequential state estimation as a tool for solving pattern-classification problems, where the problem is solved through an iterative supervised learning process.

In so doing, we demonstrate that this approach to solving pattern-classification problems offers several computational advantages compared to traditional methods, particularly when the problem of interest is a difficult one.

The chapter is structured as follows: Section II briefly discusses the back-propagation (BP) algorithm and support-vector machine (SVM) learning as two commonly used procedures for pastern classification; these two approaches are used later in this chapter as a framework for experimental comparison with the sequential state-estimation approach for pattern-classification. Section III describes how the idea of nonlinear sequential state estimation can be used as supervised training of a multilayer perceptron. With the material of Section III at hand, the stage is set for specializing the well-known extended Kalman filter (EKF) for the supervised training of a multilayer perceptron in Section IV. Section V describes a difficult pattern-classification task, which is used as a

basis for comparing the EKF algorithm with the BP and SVM algorithms. The chapter concludes with a summary and discussion in Section VI.

1.2 Back-propagation and support vector machine-learning algorithms: A Review

In this section, we review two commonly used pattern-classification algorithms, namely, the back-propagation algorithm for the supervised training of multilayer perceptrons and the support vector machine for radial basis function (RBF) networks. The reason for including this review in the chapter is essentially to describe the two frameworks against which a pattern-classifier trained with the extended Kalman filter will be compared later on in the chapter.

1.2.1 Back-propagation learning

Basically, the back-propagation (BP) algorithm is an error-correction algorithm, providing a generalization of the ubiquitous least-mean squares (LMS) algorithm. As such, the BP algorithm has many of the attributes associated with LMS algorithm, when it is implemented in the stochastic (supervised) mode of learning:

– simplicity of implementation;

– computational complexity, linear in the number of weights; and

– robustness

Most importantly, the BP algorithm is not an algorithm intended for the optimal supervised training of a multilayer perceptron. Rather, it is an algorithm for computing the partial derivatives of a cost function with respect to prescribed set of adjustable parameters [1].

Speaking of the multilayer perceptron (MLP), Figure 1.1 depicts the architectural graph of this neural network, which has two hidden layers and an output layer consisting of three computational units. Figure 1.2 depicts a portion of the MLP, in which two kinds of signals are identified in the network:

1. *Function Signals*. A functional signal is an input signal (stimulus) that comes in at the input end of the network, propagates forward neuron by neuron through the network, and emerges at the output end of the network as an output signal. We refer to such a signal as a "functional signal" for two reasons: First, it is presumed to perform a useful function at the output of the network. Second, at each neuron of the network through which a function signal passes, the signal is calculated as a function of the inputs and the associated weights applied to that neuron. The function signal is also referred to as the input signal.

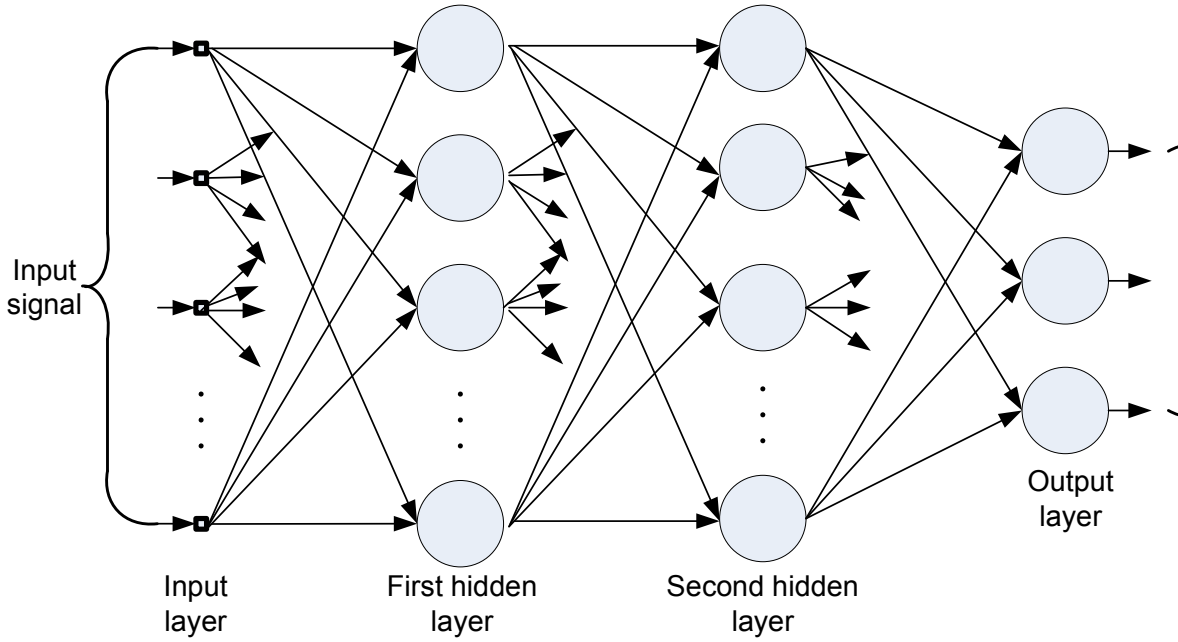


Fig. 1.1 Architectural graph of a multilayer perceptron with two hidden layers.

2. *Error Signals*. An error signal originates at the output neuron of the network and propagates backward (layer by layer) through the network. We refer to the signal as an "error signal" because its computation by every neuron of the network involves an error-dependant function in one form or another.

The output neurons constitute the output layer of the network. The remaining neurons constitute hidden layers of the network. Thus, the hidden units are not part of the output or input of the network– hence their designation as "hidden". The first hidden layer is fed from the input layer made up of sensory units (source nodes); the resulting outputs of the first hidden layer are in turn applied to the next hidden layer; and so on for the rest of the network.

Each hidden or output neuron of a MLP is designed to perform two computations:

1. the computation of the function signal appearing at the output of each neuron, which is expressed as a continuous nonlinear function of the input signal and synaptic weights associated with that neuron;

2. the computation of an estimate of the gradient vector (i.e., the gradients of the error surface with respect to the weights connected to the inputs of a neuron), which is needed for the backward pass thorough the network.

The hidden neurons act as feature detectors; as such, they play a critical role in he operation of a MLP. As the learning process progresses across the MLP, the hidden neurons begin to gradually discover the salient features that characterize the training

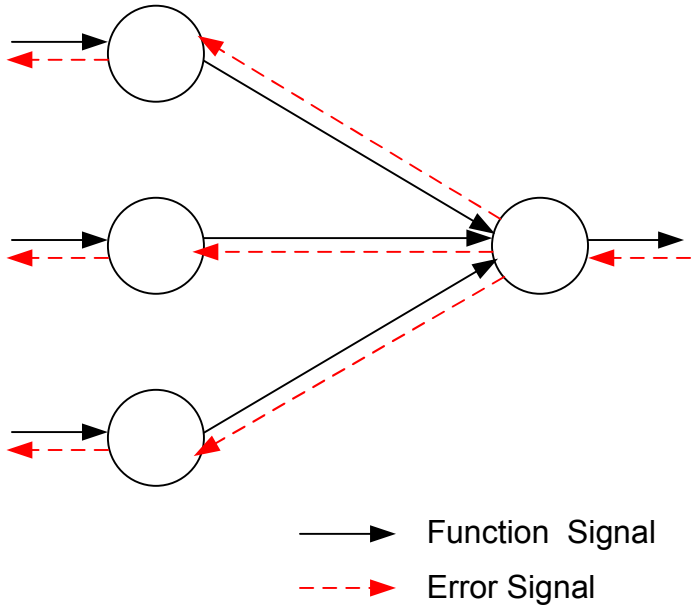


Fig. 1.2 Illustration of the directions of two basic signal flows in a MLP: forward propagation of function signals and back propagation of error signals.

data. They do so by performing a nonlinear transformation on the input data into a new space called the *feature space*. In this new space, the classes of interest in a pattern-classification task, for example, may be more easily separated form each other than could be the case in the original input data space. Indeed, it is the formation of this feature space through supervised learning that distinguishes the MLP from Rosenblatt's perceptron.

As already mentioned, the BP algorithm is an error-correction learning algorithm, which proceeds in two phases:

1. In the *forward phase*, the synaptic weights of the network are fixed, and the input signal is propagated through the network, layer by layer, until it reaches the output. Thus, in this phase, changes are confined to activation potentials and outputs of the neurons in the network.

2. In the *backward phase*, an error signal is produced by comparing the output of the network with a desired response. The resulting error signal is propagated through the network, again layer by layer, but this time the propagation is performed in the backward direction. In this second phase, successive adjustments are made to the synaptic weights of the network. Calculation of the adjustments of the output layer is straightforward, but it is more challenging for the hidden layers.

1.2.2 Support vector machine

The support vector machine (SVM) is basically a binary classification machine with some highly elegant properties. It was pioneered by Vapnik and the first description was presented in [3]. Detailed descriptions of the machine are presented in [1, 2].

Unlike the BP algorithm, supervised training of the SVM looks to optimization theory for its derivation. To develop insights into SVM theory, formulation of the learning algorithm begins with the simple case of a simple pair of linearly separable patterns. In such a setting, the objective is to maximize the margin of separation between the two classes, given a set of training data made up of "input vector-desired response" pairs. Appealing to statistical learning theory, we find that maximization of the margin of separation is equivalent to minimizing the Euclidean norm of the adjustable parameter vector. Building on this important finding, the optimization design of a linearly separable pattern-classifier is formulated as a constrained convex optimization problem, expressed in words as follows:

Minimize the sequence of the adjustable parameter vector of a linearly separable pattern-classifier, subject to the constraint that the parameter vector satisfies the supervisory training data.

Hence, in mathematical terms, the objective function is expressed as a Lagrange function where the lagrange multiplier constitute a set of auxiliary nonnegative variables equal in number to the training sample size. There are three important points that emerge form differentiating the Lagrangian function with respect to the adjustable parameters, including a bias term:

1. The Lagrange multipliers are divided into two sets: one zero, and the other non-zero.

2. The equality constraints are satisfied only by those Lagrange multipliers that are non-zero in accordance with the Karush-Kuhn-Turker (KKT) conditions of convex optimization theory.

3. The supervisory data points for which the KKT conditions are satisfied are called *support vectors*, which, in effect, define those data points that are the more difficult to classify. In light of this third point, the training data set is said to be *sparse* in the sense that only a fraction of them defines the boundaries of the margin of separation.

Given such a combined convex optimization problem, it is possible to construct another problem called the *dual problem*. Naturally, this second problem has the same optimal value as the *primal problem* (i.e., the original optimization problem), but with an important difference:

The optimal solution to the primal problem is expressed in terms of free network parameters, whereas the Lagrange multipliers provide the optimal solution to the dual problem.

This statement is of profound practical importance for it teaches us that the decision boundary for the linearly separable pattern-classification problem can be computed

without having to find the optimal value of the parameter vector. To be more specific, the objective function to be optimized in the dual problem takes the following form:

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j \tag{1.1}$$

subject to the constraints:

$$\begin{aligned} \sum_{i=1}^{N} \alpha_i d_i &= 0 \\ \alpha_i &\geq 0 \text{ for} i = 1, 2, \ldots N, \end{aligned}$$

where α_i in (1.1) are the Lagrange multipliers, and the training data sample is denoted by $\{\mathbf{x}_i, d_i\}_{i=1}^{N}$ with $\mathbf{x}_i$ denoting an input vector, d_i denoting the corresponding desired response, and N denoting the sample size.

Careful examination of (1.1) reveals essential ingredients of SVM learning, namely, the requirement to solve a *quadratic programming problem*, which can become a difficult proposition to execute in computational terms, particularly when the sample size N is large.

What we have addressed thus far is the optimal solution to a linearly separable pattern-classification problem. To extend the SVM learning theory to the more difficult case of nonseparable pattern-classification problems, a set of variables called *slack variables* are introduced into the convex optimization problem; these new variables measure the deviation of a point from the ideal condition of perfect pattern separability. It turns out that in mathematical terms, the impact of slack variables is merely to modify the second constraint in the dual optimization problem described in (1.1) as follows:

$$0 \leq \alpha_i \leq C \quad \text{for } i = 1, 2, \ldots N,$$

where C is a user-specified positive parameter. Except for this modification, formulation of the dual optimization problem for nonseparable pattern-classification problems remains intact. Moreover, the support vectors are defined in the same way as before. The new parameter C controls the tradeoff between complexity of the machine and the number of nonseparable patterns; it may therefore be viewed as the reciprocal of a parameter commonly refereed to the "regularization" parameter. When the parameter C is assigned a large value, the implication is that there is high confidence in the practical quality of the supervisory training data. Conversely, when C is assigned a small value, the training data sample is considered to be noisy, suggesting that less emphasis should be placed in its use and more attention be given to the adjustable parameter vector.

Thus far in the discussion, we have focussed attention on an algorithmic formulation of the SVM. Clearly, we need a network structure for its implementation. In this context, the radial-basis function (RBF) provides a popular choice. Figure 1.3 depicts a structure of a RBF network, in which the nonlinear hidden layer of size K (smaller than the sample size N) consists of Gaussian units; nonlinearity of the hidden layer is needed to account for nonseparability of the input patterns. Each Gaussian unit is defined in terms of its m-dimensional center, where m is the dimensionality of the input data space. To complete the specifications of the hidden layer, we need to also

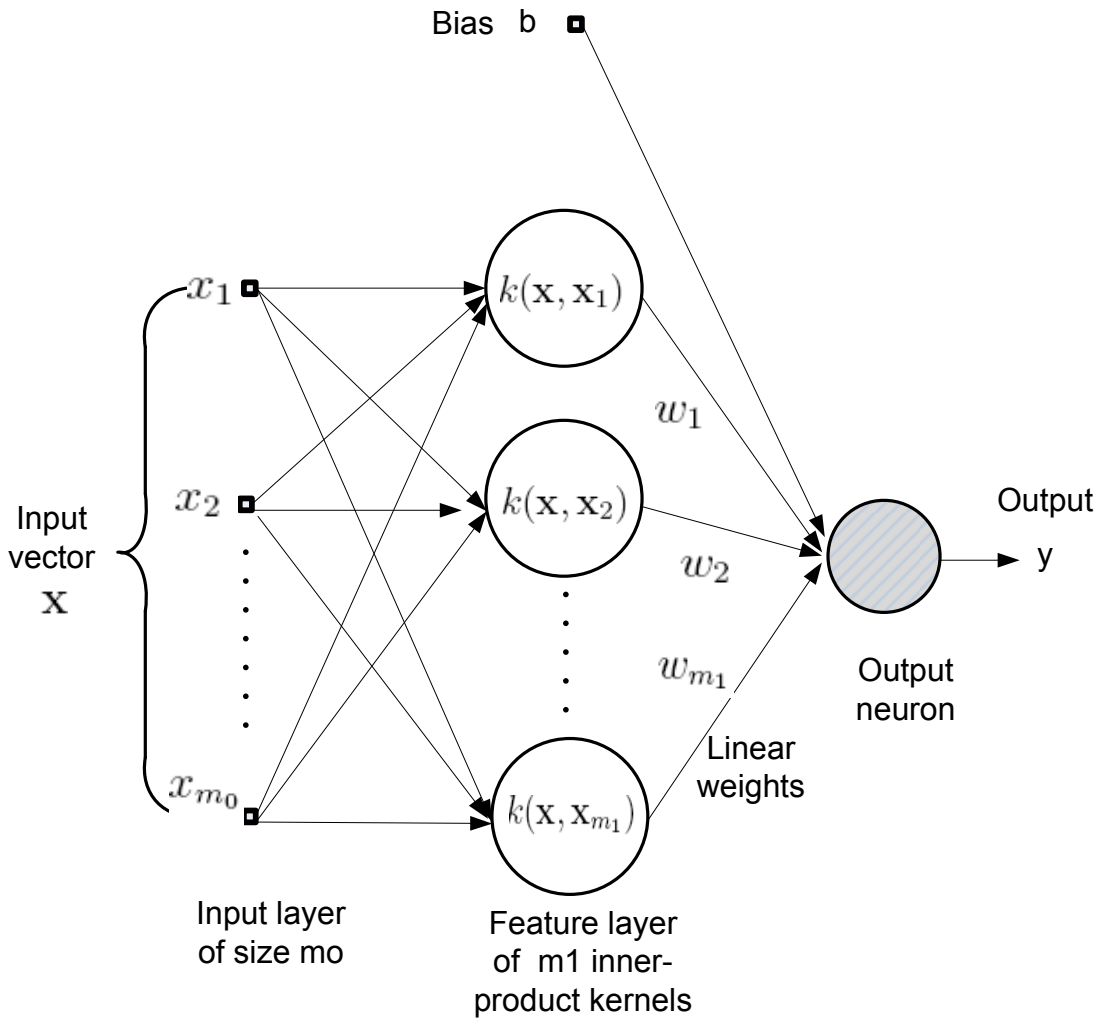


Fig. 1.3 Architecture of support vector machine, using a radial basis function network.

define the effective width of each Gaussian function acting as a "receptive field". To simplify the network design, it is common practice to assign a common width to all the Gaussian units. As for the output layer, it is linear with m_1 adjustable parameters $w_1, w_2, \ldots w_{m_1}$, and an adjustable bias.

To propose an application of the SVM algorithm to the RBF structure of Figure 1.3, we keep the following points in mind:

1. The support vectors picked out of the supervisory training data by the SVM algorithm define the centers of the Gaussian units in the hidden layer.

2. The outputs of the Gaussian units, produced in response to the data applied to the source (input) nodes of the RBF network, define the inputs applied to the adjustable parameters. In effect, $\mathbf{x}_i$ in (1.1) is now replaced by the vector

$$\mathbf{\Phi}(\mathbf{x}_i) \quad = \quad [\Phi(\mathbf{x}_i, \bar{\mathbf{x}}_1),\ \Phi(\mathbf{x}_i, \bar{\mathbf{x}}_2), \ldots \Phi(\mathbf{x}_i, \bar{\mathbf{x}}_k)]^T, \tag{1.2}$$

where $\{\bar{\mathbf{x}}_j\}_{j=1}^K$ denotes the set of support vectors computed by the SVM algorithm. Thus, using (1.2) in (1.1), we see that the inner product $\mathbf{x}_i^T\mathbf{x}_j$ in (1.1) is replaced by the new inner product

$$k(\mathbf{x}_i, \mathbf{x}_j) \quad = \quad \mathbf{\Phi}^T(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x}_j), \tag{1.3}$$

which is called the *Mercer kernel*. This terminology follows from the fact that the functions assigned to the hidden units of the RBF network must satisfy the celebrated *Mercer's theorem*, which is indeed satisfied by the choice of Gaussian functions. Equation (1.3) provides the basis for the *kernel trick*, which states the following:

Insofar as pattern-classification in the output space of the RBF network is concerned, specification of the Mercer kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ is sufficient, the implication of which is that there is no need to explicitly compute the adjustable parameters of the output layer.

For this reason, a network structure exemplified by the RBF network, is referred to as a *kernel machine* and the SVM learning algorithm used to train it is referred to as a *kernel method*.

1.3 Supervised Training framework of MLPs using nonlinear sequential state estimation

To describe how a nonlinear sequential state estimator can be used to train a MLP in a supervised manner, consider a MLP with s synaptic weights and p output nodes. With n denoting a time-step in the supervised training of the MLP, let the vector $\mathbf{w}_n$ denote the entire set of synaptic weights in the MLP computed at time step n. For example, we may construct $\mathbf{w}_n$ by stacking the weights associated with neuron 1 in the first hidden layer on top of each other, followed by those of neuron 2, carrying on in this manner until we have accounted for all the neurons in the first hidden layer; then we do the same for the second and any other hidden layer in the MLP, until all the weights in the MLP have been accounted for in the vector $\mathbf{w}_n$ in the orderly fashion just described.

With sequential state estimation in mind, the state-space model of the MLP under training is defined by the following pair of models (see Figure 1.4):

1. *System (state) model*, which is described by the first-order autoregressive equation

$$\mathbf{w}_{n+1} \quad = \quad \mathbf{w}_n + \boldsymbol{\omega}_n \tag{1.4}$$

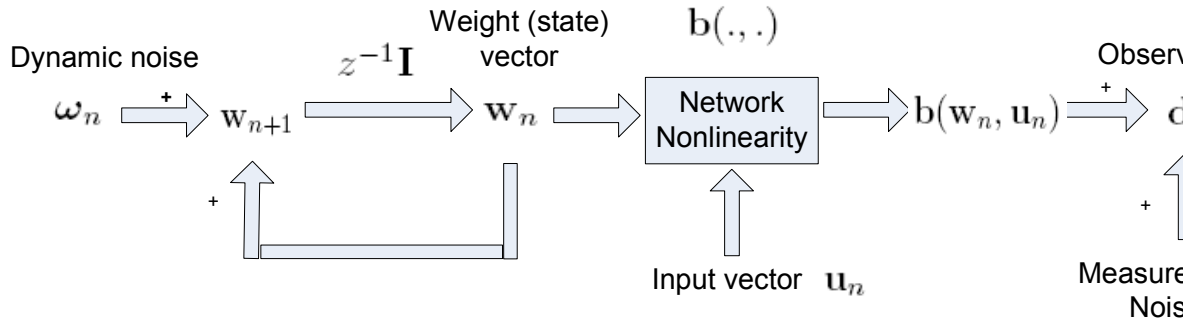


Fig. 1.4 Nonlinear state-space model depicting the underlying dynamics of a MLP undergoing supervised training.

The dynamic noise $\boldsymbol{\omega}_n$ is white Gaussian noise of zero mean and covariance matrix Q_n which is purposely included in the system model to anneal the supervised training of the MLP over time. In the early stages of the training session Q_n is large in order to encourage the supervised-learning algorithm to escape local minima, then it is gradually reduced to some finite but small value.

2. *Measurement model*, which is described by the equation

$$\mathbf{d}_n = \mathbf{b}(\mathbf{w}_n, \mathbf{u}_n) + \mathbf{v}_n \tag{1.5}$$

where the new entities are defined as follows:

– $\mathbf{d}_n$ is the observable

– $\mathbf{u}_n$ is the vector denoting the input signal applied to the network.

– $\mathbf{v}_n$ is the vector denoting the measurement noise process of zero mean and diagonal covariance matrix $\mathbf{R}_n$. The source of this noise is attributed to the way in which $\mathbf{d}_n$ is actually obtained.

The vector-valued measurement function $\mathbf{b}(.,.)$ in (1.5) accounts for the overall nonlinearity of the MLP from the input to the output layer; it is the only source of nonlinearity in the state-space model of the MLP. Here, the notion of state refers to an externally adjustable state, which manifests itself in adjustments applied applied to the MLP's weights through supervised training- hence the inclusion of the weight vector $\mathbf{w}_n$ in the state-space model described by both (1.4) and (1.5).

1.4 The Extended Kalman Filter

Given the training sample $\{\mathbf{u}_n, \mathbf{d}_n\}_{n=1}^N$, the issue of interest is how to undertake the supervised training of the MLP by means of a sequential state estimator. Since the MLP

is nonlinear by virtue of of the nonlinear measurement model of (1.5), the sequential state estimator would have to be correspondingly nonlinear. With this requirement in mind, we begin the discussion by considering how the *extended Kalman filter* (EKF) can be used to fulfil this rule.

For the purpose of our present discussion, the relevant equations of the EKF algorithm summarized in Table 1 are the following two, using the terminology of state-space model of (1.4) and (1.5):

1. the *innovations process*, defined by

$$\boldsymbol{\alpha}_n = \mathbf{d}_n - \mathbf{b}(\hat{\mathbf{w}}_{n|n-1}, \mathbf{u}_n) \tag{1.6}$$

   where the desired response d_n plays the role of the "observable" for the EKF;

2. the *weight (state) update*, defined by

$$\hat{\mathbf{w}}_{n|n} = \hat{\mathbf{w}}_{n|n-1} + \mathbf{G}_n \boldsymbol{\alpha}_n \tag{1.7}$$

   where $\hat{\mathbf{w}}_{n|n-1}$ is the predicted (old) state estimate of the MLP's weight vector $\mathbf{w}$ at time n, given the desired response up to and including time $(n-1)$, and $\hat{\mathbf{w}}_{n|n}$ is the filtered (updated) estimate of $\mathbf{w}$ on the receipt of observable $\mathbf{d}_n$. The matrix $\mathbf{G}_n$ is the *Kalman gain*, which is an integral part of the EKF algorithm.

Summary of the EKF algorithm for Supervised Training of the MLP

Training sample: $\{\mathbf{u}_n, \mathbf{d}_n\}$, $n = 1, 2, \ldots N$
where $\mathbf{u}_n$ is the input vector applied to the MLP and $\mathbf{d}_n$ is the corresponding desired response.
MLP and Kalman filter: Parameters and Variables

$\mathbf{b}(.,.)$	:	vector-valued measurement function
$\mathbf{B}$	:	linearized measurement matrix
$\mathbf{w}_n$	:	weight vector at time step n
$\hat{\mathbf{w}}_{n\|n-1}$	:	predicted estimate of the weight vector
$\hat{\mathbf{w}}_{n\|n}$	:	filtered estimate of the weight vector
$\mathbf{y}_n$	:	output vector of the MLP produced in response to $\mathbf{u}_n$
$\mathbf{Q}_n$	:	covariance matrix of dynamic noise $\boldsymbol{\omega}_n$
$\mathbf{R}_n$	:	covariance matrix of measurement noise $\mathbf{v}_n$
$\mathbf{G}_n$	:	Kalman gain
$\mathbf{P}_{n\|n-1}$	:	prediction error covariance matrix
$\mathbf{P}_{n\|n}$	:	filtering error covariance matrix

Initialization: The initial weights $\hat{\mathbf{w}}_{1|0}$ are drawn from a uniform distribution with zero mean and variance equal to the reciprocal of the number of synaptic connections feeding into a node (fan-in). The associated covariance of the initial weight estimate is fixed at $\delta\mathbf{I}$, where δ can be 'some' multiples of 10, and $\mathbf{I}$ is the identity matrix.

Recursive computation:
For $n = 1, 2, \ldots$ compute the following:

$$\begin{aligned}
\mathbf{G}_n &= \mathbf{P}_{n|n-1}\mathbf{B}_n^T[\mathbf{B}_n\mathbf{P}_{n|n-1}\mathbf{B}_n^T + \mathbf{R}_n]^{-1} \\
\boldsymbol{\alpha}_n &= \mathbf{d}_n - \mathbf{b}_n(\hat{\mathbf{w}}_{n|n-1}, \mathbf{u}_n) \\
\hat{\mathbf{w}}_{n|n} &= \hat{\mathbf{w}}_{n|n-1} + \mathbf{G}_n\boldsymbol{\alpha}_n \\
\mathbf{P}_{n|n} &= \mathbf{P}_{n|n-1} - \mathbf{G}_n\mathbf{B}_n\mathbf{P}_{n|n-1} \\
\hat{\mathbf{w}}_{n+1|n} &= \hat{\mathbf{w}}_{n|n} \\
\mathbf{P}_{n+1|n} &= \mathbf{P}_{n|n} + \mathbf{Q}_n
\end{aligned}$$

Examining the underlying operation of the MLP, we find that the term $\mathbf{b}(\hat{\mathbf{w}}_{n|n-1}, \mathbf{u}_n)$ is the actual output vector $\mathbf{y}_n$ produced by the MLP with its old weight vector $\hat{\mathbf{w}}_{n|n-1}$ in response to the input vector $\mathbf{u}_n$. We may therefore rewrite the combination of (1.6) and (1.7) as a single equation:

$$\hat{\mathbf{w}}_{n|n} = \hat{\mathbf{w}}_{n|n-1} + \mathbf{G}_n(\mathbf{d}_n - \mathbf{y}_n) \tag{1.8}$$

On the basis of this insightful equation, we may now depict the supervised training of the MLP as the combination of two mutually coupled components forming a *closed-loop feedback* system, as shown in Figure 1.5:

1. The top part of the figure depicts the supervised learning process as viewed partly from the network's perspective. With the weight vector set at its old (predicted) value $\hat{\mathbf{w}}_{n|n-1}$, the MLP computes the actual weight vector $\mathbf{y}_n$ as the predicted estimate of the observable– namely $\hat{\mathbf{d}}_{n|n-1}$.

2. The bottom part of the figure depicts the EKF in its role as the facilitator of the training process. Supplied with $\hat{\mathbf{d}}_{n|n-1} = \mathbf{y}_n$, the EKF updates the old estimate of the weight vector by operating on the current desired response $\mathbf{d}_n$. The filtered estimate of the weight vector, namely, $\hat{\mathbf{w}}_{n|n}$, is thus computed in accordance with (1.8). The EKF supplies $\hat{\mathbf{w}}_{n|n}$ to the MLP via a bank of unit-time delays.

With the transition matrix being equal to the identity matrix, as evidenced by (1.4), we may set $\hat{\mathbf{w}}_{n+1|n}$ equal to $\hat{\mathbf{w}}_{n|n}$ for the next iteration. This equality permits the supervised training cycle to be repeated until the training session is completed.

Note that in the supervised training framework of Figure 1.5, the training sample $(\mathbf{u}_n, \mathbf{d}_n)$ is split between the MLP and the EKF:

The input vector $\mathbf{u}_n$ is applied to the MLP as the excitation, and the desired response $\mathbf{d}_n$ is applied to the EKF as the observable, which is dependant on the hidden weight (state) vector $\mathbf{w}_n$.

From the *predictor-corrector property* of the Kalman filter and its variants and extensions, we find that examination of the block diagram of Figure 1.5 leads us to make the following insightful statement:

The multilayer perceptron, undergoing training, performs the role of the predictor, and the EKF, providing the supervision, performs the role of the corrector.

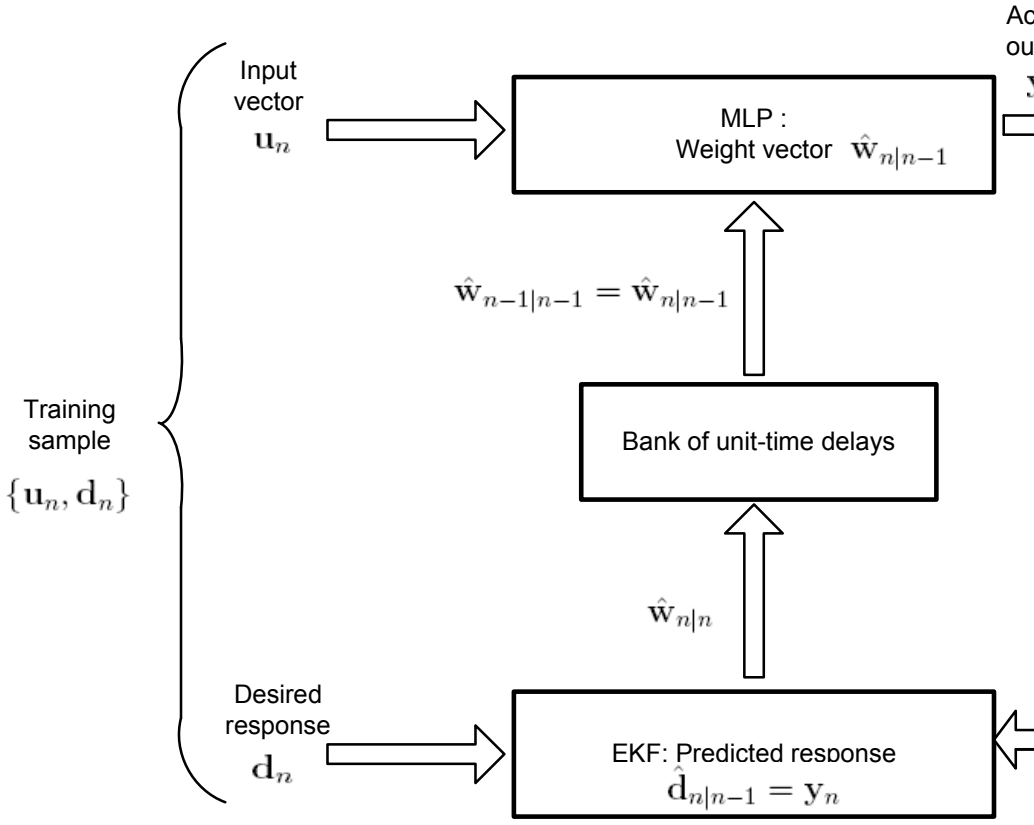


Fig. 1.5 Closed-loop feedback system embodying the MLP and the EKF: (a) The MLP, with weight vector $\hat{\mathbf{w}}_{n|n-1}$, operates on the input vector $\mathbf{u}_n$ to produce the output vector $\mathbf{y}_n$ (b) The EKF, supplied with the prediction $\hat{\mathbf{d}}_{n|n-1} = \mathbf{y}_n$, operates on the current desired response $\mathbf{d}_n$ to produce the filtered weight estimate $\hat{\mathbf{w}}_{n|n}$ thereby preparing the closed-loop feedback system for the next iteration.

Thus, whereas in traditional applications of the Kalman filter for sequential state estimation, the roles of the predictor and corrector are embodied in the Kalman filter itself, in supervised training applications these two roles are split between the MLP and the EKF. Such a split of responsibilities in supervised learning is in perfect accord with the way in which the input and the desired response of the training sample are split in Figure 1.5.

1.4.1

The EKF Algorithm

For us to be able to apply the EKF algorithm as the facilitator of the supervised learning task, we have to linearize the measurement equation (1.5) by retaining first order terms of the Taylor series expansion of the nonlinear part of the equation. With $\mathbf{b}(\mathbf{w}_n, \mathbf{u}_n)$ as the only source of nonlinearity, we may approximate (1.5) as

$$\mathbf{d}_n \quad = \quad \mathbf{B}_n \mathbf{w}_n + \mathbf{v}_n, \tag{1.9}$$

where $\mathbf{B}_n$ is the $p-$by$-s$ measurement matrix of the linearized model. The linearization process involves computing the partial derivatives of the p outputs of the MLP with respect to its s weights, obtaining the required matrix

$$\mathbf{B} \quad = \quad \begin{pmatrix} \frac{\partial b_1}{\partial w_1} & \frac{\partial b_1}{\partial w_2} & \cdots & \frac{\partial b_1}{\partial w_s} \\ \frac{\partial b_2}{\partial w_1} & \frac{\partial b_2}{\partial w_2} & \cdots & \frac{\partial b_2}{\partial w_s} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial b_p}{\partial w_1} & \frac{\partial b_p}{\partial w_2} & \cdots & \frac{\partial b_p}{\partial w_s} \end{pmatrix}, \tag{1.10}$$

where $b_i, \ i = 1, 2, \ldots p$, in (1.10) denotes the i-th element of the vectorial function $\mathbf{b}(.,.)$, and the partial derivatives on the right-hand side of (1.10) are evaluated at $\mathbf{w}_n = \hat{\mathbf{w}}_{n|n-1}$. Recognizing that the dimensionality of the weight vector $\mathbf{w}$ is s, it follows that the matrix product $\mathbf{Bw}$ is a $p-$by-1 vector, which is in agreement with the dimensionality of the observable $\mathbf{d}$.

1.5 Experimental comparison of the extended Kalman filtering algorithm with the back-propagation and support vector machine learning algorithms

In this section, we consider a binary class classification problem as shown in Figure 1.7(a). It consists of three concentric circles of radii 0.3, 0.8, and 1. It is a challenging problem because the two regions are disjoint and non-convex. For the experiment, we used the following supervised training algorithms:

– MLP trained by the BP;

– MLP trained by the EKF; and

– SVM

The structure of the MLP was chosen to have two inputs, one output, and two hidden layers, with five neurons each. Hence, the MLP has a total of 51 adjustable parameters or weights (bias included). All neurons use the hyperbolic tangent function of the form:

$$\varphi(\upsilon) \quad = \quad \tanh(\upsilon) = \frac{\exp(\upsilon) - \exp(-\upsilon)}{\exp(\upsilon) + \exp(-\upsilon)}$$

The learning-rate parameter of the back-propagation algorithm was fixed at 0.01. For the MLPs trained by the BP and the EKF, the initial weights were set up as described

in Section IV. For the EKF, two more covariance matrices were additionally required to set up:

1. The covariance matrix of dynamic noise $\mathbf{Q}_n$ was annealed such that $\mathbf{Q}_n = (\frac{1}{\lambda} - 1)\mathbf{P}_{n|n}$, where $\mathbf{P}_{n|n}$ was the error covariance associated with the weight estimate at time instant n, and $\lambda \in (0, 1)$ is the "forgetting factor" as defined in a recursive least-squares algorithm; this approximately assigns exponentially decaying weights to past observables; λ was fixed at 0.9995.

2. The variance of measurement noise R_n was fixed at unity.

For the SVM, the kernel was chosen to be the Gaussian radial-basis function. A kernel width of $\sigma = 0.2$ was found to be a good choice for this problem. This value was chosen based on the accuracy of test classification results. The soft margin of unity was found to be more appropriate for this problem. The quadratic programming code, available as an in-built routine in the MATLAB optimization toolbox, was used for training the SVM.

For the purpose of training 1000 data points were randomly picked from the considered region. In the MLP-based classification, each training epoch contained 100 examples randomly drawn from the training data set. For the purpose of testing, the test data were prepared as follows: A grid of 100×100 data points were chosen from the square region $[-1, 1] \times [-1, 1]$ (see Figure 1.7(a)) and then the grid points falling outside the unit circle were discarded. In so doing, we were able to obtain 7825 test points.

At the end of every 10 training epoch interval, the MLP was presented with the test data set. We made 50 such independent experiment trials and the ensemble-averaged correct classification rate was computed. In contrast, for the SVM, this test grid was presented at the end of the training. Figures 1.6(a) and 1.6(b) show the results of the BP-trained MLP and the EKF-trained MLP, respectively. At the end of 5000 training epochs, we found that:

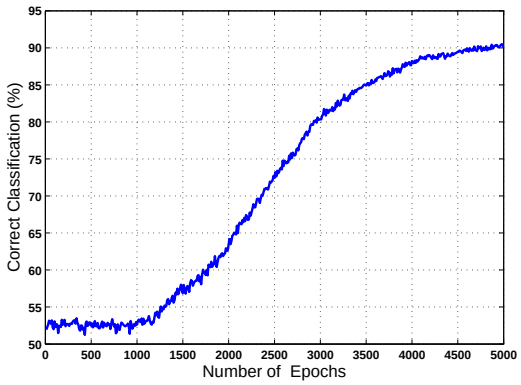
the BP-trained MLP achieves nearly 90% classification accuracy; whereas, at the end of 200 training epochs, the EKF-trained MLP achieves nearly 96% classification accuracy.

Figures 1.7(b)-1.7(d) depict a representative test classification results of the employed algorithms.

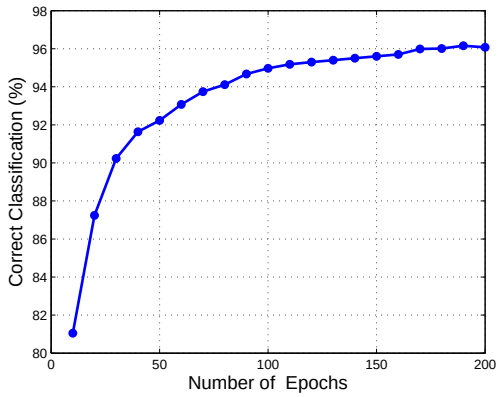
We also computed the computational times of each algorithm averaged over 50 independent trials when executing on an Intel Pentium Core Duo processor of cycle speed 2.1GHz. The summarized results are tabulated in Table I. To execute 5000 epochs of the BP, roughly a minute is required, whereas, the EKF requires only 8 seconds for completing its 200 training epochs. As expected, the SVM takes more time (nearly 3 minutes) to complete its training.

Based on the computational time-accuracy tradeoff, we may say that the EKF-trained MLP is the best choice for solving the difficult binary pattern-classification problem illustrated by Figure 1.7(a).

1.6



(a) Backprop



(b) EKF

Fig. 1.6 Ensemble-averaged (over 50 runs) correct classification rate Vs. number of epochs

Concluding Remarks

In the neural networks and learning machines literature, the solution to pattern-classification problems is commonly tackled, for example, by using a multilayer perceptron (MLP) trained with the back-propagation algorithm or a radial-basis function (RBF) network using the support-vector machine (SVM) learning algorithm [1]. In this chapter, we have discussed another way of tackling the pattern-classification problem, namely a multilayer perceptron trained using the extended Kalman filter (EKF). The rationale for this unusual approach is that the supervised learning process involving in the training of multilayer perceptron may be viewed as a state-estimation problem, in

Algorithm	Test accuracy	Time (s)
BP	90%	63s
EKF	96%	8s
SVM	96%	182s

Table 1.1 Comparison of classification accuracy and computational times averaged over 50 independent experiments

which case the weight vector characterizing the multilayer perceptron is viewed as a hidden state. An attracting feature of using the EKF for estimating the unknown weight vector is the improved utilization of the information contained in the training sample by propagating the covariance matrix of the filtered estimation error, hence the accelerated rate of learning. This point is borne out by shorter time taken to complete the learning process, compared to both the BP and SVM algorithms. Moreover, the experimental results presented in Section V show that the EKF approach to supervised training of a multilayer perceptron achieves a pattern-classification accuracy that matches the results obtainable with the SVM.

To conclude, we may say that the supervised training of a multilayer perceptron using the EKF as the tool to oversee the supervision deserves consideration when the requirement is to solve difficult pattern-classification problems.

1.7 Problems

1. You are given a set of supervisory data samples of large enough size. The data are to be used for training of a learning machine, which is required to solve a difficult pattern-classification exemplified by that of Figure 1.7(a).

   Describe the attributes of a learning machine that is required to tackle such pattern-classification problem.

2. Discuss the pros and cons of the extended Kalman filtering algorithm used to train a multilayer perceptron for solving the pattern-classification problem of Figure 1.7(a) compared to the back-propagation algorithm.

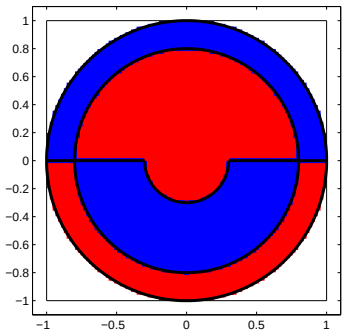
3. Repeat the experiment described in Section V to verify your conclusions in Problem 2.

4. Discuss the pros and cons of the extended Kalman filtering algorithm used to train a multilayer perceptron for solving the pattern-classification problem of Figure 1.7(a) compared to a radial-basis function network trained using the support vector machine (SVM) learning algorithm.

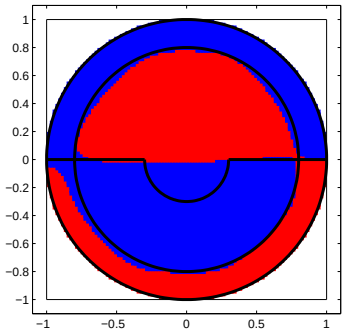
5. Repeat the experiment described in Section V to verify your conclusions in Problem 4.

6. The learning capability of the human brain extends beyond pattern-classification. To probe further, consider two persons who love listening to music. One is around 20 to 30 years old, and the other is around 50 to 70 years old. When a piece of music is played on the radio that goes back to the 20s or 30s, it is the first person who very much enjoys it. On the other hand, when the piece of music played on the same radio goes back to the 50s and 70s, it is the second person who very much enjoys it.

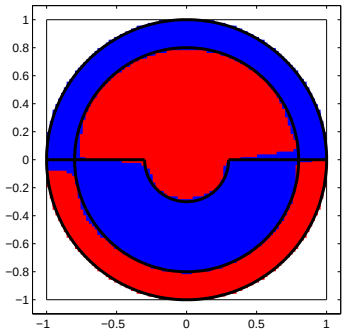
   Explain the main reason that is responsible for this difference.



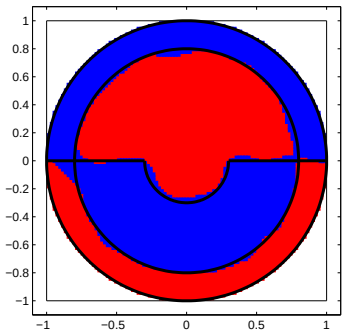
(a) True classification region



(b) Back-propagation



(c) EKF



(d) SVM

Fig. 1.7 (a): Original; (b)-(d): Representative test classification results.

References

1 S. Haykin, *Neural networks and learning machines*, 3rd ed., Prentice Hall, 2008.

2 B. Schölkolf, and A. J. Smola, *Learning with kernels: Support vector machines, rgularization, optimization, and beyond*, MA: MIT Press, 2003.

3 V. N. Vapnik, *Statistical learning theory*, NY: Wiley, 1998.