3. Learning laws for linear-in-the-parameters networks

"Discovery consists of seeing what everybody has seen, – and thinking what nobody has thought." – Albert Szenti-Gyorgyi

3.1 Introduction to learning

Despite the fact that artificial neural networks (ANNs) have been proposed primarily as nonlinear learning systems, considerable insight into the behaviour of these networks can be gained from linear modelling techniques, for which the literature is vast (see for example [8, 137, 181]), and significantly the resultant theory is directly applicable to a powerful and special class of ANNs, i.e. linear-in-the-parameters networks, which forms the basis of this book. An obvious reason to use ANNs is their ability to approximate arbitrarily well any continuous nonlinear function, with the specific network architecture and parameter/weight adjustment algorithm determining how well learning is achieved. Of particular interest in adaptive control and estimation is the ability of algorithms to model, track and control on-line. However, it is infeasible to assume that the input signals excite the whole of the state space (a prerequisite of identification theory), and so it is necessary to consider the effects of a reduced input signal on overall functional approximation for various network architectures and associated learning laws. Here learning must be local, in that adjustable network weights or parameters should only affect the network's output locally. In ANNs, the vast majority of supervised learning rules are based on the assumption that the nonlinear network can be locally linearised, so that it is natural to develop on-line learning algorithms with provable learning network stability and convergence conditions, and to choose ANNs which have linear-in-the-parameters with local behavioural characteristics. In particular, nonlinear networks, such as the cerebellum model articulation controller (CMAC), radial basis function (RBF), B-splines (see [32]), Bézier–Bernstein polynomial (see Section 7.4) networks, all have an output layer of linear parameters, with basis functions having local compact support. These so-called linear-in-the-parameters networks generally have large memory requirements for generating the optimal network parameter or weight vector from batch data, where direct optimisation methods, generally based on matrix inversion with a computational cost of $O(p^3)$ (p represents network size), can be used. However, as the system equations for these networks are generally sparse and singular, then numerically stable matrix inversion techniques that exploit these characteristics should be utilised. Alternatively, iterative techniques that avoid matrix inversion to calculate the optimal weight vector can be derived (see Section 3.3, and for a detailed discussion of learning for linear-in-the-parameters networks see Brown and Harris [32]).

In order that an adaptive network can track an unknown nonlinear time-varying system, it must be able to reorganise itself in real-time via on-line instantaneous learning or training that uses an instantaneous estimate of the current network performance prior to weight updating. This is a significantly different problem to recursive or iterative weight updating from batch data, where iteration is used for computational ease (e.g. to avoid matrix inversion), since in instantaneous learning only the estimate of the instantaneous performance value, gradient, etc. are available, which may well be significantly different from the true value. In this case, a variety of instantaneous least mean squares (LMS) learning rules have been developed for use with on-line modelling and control (see Section 3.4), in which the instantaneous error is used to approximate the total cost function, such as MSE, leading to convergence problem. The problem of using instantaneous gradient estimates introduces an additional noise term into the learning process, causing weight convergence to a domain or region around the optimal weight vection (see Section 3.4.3).

As linear-in-the-parameters networks are used throughout this book, then gradient descent rules are highly appropriate. Also since all linear-in-the-parameters networks map the input regressor vectors to a higher dimensional sparse space, prior to the output weight layer, then instantaneous learning laws can exploit this sparse structure with its inherent localised response, as only weights that contribute locally to the output are updated. Learning is local as dissimilar inputs are mapped to different weight sets, ensuring that instantaneous learning is highly appropriate for weight training for linear-in-the-parameters networks.

Throughout this chapter, various learning laws are developed, which have a common structure, such that the new weight value is the past value plus an output error multiplied by the transformed input regressor vector. Generally performance is based on the network's MSE, as it leads to simple, analytically tractable laws, which give acceptable models, and when subject to additive Gaussian noise it is also the maximum likelihood estimate.

The instantaneous learning laws developed in this chapter can also be considered as line search optimisation techniques [70], since they produce a search direction or path, along which the weights or parameters are updated, and then compute a step size which ensures stable learning, albeit with limited training data. Throughout this book we utilise instantaneous normalised least squares learning (Section 3.4.2) due to independence of the learning rate, δ, or the basis functions and its inherent stable learning properties compared with other learning laws. The following sections develop the interrelationships between error performance surfaces and various parametric learning laws.

3.2 Error or performance surfaces

A network's output error $e_y(t) = y(t) - \hat{y}(t)$ is available in supervised learning as an instantaneous measure of the current model's performance, and as such is useful for feedback via some learning law derived from a performance function for weight or parameter updating. Learning laws are derived so as to modify the estimate of $\mathbf{w}$, such that as the amount of data increases, an optimal parameter vector (weight) is derived by globally minimising some prespecified cost function $V_N(\mathbf{w})$ over all available training data D_N, as $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}}[V_N(\mathbf{w}, D_N)]$, where $V_N(\mathbf{w}, D_N)$ can be defined as various cost functions, such as

$$V_N(\mathbf{w}, D_N) = \begin{cases} E[|e_y(t)|] \\ E[e_y^2(t)] \\ \max_t |e_y(t)|. \end{cases} \tag{3.1}$$

Only when a weight vector exactly models a desired function, are the solutions to the various cost functions in (3.1) the same, since each places a different emphasis upon the instantaneous error. The choice of the cost function influences the type of learning law, its computational complexity, its convergence properties, as well as the resultant final model. For general classes of computational models with multiple adaptive layers of adjustable weights, such as multilayer perceptrons (MLPs), the error function and the consequent cost function will be a highly nonlinear function of the weights, for which many local minima exist, satisfying $\nabla V_N(\mathbf{w}, D_N) = 0$, where $\nabla V_N(\mathbf{w}, D_N)$ is the gradient of $V_N(\mathbf{w}, D_N)$.

Here it is not in general possible to find closed form analytical solutions for $\mathbf{w}$. This problem also applies to linear-in-the-parameters networks, such as RBFs, various neurofuzzy networks, CMAC, etc. in which the basis functions positions, orders, dilation parameters, etc. are simultaneously adjusted with the parameters $\mathbf{w}$. In this chapter we assume that these model structural properties are fixed or determined off-line by some model construction algorithms (see Chapter 5). In this case, the network's MSE (see (2.6)) produces a (hyper)quadratic surface in p-dimensional weight space with a unique global minimum $\hat{\mathbf{w}}$ (see Figure 3.1 for the 2-D weight space case).

Substituting the following model (2.16)

$$\hat{y}(t) = \psi^T(\mathbf{x}(t))\mathbf{w} \tag{3.2}$$

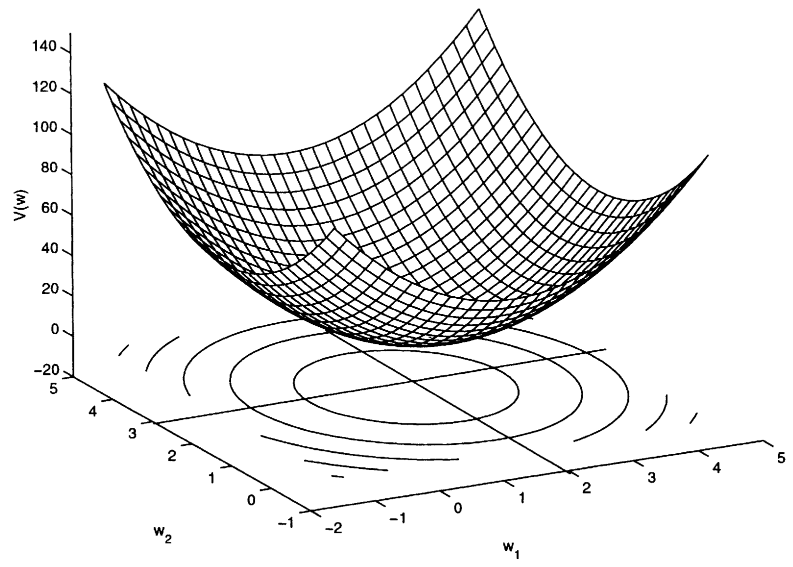

Fig. 3.1. Error surface in a 2-D weight space. The optimal weight occurs at $\mathbf{w} = [2, 3]^T$

into the quadratic cost function

$$V_N(\mathbf{w}, D_N) = \frac{1}{N}\sum_{t=1}^{N}[y(t) - \hat{y}(t)]^2, \tag{3.3}$$

gives

$$V_N(\mathbf{w}, D_N) = \frac{\mathbf{y}^T\mathbf{y}}{N} + \mathbf{w}^T R\mathbf{w} - 2\mathbf{m}^T\mathbf{w}, \tag{3.4}$$

where R is a $p \times p$ semi-positive definite, symmetric autocorrelation matrix of the basis functions defined by

$$\begin{aligned} R &= E[\psi(t)\psi^T(t)] = \frac{1}{N}\mathbf{A}^T\mathbf{A} \\ &= \frac{1}{N}[\psi(\mathbf{x}(1))...\psi(\mathbf{x}(N))][\psi(\mathbf{x}(1))...\psi(\mathbf{x}(N))]^T \qquad (3.5) \\ &= \begin{bmatrix} \frac{1}{N}\sum_{t=1}^{N}[\psi_1^2(t)] & \frac{1}{N}\sum_{t=1}^{N}[\psi_1(t)\psi_2(t)] & \cdots & \frac{1}{N}\sum_{t=1}^{N}[\psi_1(t)\psi_p(t)] \\ \frac{1}{N}\sum_{t=1}^{N}[\psi_2(t)\psi_1(t)] & \frac{1}{N}\sum_{t=1}^{N}[\psi_2^2(t)] & \cdots & \frac{1}{N}\sum_{t=1}^{N}[\psi_2(t)\psi_p(t)] \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{N}\sum_{t=1}^{N}[\psi_p(t)\psi_1(t)] & \frac{1}{N}\sum_{t=1}^{N}[\psi_p(t)\psi_2(t)] & \cdots & \frac{1}{N}\sum_{t=1}^{N}[\psi_p^2(t)] \end{bmatrix}, \end{aligned}$$

and $\mathbf{m} = \frac{1}{N}\mathbf{A}^T\mathbf{y}$ is a cross-correlation vector.

When R is nonsingular, the optimal weight vector $\hat{\mathbf{w}}$ is given by differentiating (3.4) with respect to $\mathbf{w}$ and setting it to zero, resulting in the set of (so-called normal) linear equations:

$$R\hat{\mathbf{w}} = \mathbf{m}, \tag{3.6}$$

whose solution is given by the pseudo-inverse as

$$\hat{\mathbf{w}} = R^{-1}\mathbf{A}^T\mathbf{y} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}. \tag{3.7}$$

For this least squares (also maximum likelihood if the additive noise process $e(t)$ in (2.4) is a Gaussian process) solution is unique if $rank(\mathbf{A}) = p$, otherwise it is under-determined with an infinity of solutions. In this situation it is desirable to find the minimum norm solution ($\min \|\mathbf{w}\|_2$), since such weight values mean that the network should generalise sensibly across its input space (see Section 2.4). For many problems the solution matrix $\mathbf{A}$ is sparse, whilst R is not, and hence algorithms that do not explicitly form R can often prove computationally efficient [32].

Fortunately the networks discussed in this book, e.g. B-spline neurofuzzy systems, are naturally sparse due to their inherent structure and this sparsity is preserved in R. Irrespective of what method is used in solving (3.7) the ability to find a solution is dependent upon the condition number, $C(R)$, of the matrix R:

$$C(R) = \frac{\max\ nonzero\ \lambda_i}{\min\ nonzero\ \lambda_i}, \tag{3.8}$$

where λ_i are the eigenvalues of R. If the condition number is large (e.g. 10^6), then the problem is ill-conditioned, whilst if it is close to unity, it is well conditioned (see [32] for detailed discussion on conditioning of neural networks).

The minimum MSE, $V_N(\hat{\mathbf{w}}) = V_{min}$, occurs when $\mathbf{w} = \hat{\mathbf{w}}$ in (3.4). Noting that R is symmetric, we have

$$V_{min} = \frac{\mathbf{y}^T\mathbf{y}}{N} - \mathbf{m}^T\hat{\mathbf{w}}. \tag{3.9}$$

So defining the current weight error as $\mathbf{e}_\mathbf{w} = \hat{\mathbf{w}} - \mathbf{w}$, then the cost (3.4) becomes on simplification via (3.9):

$$V_N(\mathbf{w}) = V_{min} + \mathbf{e}_\mathbf{w}^T R \mathbf{e}_\mathbf{w}, \tag{3.10}$$

which defines a quadratic function around $\mathbf{w} = \hat{\mathbf{w}}$ (as R is positive definite). When $\mathbf{e}_\mathbf{w}$ lies in the null space of R or is zero, $V_N = V_{min}$, and as with the network condition number, $C(R)$, the autocorrelation matrix R is critical in determining network performance.

Note: Consider the above model is used to determine the nonlinear process of (2.4), via (2.16), in which the additive noise is uncorrelated with zero mean and variance σ^2, the least squares solution is identical to (3.7) and the estimate $\hat{\mathbf{w}} = E(\mathbf{w})$ (i.e. $\hat{\mathbf{w}}$ is unbiased) and $\text{cov}(\hat{\mathbf{w}}) = \sigma^2(\mathbf{A}^T\mathbf{A})^{-1}$ (see

[104]), which tends to zero as $N \to \infty$ if $(\mathbf{A}^T\mathbf{A})^{-1} \to 0$ (i.e. $\hat{\mathbf{w}}$ is a consistent estimator).

Now the autocorrelation matrix R can be decomposed into its normal form

$$R = U\Lambda U^T,$$

where U is the unitary matrix composed of the orthonormal eigenvectors of R and $\Lambda = diag\{\lambda_i\}$, is the $p \times p$ diagonal matrix of the non-negative eigenvalues of R. Defining the vector

$$\mathbf{v} = U^T \mathbf{e_w},$$

then (3.10) can be expressed in terms of the eigenvalues of R as

$$V_N(\mathbf{w}) = V_{min} + \mathbf{v}^T \Lambda \mathbf{v} = V_{min} + \sum_{i=1}^{p} v_i^2 \lambda_i. \tag{3.11}$$

So differentiating (3.11) twice with respect to v_i gives

$$\lambda_i = \frac{1}{2}\frac{\partial^2 V_N(\mathbf{w})}{\partial v_i^2}, \quad \forall i.$$

That is, the performance function curvature is equivalent to the eigenvalues of R. The cost function, V_N, defines in the $\mathbf{v}$-space a hyperellipsoid, whose principal axes are the eigenvectors of R, but since U is a unitary matrix, then $\mathbf{v}$ is a rotated version of $\mathbf{e_w}$ in the original weight ($\mathbf{w}$) space. For any $\lambda_i = 0$, the performance function produces a valley in the rotated $\mathbf{v}$ space (with an infinite number of solutions for $\hat{\mathbf{w}}$) along its principal axis in the rotated $\mathbf{v}$ performance space (singular).

3.3 Batch learning laws

3.3.1 General learning laws

For some modelling problems a batch of data $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$ is available, and single shot solution for the optimal weight vector $\hat{\mathbf{w}}$, such as (3.7), can be utilised. However since the cost or performance function is quadratic with a global minimum, iterative and recursive based methods can be used to avoid computational problems associated with large data storage. Typically they use weight updates of the form:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \delta(k)\mathbf{d}(k), \tag{3.12}$$

where k represents the iteration index, $\delta(k)$ is the learning rate or step size, and $\mathbf{d}(k)$ a direction or search vector in the p-dimensional weight space. The various learning laws which have to generate those $\{\delta(k), \mathbf{d}(k)\}$ ensure that

$$E[V(\mathbf{w}(k+1))] = E[V(\mathbf{w}(k) + \delta(k)\mathbf{d}(k))] < E[V(\mathbf{w}(k))].$$

Clearly, irrespective of the shape of the performance function $V(\mathbf{w})$, it can be locally expanded in a Taylor series as a quadratic function for small steps $\delta(k)$:

$$V(\mathbf{w}(k+1)) \simeq V(\mathbf{w}(k)) + \nabla^T V(\mathbf{w}(k))\Delta\mathbf{w}(k) + \frac{1}{2}\Delta\mathbf{w}^T(k)\nabla^2 V(\mathbf{w}(k))\Delta\mathbf{w}(k), \tag{3.13}$$

where $\Delta\mathbf{w}(k) = \mathbf{w}(k+1) - \mathbf{w}(k)$, and $\nabla^2 V(\mathbf{w}) = \mathbf{H}$ the Hessian matrix of $V(\mathbf{w})$, and $\nabla^T V(\mathbf{w})$ is the gradient of $V(\mathbf{w})$. Assuming that the inverse of $\mathbf{H}$ exists, then the optimal iteration to this approximated quadratic is the Newton–Raphson method ($\delta(k) = 1$)

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{H}^{-1}\nabla V(\mathbf{w}(k)). \tag{3.14}$$

If $\mathbf{H}$ is positive definite and $V(\mathbf{w})$ is quadratic, then Newton's method goes to the optimal weight vector $\hat{\mathbf{w}}$ in one step! (Note: If $\mathbf{H}$ is not positive definite, the Newton direction may point towards a local maximum or saddle point. $\mathbf{H}$ can be made positive definite by adding a positive matrix, say $\lambda\mathbf{I}$, so that replacing $\mathbf{H}$ by $\mathbf{H}' = \mathbf{H} + \lambda\mathbf{I}$ in (3.14), resulting in the Levenberg-Marquardt algorithm; this is a form of regularisation and ridge regression.)

3.3.2 Gradient descent algorithms

Computing the Hessian matrix is computationally expensive and inappropriate for batch processes, so many practical algorithms just utilise the gradient $\nabla V = \partial V/\partial\mathbf{w}$, which for the quadratic cost (3.3) is

$$\nabla V = \partial V/\partial\mathbf{w} = 2R\mathbf{w} - 2\mathbf{m} = -2R\mathbf{e}_{\mathbf{w}}. \tag{3.15}$$

Again we note that it depends on the autocorrelation matrix R. Transforming (3.15) to the eigenspace $\mathbf{v}$ via $\mathbf{v} = U^T\mathbf{e}_{\mathbf{w}}$, we have alternatively

$$\nabla V_{\mathbf{v}} = \partial V/\partial\mathbf{v} = 2\Lambda\mathbf{v}.$$

That is, the gradient, with respect to v_i, depends only on the ith eigenvalue of R.

Substituting (3.15) into (3.12) with $\mathbf{d}(k) = -\frac{\nabla V}{2}$ gives the gradient descent weight updating rule:

$$\begin{aligned}\mathbf{w}(k+1) &= \mathbf{w}(k) - \delta(k)(R\mathbf{w}(k) - \mathbf{m}),\\ \textit{or}\ \ \Delta\mathbf{w}(k) &= \delta(k)R\mathbf{e}_{\mathbf{w}}(k),\end{aligned} \tag{3.16}$$

where $\Delta\mathbf{w}(k) = \mathbf{w}(k+1) - \mathbf{w}(k)$, and $\mathbf{w}(k+1) - \mathbf{w}(k) = \mathbf{e}_{\mathbf{w}}(k+1) - \mathbf{e}_{\mathbf{w}}(k)$. Substituting $\hat{\mathbf{w}}$ from (3.6) into (3.16) gives

$$\mathbf{e}_{\mathbf{w}}(k+1) = (1 - \delta(k)R)\mathbf{e}_{\mathbf{w}}(k),$$

or since $\mathbf{v} = U^T\mathbf{e}_{\mathbf{w}}$, in decoupled components,

$$v_i(k+1) = (1 - \delta(k)\lambda_i)v_i(k) \quad \forall i, \tag{3.17}$$

whose solutions are

$$v_i(k+1) = (1-\delta(k)\lambda_i)^k v_i(1). \tag{3.18}$$

Clearly for stable learning, $|1-\delta(k)\lambda_i| < 1$, $\forall i$, or the learning rate $\delta(k)$ must satisfy

$$0 < \delta(k) < \frac{2}{\lambda_{max}}, \tag{3.19}$$

and the model weight vector $\mathbf{w}$ tends to $\hat{\mathbf{w}}$ as $k \to \infty$ (see (3.18)). For zero eigenvalues of R, the error along the ith principal axis does not decay, as there are an infinite number of global minima in the weight space. In this case the error along the principal axis constrains parameter adjustment to a unique value which is dependent upon the initial weight values. Note that replacing δ by $\delta' = \delta/\lambda_{max}$ enables various gradient based learning algorithms to be compared as stability is dictated by the condition $0 < \delta < 2$. The rate of convergence in weight adjustment can be estimated by placing an exponential envelope on the iteration (3.18) as

$$v_i(1)\exp(-\frac{k}{\tau_i}) = v_i(1)(1-\delta'\lambda_i)^k.$$

From which the effective time constant is

$$\tau_i = -\frac{1}{\ln(1-\delta\lambda_i/\lambda_{max})}.$$

As the largest time constant, τ_{max}, corresponds to the smallest eigenvalue, λ_{min}, of R, then

$$\tau_{max} = -\frac{1}{\ln(1-\delta\lambda_{min}/\lambda_{max})} = -\frac{1}{\ln(1-\delta/C(R))}. \tag{3.20}$$

For a well conditioned network, $C(R)$ is close to unity and weight convergence is fast; for an ill-conditioned network ($C(R)$ is large), the performance function is valley shaped rather than bowl shaped, leading to slow convergence.

Steepest descent is one of the most popular gradient algorithms for parametric optimisation; it computes the one-step-ahead optimal step size and moves parallel to the negative gradient by this amount. It is more complex, but faster in convergence than the standard gradient algorithm (3.16). The weight update rule is

$$\triangle\mathbf{w}(k) = \delta(k)\mathbf{d}(k),$$

where $\mathbf{d}(k) = -\frac{1}{2}\partial V/\partial\mathbf{w} = -(R\mathbf{w}(k) - \mathbf{m}) = -\mathbf{r}(k)$, for $\mathbf{r}(k)$ the residual errors in the estimated cross-correlation vector $\mathbf{m}$. The ideal step size is problem dependent. A good practical choice is to use a $\delta(k)$ such that $V_N(\mathbf{w}(k))$ is minimised, which results in choosing δ such that the current search direction is orthogonal to the previous ones, giving [178]

$$\delta(k) = \frac{\mathbf{r}^T(k)\mathbf{r}(k)}{\mathbf{r}^T(k)\mathbf{A}\mathbf{r}(k)}. \tag{3.21}$$

Note that the steepest descent is the same as the Levenberg–Marquardt (Newton's amended algorithm) for $\lambda \to \infty$.

An important aspect of network modelling is the relationship between the weight error vector $\mathbf{e}_\mathbf{w}$, on which most learning laws operate, and the consequent network output prediction error $\mathbf{e}_y$. This functional relationship is given by [32] as

$$\frac{||\mathbf{e}_\mathbf{w}||}{||\mathbf{w}||} \leq \sqrt{C(R)}\frac{||\mathbf{e}_y||}{||\mathbf{y}||}. \tag{3.22}$$

Hence for a poorly conditioned network, a small measured output error does not imply a small weight or parametric error. The network's ability to generalise locally depends on the weight error, and terminating the learning prematurely (as e_y is small) may mean that some weights are not close to their optimal values. Condition (3.22) together with (3.20) and (3.7) shows how important proper selection of the basis functions, their order and positions, and training data is in determining effective network generalisaion behaviour.

Notes:

(i) Conditioning of linear-in-the-parameters models depends on the shape of the basis functions and the distribution of the training data. The degree of activation and overlap of the basis functions determines the form of the autocorrelation, e.g. semi-global basis functions such as sigmoidal may lead to ill-conditioned networks. Generally linear-in-the-parameters networks are well conditioned as the basis functions are locally unimodal, mutually orthogonal (R is sparse), and the basis function power $E(\psi_i^2)$ is generally greater than its interaction with other basis functions as R tends to be diagonally dominant (see Gersgorin's theorem). For examples of various neural network model conditioning see [32].

(ii) Frequently R has at least one zero eigenvalue due to basis functions having no training data lying in its support. Whilst this may cause problems in inverting R, it does not affect overall learning behaviour, such as its learning rate. It only influences which optimal solution (amongst the infinite number possible) will be found, as gradient based algorithms always find the minimum norm solution for $\mathbf{w}$ if appropriately initiated.

3.4 Instantaneous learning laws

3.4.1 Least mean squares learning

For on-line control, conditioning monitoring and tracking problems, and for the modelling of time varying dynamical processes data is generated in real time, therefore estimation must be carried out as data is generated. Also in adaptive control, instantaneous learning algorithms are used where the unknown parameters or weights are recursively estimated using currently available input–output data. Only the instantaneous estimate of the network performance is available for evaluating the network prior to parametric update. The instantaneous MSE, $\frac{1}{2}e_y^2(t) = \frac{1}{2}[y(t) - \hat{y}(t|t-1)]^2$, where $\hat{y}(t|t-1) = \psi^T(t)\mathbf{w}(t-1)$, is frequently used as it is both simple and utilises only the latest available observable data for training. The unbiased instantaneous estimate of the true gradient at t is

$$\hat{\nabla}V = -[y(t) - \hat{y}(t)]\psi(t) = -e_y(t)\psi(t). \tag{3.23}$$

Updating the weight vector in proportion to this gradient estimate gives the least mean squares (LMS) algorithm:

$$\triangle\mathbf{w}(t) = \delta e_y(t)\psi(t), \tag{3.24}$$

where δ is the learning rate. Here the weights are updated in proportion to the current error multipled by the size of its contribution to the output via the transformed input $\psi(t)$. As $\psi(t)$ is sparse, only those weights that influence the output are updated, leading to a simple but highly efficient algorithm. After updating, the *a posteriori* network output is

$$\hat{y}(t) = \psi^T(t)\mathbf{w}(t) = \delta\|\psi(t)\|^2 y(t) + (1 - \delta\|\psi(t)\|^2)\hat{y}(t|t-1),$$

where $\|\psi(t)\|^2 = \psi^T(t)\cdot\psi(t)$ and the *a posteriori* output error is

$$e_y(t) = y(t) - \hat{y}(t) = (1 - \delta\|\psi(t)\|^2)e_y(t|t-1), \tag{3.25}$$

where $e_y(t|t-1) = y(t) - \hat{y}(t|t-1)$ is a *a priori* error. For stability in learning we require that the *a posteriori* error $e_y(t)$ is less than the *a priori* error $e_y(t|t-1)$. This depends on both the learning rate δ and the Euclidean norm $\|\psi(t)\|^2$ of the transformed input vector (i.e. on the basis function ψ and data), since from (3.25) we have

$$\|e_y(t+1)\| < \|e_y(t)\|, \quad if\ \delta \in (0, \frac{2}{\|\psi(t)\|^2}). \tag{3.26}$$

Note: If the variance of the input signal is large, then $\|\psi(t)\|^2$ is large, leading to slow learning, as δ is small. Small δ also provides insensitivity to model mismatch and measurement noise.

3.4.2 Normalised least mean squares learning

The dependence of the learning rate δ on the norm of the transformed input $\psi(t)$, can be removed by setting

$$\delta(t) = \frac{\delta'}{\|\psi(t)\|^2}, \tag{3.27}$$

giving the normalised least mean squares (NLMS) weight update law:

$$\triangle\mathbf{w}(t) = \frac{\delta' e_y(t)\psi(t)}{\|\psi(t)\|^2}, \quad \delta' \in [0,2]. \tag{3.28}$$

The normalisation term means that information is always stored when $\delta' = 2$ (*a posteriori* error=0). However, the NLMS law (3.28) no longer minimises the MSE, but rather a normalised version of it, $E[\frac{e_y^2(t)}{\|\psi(t)\|^2}]$. But if there exists a unique weight vector such that $e_y(t) = 0$, $\forall t$ (i.e. no modelling error), or if $\|\psi(t)\|^2$ is constant, then the LMS and NLMS lead to the same optimal weight vector, otherwise their minima occur at different points in the weight space. The rate of convergence of the NLMS algorithm and its condition depend on the normalised autocorrelation matrix $\bar{R}$.

The NLMS learning law (3.28) is a special case ($r = 2$) of the generalised NLMS rule [10] that satisfies the following conditions: find a $\mathbf{w}$ such that

$$\hat{y}(t) = \psi^T(t)\mathbf{w}(t), \quad \|\Delta\mathbf{w}(t)\|_r \text{ is minimised} \tag{3.29}$$

for different values of r, $1 \leq r < \infty$. The general solution to (3.29) is

$$\Delta\mathbf{w}(t) = \delta\frac{e_y(t)\mathbf{s}[\psi(t)]}{\psi^T(t)\mathbf{s}[\psi(t)]}, \tag{3.30}$$

where $\mathbf{s}[\cdot]$ is a modified search direction such that

$$s_i(\psi) = \begin{cases} |\psi_i|^{q-1}sgn(\psi_i) & \text{if } 1 \leq q < \infty, \\ \delta_{i,k} & \text{if } q = \infty, \end{cases}$$

where $\delta_{i,k}$ is the Kronecker delta function, $k = \arg\max_j |\psi_j|$, and q is the value that satisfies $(1/r + 1/q) = 1$.

Special cases of the generalised NLMS law (3.30) include $r = 2$ (Euclidean norm, see (3.28)); $r = 1$ (the max-NLMS rule):

$$\Delta w_i = \delta\frac{e_y(t)}{\psi_k(t)}\delta_{i,k},$$

and $r = \infty$ (the sgn-NLMS rule):

$$\Delta w_i = \delta\frac{e_y(t)sgn[\psi(t)]}{\|\psi(t)\|_1}.$$

It is easily seen that the *a posteriori* error is always zero for these learning laws (with $\delta = 1$), so that the only difference between them is how they search the weight space. The weight updates in the weight space for the three learning rules are shown in Figure 3.2. The max-NLMS always updates the weight vector parallel to an axis (with largest error), the sgn-NLMS rule causes the update rule to be set at 45 degrees to an axis, whereas the standard ($r = 2$) NLMS rule always projects the weight vector perpendicularly onto the solution hyperplane (for $\delta = 1$).

3.4.3 NLMS weight convergence

Consider now the weight error $\mathbf{e_w}(t) = \hat{\mathbf{w}} - \mathbf{w}(t)$. From (3.30), an iterative equation in $\mathbf{e_w}(t)$ follows as:

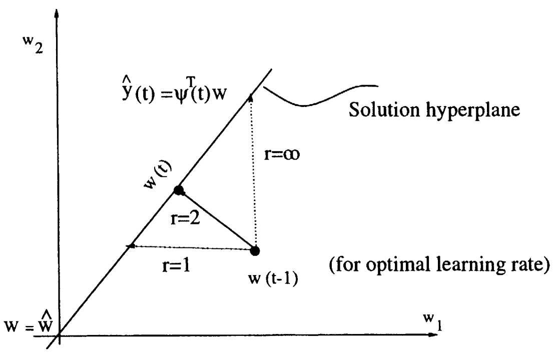

Fig. 3.2. Weight updates in the weight space

$$\mathbf{e_w}(t) = [I - \delta \frac{\mathbf{s}^T(t)\psi(t)}{\psi^T(t)\mathbf{s}(t)}]\mathbf{e_w}(t-1). \tag{3.31}$$

Denoting the matrix on the right hand side by $P(t)$, then the *a posteriori* weight error is

$$\|\mathbf{e_w}(t)\|_r = \|P(t)\mathbf{e_w}(t-1)\|_r \leq \|P(t)\|_r \|\mathbf{e_w}(t-1)\|_r \tag{3.32}$$

when $\delta = 1$, $P(t)$ is a projection matrix or operator [61], as:

$$P(P(\mathbf{e_w}(t-1))) = P(\mathbf{e_w}(t-1)),$$

whose structure determines the stability and convergence properties of different learning laws.

For the standard NLMS training algorithm, the projection operator is given by

$$P(t) = I - \delta \frac{\psi(t)\psi^T(t)}{\|\psi(t)\|_2^2},$$

from which it is easy to show that it has $(p-1)$ unity eigenvalues whose corresponding eigenvectors are orthogonal to $\psi(t)$, and one eigenvalue of $(1-\delta)$ whose corresponding eigenvector is $\psi(t)$. The magnitudes of the weight errors from a strictly non-increasing sequence as $\|P(t)\|_2 = 1$, and when the weight error vector has a non-zero component that is parallel to $\psi(t)$, $\|\mathbf{e_w}\|_2$ will decrease.

For the max-NLMS learning algorithm, the corresponding projection matrix is equivalent to an identity matrix except for the kth row which corresponds to the weight that is being updated. In this case the projection matrix kth row is

$$[-\delta\frac{\psi_1(t)}{\psi_k(t)}, \ldots, -\delta\frac{\psi_{k-1}(t)}{\psi_k(t)}, 1-\delta, -\delta\frac{\psi_{k+1}(t)}{\psi_k(t)}, \ldots, -\delta\frac{\psi_p(t)}{\psi_k(t)}].$$

Therefore in order that $\|P(t)\|_\infty \leq 1$, and that the weight error is a non-increasing function of t, then the absolute sum of elements of the kth row of

P must sum to ≤ 1 [112]. Therefore the input training pattern must be input or diagonally dominant, i.e.

$$|\psi_k(t)| \geq \sum_{i=1, i\neq k}^{p} |\psi_i(t)|,$$

which is independent of the learning rate δ which must also be such that $\delta \in [0,1]^2$. Whilst the input dominance condition is sufficient to guarantee convergence, but this is not always necessary, as the matrix norm bound is always a worst case analysis.

For the sgn-NLMS training law, the projection matrix kth row is given by

$$[-\delta\frac{\psi_1(t)sgn(\psi_k(t))}{\|\psi(t)\|_1}, ..., 1-\delta\frac{\psi_k(t)sgn(\psi_k(t))}{\|\psi(t)\|_1}, ..., -\delta\frac{\psi_p(t)sgn(\psi_k(t))}{\|\psi(t)\|_1}],$$

whose norm is given by [112]

$$\|P(t)\|_1 = 1 + (p-2)\delta\frac{\|\psi(t)\|_\infty}{\|\psi(t)\|_1} \leq 1 + (p-2)\delta. \tag{3.33}$$

Therefore for any three (or higher) input dimensional network, the sgn-NLMS algorithm may be unstable for any non-zero value of the learning rate and any type of (non-zero) input vector. Whether or not the algorithm is unstable depends on the signs and magnitudes of the components of the current weight error vector. For input signals that are input dominant, then the sgn-NLMS algorithm is potentially more unstable as $\|\psi(t)\|_\infty/\|\psi(t)\|_1$ is closer to unity.

In practice convergence depends on the properties of the transformed input process $\psi(t)$. For example, if $\psi(t)$ is not persistently exciting, but say settles to a constant value or varies slowly, then the standard NLMS algorithm will not converge to its optimal solution. The max-NLMS and sgn-NLMS perform even worse, for example, the sgn-NLMS does not converge in any sense when all the components of $\psi(t)$ are positive (or negative), equally the max-NLMS cannot converge when an independent element of $\psi(t)$ never attains the maximum value. For statistical signals we can consider convergence in the mean squared sense, i.e. if $\lim_{t\to\infty} E[\mathbf{e}_\mathbf{w}^T(t)\mathbf{e}_\mathbf{w}(t)] \to 0$.

From (3.31) for all three learning laws,

$$\begin{aligned}\mathbf{e}_\mathbf{w}^T(t)\mathbf{e}_\mathbf{w}(t) = \mathbf{e}_\mathbf{w}^T(t-1)[\ I\ &-\delta\frac{\psi(t)\mathbf{s}^T(t)}{\psi^T(t)\mathbf{s}(t)} - \delta\frac{\mathbf{s}(t)\psi^T(t)}{\psi^T(t)\mathbf{s}(t)} \\ &+ \delta^2\frac{\psi(t)\mathbf{s}^T(t)\mathbf{s}(t)\psi^T(t)}{(\psi^T(t)\mathbf{s}(t))^2}]\mathbf{e}_\mathbf{w}(t-1).\end{aligned}$$

Denote $v^2(t) = E[\mathbf{e}_\mathbf{w}^T(t)\mathbf{e}_\mathbf{w}(t)]$ as the weight error variance, then from the above

$$v^2(t) = (1 - 2p^{-1}\delta + \beta\delta^2)v^2(t-1), \tag{3.34}$$

where

$$\beta = \begin{cases} \beta_{standard\ NLMS} = p^{-1} \\ \beta_{sgn-NLMS} = E[\frac{\psi^T(t)\psi(t)}{\psi^T(t)sgn(\psi(t))^2}] \\ \beta_{max-NLMS} = p^{-1}E[\frac{\psi^T(t)\psi(t)}{\psi_k^2(t)}] \end{cases}. \tag{3.35}$$

Clearly for convergence in the weight error variance we require that

$$0 < r = (1 - 2p^{-1}\delta + \beta\delta^2) < 1,$$

from which the optimal learning rate is

$$\delta^* = (p\beta)^{-1} \text{ with } r^* = (1 - p^{-1}\delta^*). \tag{3.36}$$

Generally

$$\delta^*_{standard\ NLMS} = 1 \geq \delta^*_{sgn-NLMS}, \delta^*_{max-NLMS},$$

equally

$$r^*_{sgn-NLMS}, r^*_{max-NLMS} \geq r^*_{standard\ NLMS} = (1 - p^{-1}).$$

To evaluate the optimal learning rate and region of convergence for the modified NLMS algorithms, the probability distribution $p(\psi_i)$ is necessary, so for example take $p(\psi_i) = \alpha^{-1}\exp(-|\psi_i|/\alpha)$ (a Laplace distribution), it can be shown [10] that

$$\beta_{sgn-NLMS} = \frac{2}{(p+1)}, \quad \delta^*_{sgn-NLMS} = \frac{(p+1)}{2p}.$$

So that the convergence rate for p large for the sgn-NLMS is approximately twice that of the standard NLMS algorithm.

If the model (3.2), $\hat{y}(t) = \psi^T(t)\mathbf{w}$, is also subject to additive white noise $e(t)$ with variance σ_ξ^2, then the above variance analysis can be repeated to give [10]

$$v^2(t) = (1 - 2p^{-1}\delta + \beta\delta^2)v^2(t-1) + \delta^2\gamma\sigma_\xi^2, \tag{3.37}$$

where

$$\gamma = \begin{cases} \gamma_{standard\ NLMS} = E[(\psi^T(t)\psi(t))^{-1}] \\ \gamma_{sgn-NLMS} = pE[(\psi^T(t)sgn(\psi(t)))^{-1}] \\ \gamma_{max-NLMS} = E[\psi_k^{-1}(t)] \end{cases}, \tag{3.38}$$

so that as $t \to \infty$, we have the weight error variance

$$v^2 = p\sigma_\xi^2\gamma\frac{\delta}{(2 - \beta\delta p)}. \tag{3.39}$$

So that after an initial transient convergence all three NLMS learning algorithms will converge to a mean weight value of $\hat{\mathbf{w}}$, but will then 'jitter' around the mean value with a variance v^2. This region is called a minimal capture zone [32]. For a specific input distribution, the size of the minimal capture zone can be minimised by selecting a small value of δ, but this in turn decreases the rate of convergence.

The above modified NLMS learning algorithms are computationally efficient as they search the weight space in orthogonal directions, however they introduce serious problems of learning stability and associated rate of convergence. Only the standard NLMS is unconditionally stable, updating weight vectors parallel to the input vectors.

3.4.4 Recursive least squares estimation

Equation (3.2) can be rewritten for all the data D_N as

$$\mathbf{y} = \mathbf{A}\mathbf{w} + \mathbf{e},$$

where $\mathbf{e} = \mathbf{y} - \mathbf{A}\mathbf{w}$, whose least mean squared solution is given by (3.7) at the tth data point as

$$\mathbf{w}(t) = (\mathbf{A}_t^T \mathbf{A}_t)^{-1}\mathbf{A}_t^T \mathbf{y}_t$$

for $\mathbf{A}_t = [\psi(\mathbf{x}(1)), ..., \psi(\mathbf{x}(t))]^T$, $\mathbf{y}_t = [y(1), ..., y(t)]^T$. Rather than solve this equation as a single shot or batch solution it can be more efficiently solved iteratively by progressing through the data set D_N sequentially.

Suppose that at iteration $t+1$, the new data is $y(t+1)$ and $\psi(\mathbf{x}(t+1)) \equiv \psi(t+1)$ (where $\mathbf{x}(t+1)$ is the observed regressor vector at $t+1$). Hence

$$\begin{aligned}\mathbf{w}(t+1) = &\left(\begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix}^T \begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix} \right)^{-1} \\ &\times \begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix}^T \begin{bmatrix} \mathbf{y}_t \\ y(t+1) \end{bmatrix}. \end{aligned} \tag{3.40}$$

Define $P(t) = (\mathbf{A}_t^T \mathbf{A}_t)^{-1}$, then

$$\begin{aligned} &P(t+1) \\ &= \left(\begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix}^T \begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix} \right)^{-1} \qquad (3.41) \\ &= [\mathbf{A}_t^T \mathbf{A}_t + \psi(t+1)\psi^T(t+1)]^{-1} \\ &= [P^{-1}(t) + \psi(t+1)\psi^T(t+1)]^{-1} \\ &= P(t) - P(t)\psi(t+1)[1 + \psi^T(t+1)P(t)\psi(t+1)]^{-1}\psi^T(t+1)P(t) \\ &= P(t) - \frac{P(t)\psi(t+1)\psi^T(t+1)P(t)}{1 + \psi^T(t+1)P(t)\psi(t+1)}. \end{aligned}$$

This iterative equation for $P(t)$ has been derived by the matrix inversion lemma [87]. Hence the weight update can be rewritten as

$$\begin{aligned} \mathbf{w}(t+1) &= P(t+1)[\mathbf{A}_t^T \mathbf{y}_t + \psi(t+1)y(t+1)] \qquad (3.42) \\ &= P(t+1)[P^{-1}(t)\mathbf{w}(t) + \psi(t+1)y(t+1)] \\ &= \mathbf{w}(t) + P(t+1)\psi(t+1)[y(t+1) - \psi^T(t+1)\mathbf{w}(t)]. \end{aligned}$$

That is, the new estimate is the old estimate plus a correction term.
Note: Assuming that $\mathbf{e} \sim N(0, \sigma^2 I)$, then it can be easily shown that:

- $P(t)$ is proportional to the estimator covariance matrix $\text{cov}(\mathbf{w})$.
- Equations (3.41) and (3.42) form a Kalman filter type estimator [50] (see also Chapter 8)
- $\hat{\mathbf{w}}$ is an unbiased estimate.
- $\hat{\mathbf{w}}$ is a consistent estimate.
- The above least mean squares estimator is also a maximum likelihood estimator.
- This recursive algorithm requires no matrix inversion, utilises only the last iterative values of $\mathbf{w}(t)$, $P(t)$, and therefore is computationally efficient.

3.5 Gradient noise and normalised condition numbers

The condition of the autocorrelation matrix R not only determines the rate of convergence of batch gradient based weight training algorithms, it also influences the rate of convergence of the instantaneous LMS and NLMS learning rules as well. As we have seen above a full theoretical comparison of even simple learning laws such as LMS, NLMS is difficult since they depend strongly on the particular data input distribution used to train the network, so these results usually assume that statistically independent inputs are drawn randomly from a Gaussian input distribution with zero-mean.

The instantaneous LMS (or NLMS) rule typically uses the instantaneous gradient $\hat{\nabla}$, rather than the batch performance gradient ∇, introducing an error

$$\mathbf{n}(t) = \nabla(t) - \hat{\nabla}(t).$$

This difference in gradients is called gradient noise, so that the instantaneous gradient (see (3.23)) can be written as

$$e_y(t)\psi(t) = E[e_y(t)\psi(t)] + \mathbf{n}(t).$$

The gradient noise covariance matrix $\mathbf{N}(t) = E[\mathbf{n}(t)\mathbf{n}^T(t)]$ is therefore

$$\mathbf{N}(t) = E[\psi(t)e_y(t)e_y(t)\psi^T(t)] - E[e_y(t)\psi(t)]E[e_y(t)\psi^T(t)].$$

This noise covariance expression describes the relative spread of the gradient noise components and is important for analysing the relative convergence rates for stochastic inputs for LMS (NLMS) learning laws. However this expression contains a forth order moment term which is difficult to analyse, but assuming that ψ_i's are stationary, Gaussian distributed, and their zero mean components are statistically independent over t iterations, then it can be simplified [4] by using the real Gaussian factoring theorem to

$$\mathbf{N} = R\mathbf{e}_\mathbf{w}\mathbf{e}_\mathbf{w}^T R^T + R(v_{min} + \mathbf{e}_\mathbf{w}^T R\mathbf{e}_\mathbf{w}) + R\bar{e}_y^2 + R\mathbf{e}_\mathbf{w}(\overline{e_y\psi^T}), \tag{3.43}$$

where $v_{min} = \min_{\mathbf{w}} E(y - \hat{y})^2$. Assuming that ψ and e_y are unbiased then (3.43) simplifies to

$$\mathrm{N} = R\mathbf{e}_{\mathbf{w}}\mathbf{e}_{\mathbf{w}}^T R^T + R(v_{min} + \mathbf{e}_{\mathbf{w}}^T R\mathbf{e}_{\mathbf{w}}). \tag{3.44}$$

Clearly the orientation and magnitude of the gradient noise covariance matrix is a function of the current weight error vector $\mathbf{e}_{\mathbf{w}}$, the autocorrelation matrix R, and the minimum MSE v_{min}. Roughly speaking, the gradient noise is correlated with the steepest principal axis in the weight space accounting for the jittery learning behaviour of the weight vector when updated using LMS (NLMS) as illustrated in Figure 3.3 [24].

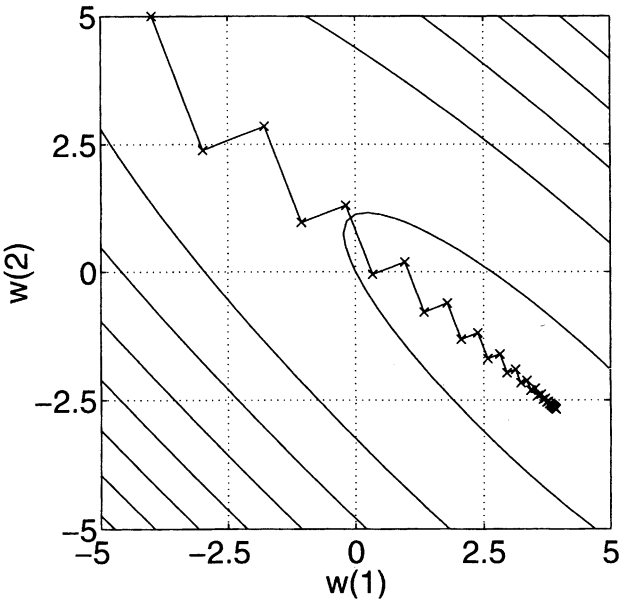

Fig. 3.3. A illustration of the jittery learning behaviour (here the parameter estimate still converges)

Near the optimal weight value $\hat{\mathbf{w}}$, the weight error vector is $\mathbf{e}_{\mathbf{w}} \simeq 0$, this occurs for a well conditioned network, or for very large number of iterations (i.e. $t \longrightarrow \infty$), in this case the gradient noise covariance is

$$\mathrm{N} = Rv_{min}. \tag{3.45}$$

Therefore whenever there exists a modelling error or there is measurement noise ($\sigma^2 \neq 0$), LMS (NLMS) rules do not converge to a point but rather to a domain called the minimal capture zone (see Section 3.4.3 and (3.39)), whose size is dependent upon the autocorrelation matrix R.

Finally (3.22) relates the observable network output error to the network's weight error through the network's condition number $C(R)$. In general the

NLMS algorithm provides faster convergence and better network conditioning than the conventional LMS learning algorithms as $C(R_{NLMS}) \simeq \sqrt{C(R_{LMS})} < C(R_{LMS})$ [5].

3.6 Adaptive learning rates

In the above instantaneous LMS, NLMS learning laws, it has been assumed that the learning rate δ is a positive constant number which satisfies inequalities (3.26) to ensure learning stability. Whenever (in all real situations) there exist modelling mismatch errors, observation noise and gradient noise, then these algorithms will only converge if the learning rate is scheduled against iterations to filter out these noise influences as well as minimising minimal capture domains. There are various methods for achieving this, e.g. assign an individual learning rate, δ_i, to each basis function and reduce δ_i over time as the belief in a particular weight value increases. This approach was originated as the Robbins–Munro stochastic approximation algorithm for which the necessary conditions for convergence on the learning rates, δ_i, associated with the ith basis function are:

$$\begin{aligned} \delta_i(t) &> 0 \\ \lim_{t\to\infty} \delta_i(t) &= 0 \\ \textstyle\sum_{t=1}^{\infty} \delta_i(t) &= \infty \\ \textstyle\sum_{t=1}^{\infty} \delta_i^2(t) &< \infty. \end{aligned}$$

The first condition is to ensure that the weight change direction is towards the hyperplane solution, the second condition ensures that the algorithm terminates asymptotically, the third condition implies that $\delta_i(t) \to 0$ and the final condition ensures that finite energy is used in achieving learning. One such condition that satisfies all these conditions is

$$\delta_i(t) = \frac{\delta_1}{(1 + t_i/\delta_2)},$$

where $\delta_1, \delta_2 > 0$ and t_i is the number of times that the ith basis function has been updated. The stochastic NLMS learning algorithm is of the form

$$\triangle w_i(t) = \delta_i \frac{\psi_i(t)}{\|\psi(t)\|_2^2} e_y(t).$$

In this chapter we have introduced the basic theory of least squares parametric learning laws for both batch and on-line training. Whilst high order learning algorithms are possible they are unnecessary for linear-in-the-parameters networks. Throughout the remainder of this book we utilise either the normalised least mean squares algorithm of Section 3.4.2 or recursive least squares estimators of Section 3.4.4 (or its Kalman filter derivatives).