Adaptive Linear Modelling

The model of a linear model at time t is given by:

$$y(t) = \sum_{i=1}^{p} a_i(t) w_i(t-1) = \mathbf{a}(t)\mathbf{w}_i(t-1)$$

where $\mathbf{a}(t)$ and $\mathbf{w}(t)$ are p dimensional vectors.

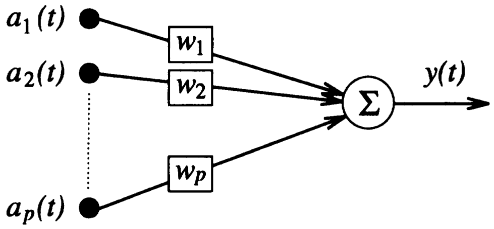


Figure 4.1 A basic linear model.

Performance of the model

- The instantaneous output error is:

$$\epsilon_y(t) = \hat{y}(t) - y(t).$$

- Some possible *Performance* functions for adapting the weight vector are:

$$J = \begin{cases} E(|\epsilon_y(t)|) \\ E(\epsilon_y^2(t)) \\ \max_t |\epsilon_y(t)| \end{cases}$$

- where the expectation operator $E()$ is taken over t for the training set $\{x(t), \hat{y}\}_{t=1}^{L}$

Mean Squared Error Performance

- The mean squared error performance is defined as:

$$J = E(\epsilon_y^2(t));$$

- the instantaneous network output error:

$$\epsilon_y(t) = \hat{y} - y(t) = \hat{y}(t) - \mathbf{a}^T(t)\mathbf{w},$$

- squaring $\epsilon_y(t)$:

$$\epsilon_y^2(t) = \hat{y}^2(t) + \mathbf{w}^T \mathbf{a}(t)\mathbf{a}^T(t)\mathbf{w} - 2\hat{y}(t)\mathbf{a}^T(t)\mathbf{w},$$

- and taking the expected values over t gives:

$$J = E(\hat{y}^2(t)) + \mathbf{w}^T E(\mathbf{a}(t)\mathbf{a}^T(t))\mathbf{w} - 2E(\hat{y}(t)\mathbf{a}^T(t))\mathbf{w}.$$

Typical MSE performance surface

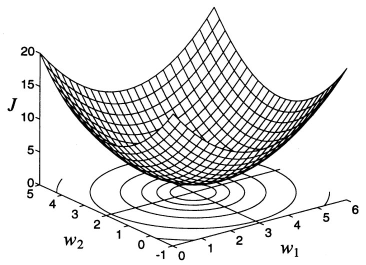


Figure 4.2 A typical MSE performance surface in two-dimensional weight space. The optimal weight vector occurs at $\hat{\mathbf{w}} = (3,2)^T$, and a contour plot is projected onto the base of the graph.

Typical MSE performance surface

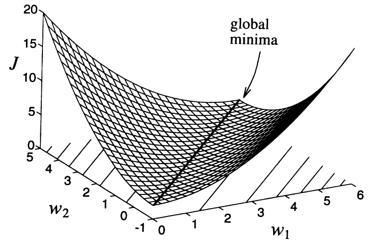


Figure 4.3 A typical *singular* MSE performance surface in two-dimensional weight space. The candidate optimal weight vectors occur along the line $w_2 = w_1 - 1$, and the contour plot is projected onto the base of the graph.

The *Autocorrelation* matrix $\mathbf{R}$ can be defined as:

$$\begin{aligned}\mathbf{R} &= E(\mathbf{a}(t)\mathbf{a}^T(t)) \\ &= \begin{bmatrix} E(a_1^2(t)) & E(a_1(t)a_2(t)) & \dots & E(a_1(t)a_p(t)) \\ E(a_2(t)a_1(t)) & E(a_2^2(t)) & \dots & E(a_2(t)a_p(t)) \\ \vdots & \vdots & \vdots & \vdots \\ E(a_p(t)a_1(t)) & E(a_p(t)a_2(t)) & \dots & E(a_p^2(t)) \end{bmatrix}\end{aligned}$$

The *Cross-correlation* vector $\mathbf{p}$ can be defined as:

$$\begin{aligned}\mathbf{p} &= E(\hat{y}(t)(a)(t)) \\ &= \begin{bmatrix} E(\hat{y}(t)a_1(t)) \\ E(\hat{y}(t)a_2(t)) \\ \vdots \\ E(\hat{y}(t)a_p(t)) \end{bmatrix}\end{aligned}$$

Now, we can define the mean squared error performance as:

$$J = E(\hat{y}^2(t)) + \mathbf{w}^T\mathbf{R}\mathbf{w} - 2\mathbf{p}^T\mathbf{w}.$$

The minimum MSE, J_{min}, which occurs when $\mathbf{w} = \hat{\mathbf{w}}$ can be written as:

$$\begin{aligned}J_{\text{min}} &= E(\hat{y}^2(t)) - \mathbf{p}^T\mathbf{R}^{-1}\mathbf{p} \\ &= E(\hat{y}^2(t) - \mathbf{p}^T\hat{\mathbf{w}}.\end{aligned}$$

$$0 \leq J_{\text{min}} \leq E(\hat{y}^2(t))$$

The MSE can be expressed in a much simpler form, called as *Normal Form of the Performance Surface*:

$$J = J_{\text{min}} + \epsilon_w^T\mathbf{R}\epsilon_w.$$

Normal form of the auto-correlation matrix

The auto-correlation matrix can be decomposed as:

$$\mathbf{R} = \mathbf{Q}\Lambda\mathbf{Q}^{-1} = \mathbf{Q}\Lambda\mathbf{Q}^{\mathbf{T}},$$

where Λ is a $p \times p$ diagonal matrix composed of the non-negative eigenvalues of $\mathbf{R}$ and $\mathbf{Q}$ is a unitary matrix whose columns are the corresponding orthonormal eigenvectors.

Two examples which can reproduce exactly the function $\hat{y} = a + bxt$ defined on the unit interval.

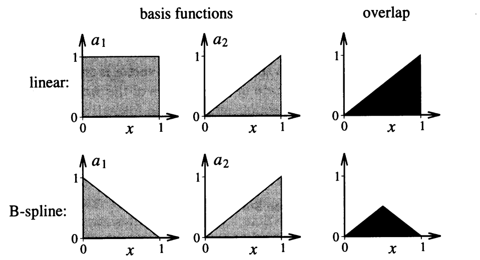


Figure 4.5 Top: the basis functions corresponding to a univariate linear model. Bottom: the basis functions for a B-spline network of order 2. Both networks have the same modelling capability on the unit interval, only the internal representation is different.

Model 1

$$y(t) = a_1(x(t))w_1 + a_2(x(t))w_2;$$
$$a_1(x(t)) = 1.0;$$
$$a_2(x(t)) = x(t);$$

where $x(t)$ has a uniform probability density function on the unit interval.

$$\mathbf{R} = \begin{bmatrix} \int_0^1 1dx & \int_0^1 xdx \\ \int_0^1 xdx & \int_0^1 x^2dx \end{bmatrix} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.333 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} 1.268 & 0 \\ 0 & 0.066 \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} 0.882 & 0.472 \\ 0.472 & -0.882 \end{bmatrix}$$

with $C(\mathbf{R}) = \lambda_{\mathsf{max}}/\lambda_{\mathsf{min}} = 19.3$

Model 2

$$y(t) = a_1(x(t))w_1 + a_2(x(t))w_2;$$
$$a_1(x(t)) = (1 - x(t));$$
$$a_2(x(t)) = x(t);$$

where $x(t)$ has a uniform probability density function on the unit interval.

$$\mathbf{R} = \begin{bmatrix} \int_0^1 (1-x)^2dx & \int_0^1 x(1-x)dx \\ \int_0^1 x(1-x)dx & \int_0^1 x^2dx \end{bmatrix} = \begin{bmatrix} 0.333 & 0.167 \\ 0.167 & 0.333 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.167 \end{bmatrix} \qquad \mathbf{Q} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

with $C(\mathbf{R}) = \lambda_{\mathsf{max}}/\lambda_{\mathsf{min}} = 3$

Decoupling the performance surface

$$J = J_{\mathsf{min}} + \epsilon_w^T \mathbf{Q} \Lambda \mathbf{Q}^T \epsilon_w$$

which can be rewritten as:

$$J = J_{\mathsf{min}} + \mathbf{v}^T \Lambda \mathbf{v}$$
$$= J_{\mathsf{min}} + \sum_{i=1}^{p} \lambda_i v_i^2,$$

where $\mathbf{v} = \mathbf{Q}\epsilon_w$

- The contour projection of J is a hyperellipsoid in v-space, with a minimum occurring at the origin and with p mutually orthogonal lines which are perpendicular to all contours of J.
- These lines are known as the principle axes of the ellipse, and they are simply the eigenvectors of the autocorrelation matrix.
- Also, $\mathbf{Q}$ is a unitary matrix, and so $\mathbf{v}$ is simply a rotated version of ϵ_w in w space, with an origin $\hat{\mathbf{w}}$.

The eigenvalue of $\mathbf{R}$, which are contained in the diagonal matrix Λ, represent the second derivative of J along any of the principle axes as:

$$\frac{\partial^2 J}{\partial v_i^2} = 2\lambda_i,$$

for $i = 1, \ldots, p$.

- The second proportional derivatives of J are proportional to the corresponding eigenvalues of the autocorrelation matrix.
- They also contain *curvature information* about the performance function, and so if the eigenvalues of the autocorrelation matrix is widelyy spread, this is reflected in the relative steepness of the performance along the principal axes. The performance surface is steepest along the major principal axis and is flattest along the minor principal axis, and the ratio of these two quantities (condition number) determines the rate of convergence of the gradient descent rules.

Two examples of the normalised performance surface

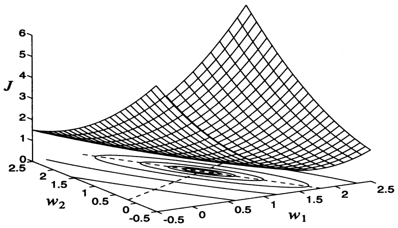


Figure 4.6 A contour plot of the MSE performance surface for a conventional linear model. The contour values are $J = 0.005, 0.015, 0.05, 0.15, 0.5, 1.5$, $\hat{w} = (1,1)^T$ and the two dashed lines are the principal axes (eigenvectors).

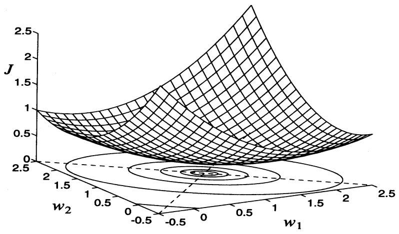


Figure 4.7 A contour plot of the MSE performance surface for an order 2 B-spline model. The contour values are $J = 0.005, 0.015, 0.05, 0.15, 0.5, 1.5$, $\hat{w} = (1,1)^T$, and the two dashed lines are the principal axes (eigenvectors).

Gradient descent learning algorithm

The gradient of J in the w space can be defined as:

$$\bigtriangledown = \frac{\partial J}{\partial \mathbf{w}} 2\mathbf{R}\mathbf{w} - 2\mathbf{p}.$$

Or alternatively as:

$$\bigtriangledown = -2\mathbf{R}\epsilon_w,$$

where $\epsilon_w = \hat{\mathbf{w}} - \mathbf{w}$ is the error in the weight vector.

Or more:

$$\bigtriangledown = -2E(\epsilon_y(t)\mathbf{a}(t)).$$

The gradient in the transformed weight space is given by:

$$\frac{\partial J}{\partial \mathbf{v}} = 2\Lambda \mathbf{v},$$

or

$$\frac{\partial J}{\partial v_i} = 2\lambda_i v_i,$$

thus the gradient o J with respect to v_i only depends on the eigenvalue λ_i and v_i. The gradient components, in v-space are decoupled.

Gradient descent learning rules

$$\triangle \mathbf{w}(t-1) = -\frac{\delta}{2} \bigtriangledown (t),$$

where $\triangle \mathbf{w}(t-1) = \mathbf{w}(t) - \mathbf{w}(t-1)$, substituting for $\bigtriangledown(t)$ gives:

$$\begin{aligned} \triangle \mathbf{w}(t-1) &= -\delta(\mathbf{R}\mathbf{w}(t-1) - \mathbf{p}), \\ &= -\delta \mathbf{R}\epsilon_w(t-1). \end{aligned}$$

Thus the weight change depends on the structure of the autocorrelation matrix and the current weight error vector.

The gradient descent algorithm can be interpreted as a feedback model as can be seen on the following figure, where the weights are states with the current performance of the model being fed back to modify the model. From this interpretation it is obvious that the size of δ determines the stability of the closed loop system and too high a value causes unstable learning.

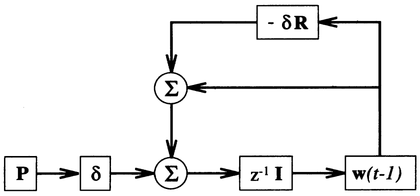


Figure 4.11 Feedback representation of gradient descent adaptation.

Parameter convergence for different auto-correlation matrices - 1

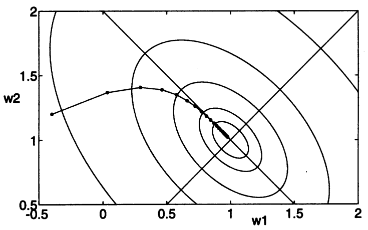


Figure 4.8 Gradient descent on the performance surface corresponding to a B-spline model, $(C(\mathbf{R}) = 3)$, after twenty weight updates with a normalised learning rate of 0.5. The performance surface's contours correspond to $J = 0.005, 0.015, 0.05, 0.15, 0.5$.

Parameter convergence for different auto-correlation matrices - 2

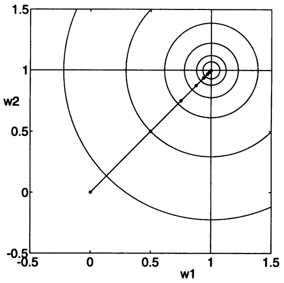


Figure 4.9 Gradient descent on a performance surface with $C(\mathbf{R}) = 1$ after ten weight updates with a normalised learning rate of 0.5. The performance surface's contours correspond to $J = 0.005, 0.015, 0.05, 0.15, 0.5, 1.5$.

Parameter convergence for different auto-correlation matrices - 3

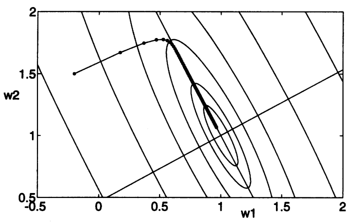


Figure 4.10 Gradient descent on the performance surface corresponding to a linear model ($C(\mathbf{R}) = 19.3$), after a hundred weight updates with a normalised learning rate of 0.5. The performance surface's contours correspond to $J = 0.005, 0.015, 0.05, 0.15, 0.5, 1.5$.

Convergence and stability analysis

The gradient descent rule can be written as:

$$\triangle\mathbf{w}(t-1) = -\delta\mathbf{R}\epsilon_w(t-1),$$

or

$$\epsilon_w(t) = (\mathbf{I} - \delta\mathbf{R})\epsilon_w(t-1).$$

Because this relationship is difficult to interpret, since $\mathbf{R}$ is not diagonal, it can transformed to the decoupled form:

$$\triangle\mathbf{v}(t-1) = -\delta\Lambda\mathbf{v}(t-1),$$

which can be rewritten in its decoupled components as:

$$v_i(t) = (1 - \delta\lambda_i)v_i(t-1) \qquad \text{for } i = 1, 2, ..., p,$$

with closed form solutions:

$$v_i(t) = (1 - \delta\lambda_i)^t v_i(0) \qquad \text{for } i = 1, 2, ..., p,$$

where $v_i(0)$ is the rotated i^{th} initial error in the weight vector.

Convergence and stability analysis

For stable learning, it is required that:

$$|v_i(t)| \leq |v_i(0)| \qquad \forall i = 1, 2, \ldots, p$$

or equivalently

$$|1 - \delta\lambda_i| < 1.$$

Therefore learning is stable if and only if the learning rate δ satisfies:

$$0 < \delta < \frac{2}{\lambda_{\max}}$$

Rate of convergence

By considering a new learning rate defined by $\delta_\lambda = \delta/\lambda_{\text{max}}$ then stable learning is assured if and only if:

$$0 < \delta < 2$$

An exponential decay curve can be fitted to the discrete values of $v_i(t)$ as shown in the figure. This curve has an initial value $v_i(0)$ and a decay constant τ_i defined by:

$$v_i(0) \exp\left(-\frac{t}{\tau_i}\right) = v_i(0)(1 - \delta_\lambda \lambda_i)^t.$$

Solving for τ_i gives:

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

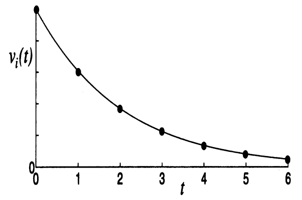


Figure 4.12 Convergence of the i^{th} natural mode, $v_i(t)$, which is adapted using a gradient descent rule with $(1 - \delta\lambda_i) < 1$ (an exponential curve has been fitted to the discrete points).

The largest time constant τ_{max} corresponds to the smallest eigenvalue and it is given by:

$$\tau_{\text{max}} = \frac{-1}{\ln(1 - \delta\lambda_{\text{min}}/\lambda_{\text{max}})}, \quad = \frac{-1}{\ln(1 - \delta/C(\mathbf{R}))}$$

where $C(\mathbf{R}) = \lambda_{\text{max}}/\lambda_{\text{min}}$.

Networks with large condition numbers have components of the weight vector which posses very large time constants and they learn the information which lies along the minor principal axis very slowly.

The condition of the matrix is not allowed to be infinite, as the zero eigenvalues do not influence the rate of convergence; they only influence the actual values of the weights.

Condition of linear models

Consider the following linear model:

$$y(t) = a_1(t)w_1(t-1) + a_2(t)w_2(t-1);$$
$$a_1(t) = 1;$$
$$a_2(t) = x(t);$$

and it is defined on the interval $[a, b]$. The corresponding auto-correlation matrix is:

$$\mathbf{R} = \begin{bmatrix} \int_a^b p(x)dx & \int_a^b xp(x)dx \\ \int_a^b xp(x)dx & \int_a^b x^2 p(x)dx \end{bmatrix},$$

for a given probability function $p(x)$ on $[a, b]$.

The eigenvalues of $C(\mathbf{R})$ vary according to the values of a and b and the associated probability density function.

The simplest probability density function is a uniform distribution on the interval $[a, b]$ given by:

$$p(x) = \begin{cases} 1/(b-a) & \text{if } x \in [a, b], \\ 0 & \text{otherwise} \end{cases}$$

Linear models defined on positive intervals

The condition of a linear model is now derived for training data which have a uniform probability density function on the interval $[0, b]$. This produces an autocorrelation matrix of the form:

$$\mathbf{R} = \begin{bmatrix} b^{-1}\int_0^b 1dx & b^{-1}\int_0^b xdx \\ b^{-1}\int_0^b xdx & b^{-1}\int_0^b x^2 dx \end{bmatrix}.$$

Linear models defined on positive intervals

When b is close to zero, the power of the bias term dominates the other terms in $\mathbf{R}$ and the network is slow to learn the information contained in the linear term.

As b increases the power of the linear term increases and the eigenvalues move closer together until $b = \sqrt{3}$. At this point, the power of the bias term is equal to the power of the linear term, so $\mathbf{R}$ is symmetrical with equal diagonal elements (although the off-diagonal elements are significant). Then, as b increases still further, the power of the linear term starts to dominate the other elements of $\mathbf{R}$.

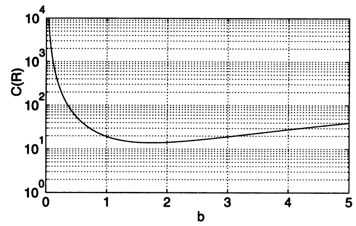


Figure 4.15 Condition of the autocorrelation matrix for the linear model, when the input is defined on the interval $[0, b]$.

Linear models defined on symmetric intervals

The condition of a linear model is now derived for training data which have a uniform probability density function on the interval $[-b, b]$. This produces an autocorrelation matrix of the form:

$$\mathbf{R} = \begin{bmatrix} (2b)^{-1}\int_{-b}^{b} 1dx & (2b)^{-1}\int_{-b}^{b} xdx \\ (2b)b^{-1}\int_{-b}^{b} xdx & (2b)^{-1}\int_{-b}^{b} x^2dx \end{bmatrix},$$
$$= \begin{bmatrix} (2b)^{-1}\int_{-b}^{b} 1dx & (2b)^{-1}0 \\ 0 & (2b)^{-1}\int_{-b}^{b} x^2dx \end{bmatrix}.$$

The off-diagonal elements are zero because the integrands are anti-symmetric functions and the interval is symmetric.

There is a unique minimum at $b = \sqrt{3}$ where $C(\mathbf{R}) = 1$, the two eigenvalues are equal since $\mathbf{R}$ is diagonal with equal diagonal terms. Therefore to make a network as well conditioned as possible, the autocorrelation matrix should be nearly diagonal with terms of equal magnitude.

Linear models defined on symmetric intervals

If it is required that $C(\mathbf{R}) = 1$, then instead of integrating over $[-\sqrt{3}, \sqrt{3}]$, the bias output could be set to be $1/\sqrt{3}$ and the integral taken over the interval $[-1, 1]$ since this produces the same result.

It should also be noted that for any input probability density function the power of a unity bias term is always greater than or equal to power of the linear term, which is defined on the interval $[-1, 1]$. Hence the condiion of the basic models can be drastically improved by choosing the size of the bias term appropriately and by altering the domain of the linear term.

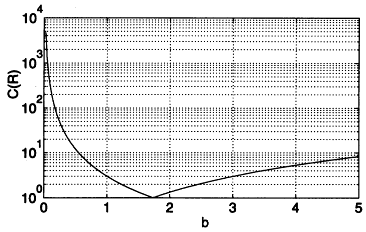


Figure 4.16 Condition of the autocorrelation matrix for a linear model, when the input is defined on the interval $[-b, b]$.

Orthogonal basis functions

It has been shown that a network is well conditioned if the diagonal elements of the autocorrelation matrix are approximately equal and the off-diagonal elements are almost zero as this results in a condition number of almost unity.

The requirements that the off-dignoal elements must be zero means that the inner product:

$$(a_i, a_j) = \int_a^b a_i(x)a_j(x)p(x)dx = 0 \qquad \text{for } i \neq j;$$

and the requirement that the diagonal terms have equal power means that the inner product:

$$(a_i, a_j) = \int_a^b a_i(x)a_j(x)p(x)dx = c \qquad \text{for } i = 1, \ldots, p;$$

Two functions whose inner product is identically zero are said to be orthogonal on the interval $[a, b]$. If in addition $(a_i, a_j) = 1, \forall i$, the functions are said to be orthonormal. Therefore, when the set of basis functions $a_{i\,i=1}^{\,p}$ forms an orthonormal set, the network is well conditioned for gradient descent learning and one-shot learning can be achieved by setting $\delta = 1/(a_i, a_i)$

Multi-Layer Perceptrons and Back-Propagation

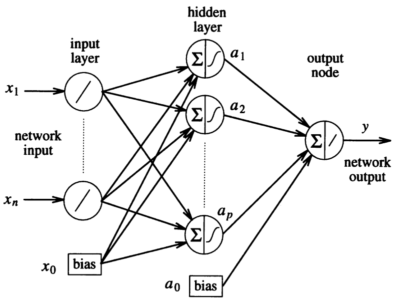


Figure 4.17 Three-layer network with linear input nodes and a linear output node.

$$a_i(t) = \begin{cases} 1 & \text{if } i = 0, \\ f(u_i(t)) & \text{otherwise} \end{cases}$$

where $u_i(t) = \sum_{j=0}^{n} x_j(t) w_{i,j} = \mathbf{x}^T(t)\mathbf{w}_i$ for $i = 1, 2, \ldots, p$.

When the network has an output node with a linear transfer function its output is:

$$y(t) = \sum_{j=0}^{p} a_j(t) w_{0,j} = \mathbf{a}^T(t)\mathbf{w}_0.$$

- Output layer learning:

$$\begin{aligned} \triangle \mathbf{w}_0 &= \delta_0 E(\epsilon_y(t)\mathbf{a}(t)) \\ &= -\delta_0(\mathbf{R}\mathbf{w}_0 - \mathbf{p}), \end{aligned}$$

  where $\mathbf{R}$ is hidden layer autocorrelation matrix, $\mathbf{p}$ is the cross-correlation vector, δ_0 is the learning rate of the output layer and $\triangle \mathbf{w}_0$ denotes the change in the weight vector.

- Hidden layer learning:
  The network output depends nonlinearly on the hidden layer weight vectors, so using the local gradient information gives an update rule of the form:

$$\triangle \mathbf{w}_i = -\delta_i w_{0,i} E\left(\frac{df(u_i(t))}{du}\epsilon_y(t)\mathbf{x}\right),$$

  for $i = 1, 2, \ldots, p$
  where δ_i is the learning rate for the ith-node in the hidden layer, and $w_{0,i}$ is the ith element of the output layer weight vector.

condition of the linear optimisation problems

- the rate of convergence of the weight vector of the output node depends on the condition of the hidden layer autocorrelation matrix $C(\mathbf{R})$ and the rate of adaptation of the hidden layer weights;
- if the weights in the hidden layer are fixed, the exists a stationary global minimun in the output's layer weight space;
- if the hidden layer weights adapt, the global minimum of the output layer is non-stationary and its position in weight space changes as the structure of the hidden layer is updated;
- When the global minimum is stationary, the rate of cconvergence is directly related to $C(\mathbf{R})$, which depends on the interaction between p sigmoidal transfer functions and a single bias node;

Backwards error propagation through a sigmoid

- When the output error is uncorrelated with the derivative of the sigmoid, the hidden layer adaptation can be decomposed into a set of linear optimisation subproblems and the rate of convergence is a function of the condition of these linear subnetworks;
- However, the strength of the output error which is back propagated through the network is directly related to the form of the nonlinear transfer functions used in each node;
- The reason for the usual slow convergence of MLP-s is due to the use of the sigmoidal function which has a small derivative, together with a choice of very small learning rates;
- One way of improving the convergence is the choice sigmoidal functions which have larger derivatives than the usual ones associated with adequate learning rates.

Sigmoidal transfer functions

$$f_1(u) = \frac{1}{1+\exp(-u)} \qquad \in (0,1);$$

$$f_2(u) = \tanh(u) = \frac{1-\exp(-2u)}{1+\exp(-2u)} \qquad \in (-1,1);$$

$$f_3(u) = \left(\frac{2}{1+\exp(-u)}\right) - 1 \qquad \in (-1,1);$$

$$f_2(u) = 2f_1(2u) - 1;$$

$$\frac{df_1(u)}{du} = \frac{\exp(-u)}{(1+\exp(-u))^2} = f_1(u)(1-f_1(u)))$$
$$\in (0, 1/4];$$

$$f_3(u) = 2f_1(u) - 1.$$

Sigmoidal transfer functions

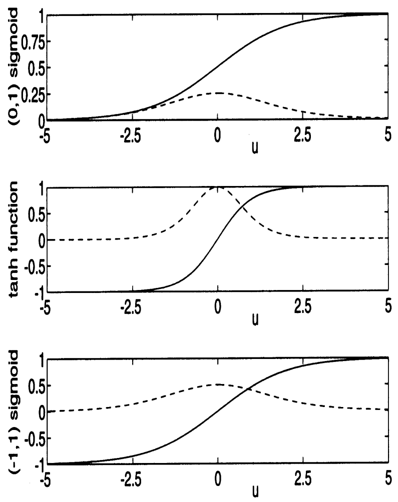


Figure 4.18 Three nonlinear sigmoidal transfer functions (solid lines) and their derivatives (dashed lines).

Example: Single node layer network

Let's suppose that our network consists of a single linear node with a gain of 1/2

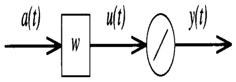


Figure 4.19 Single-layered network with a linear output node.

The weight update can be expressed as:

$$\triangle w = -\frac{\delta}{2}E\left(\frac{d\epsilon_y^2(t)}{dw}\right) = \delta E\left(\frac{dy(t)}{du(t)}\frac{du(t)}{dw}\epsilon_y(t')\right)$$
$$= \frac{\delta}{2}E(\epsilon_y(t')a(t'))$$

The expected *a posteriori* output $\underline{y}$(t), must be evaluated, and is equal to:

$$E(\underline{y}(t')) = E\left(\frac{a(t')}{2}w + \frac{a(t')}{2}\triangle w\right)$$
$$= E\left(y(t') + E\left(\frac{\delta\epsilon_y(t')}{4}\right)\right)$$

for $a(t) = 1.0$

The *a posteriori* output error in terms of the a *priori* output error can be written as:

$$E\left(\epsilon_{\underline{y}}(t')\right) = \left(1 - \frac{\delta}{4}\right)E(\epsilon_y(t')).$$

For this case the a priori plant output error is being scaled by 1/4 which is equal to $E(\left(\frac{dy}{du}\right)^2$.

The learning rate δ must be chosen to compensate this quantity or adaptation can be very slow.

The output layer

- If f is nonlinear no exact scaling factor for the learning rate can be determined to counteract the size of f';
- If it is assumed that the ouput error is not correlated with the derivative of the sigmoid one can use the inverse of the squared derivative $\frac{1}{E(f'(u))^2}$;

$$
\begin{aligned}
&f_1 \to E(f'(u))^2 = \frac{1}{36},\\
&f_2 \to\\
&f_3 \to E(f'(u))^2 = \frac{1}{9}.
\end{aligned}
$$

- If it is correlated then one should use:

$$
\begin{aligned}
&f_1 \to \min(f'(u))^{-2} = 16,\\
&f_2 \to\\
&f_3 \to \min(f'(u))^{-2} = 4.
\end{aligned}
$$

The hidden layer

- learning rate should be multiplied by:

$$[w_{0,i}E(f'(t)a_i'(t))]^{-2}.$$

- together with the inverse of $\lambda_{\max}$;
- output layer must learn faster than the hidden layer;
- they should be chosen to be normalised (sum one), 0.7 and 0.3 for example;

In summary:

- output layer:

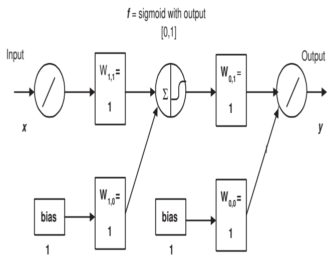
$$\delta_0 = \frac{0.7 * \delta}{\lambda_{\max}[E(f'(u(t)))]^2},$$

- hidden layer:

$$\delta_1 = \min_{i=1}^{p} \frac{0.3\delta}{\lambda_{\max}[w_{0,i}E(a_i' f'(u(t)))]^2}.$$

One example: MLP trained using gradient descent

Consider a desired function that can be modelled exactly by a three-layered network, as shown in the Figure below:



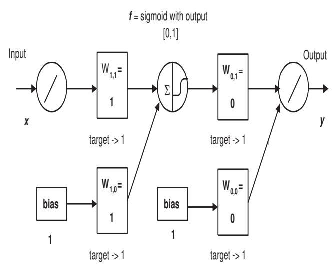
The single input single output network has linear input and ouput node transfer functions, there exists one sigmoidal node in the hidden layer whose outpus lies in the interval (0,1), plus a unit output bias node in each layer and the desired value of each weight is one.

Two networks will be compared

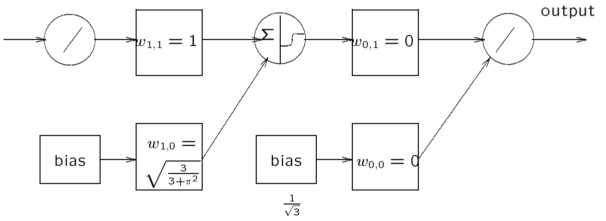
The first one has the same structure as the desired function, with sigmoids whose output lies in (0,1) and unity bias terms. Therefore, all desired weights are equal to unity.

The simulations are started with the hidden layer weight vectors being correct and the ouput layer weights being set to zero.

The training samples are chosen so that a uniform probability density function on the output of the hidden layer sigmoid is produced.



The second network has the same basic structure as the desired network, except that the output of the sigmoid lies in the interval (-1,1) (function f_3) and the outputs of the bias nodes are reduced, so that the power of the bias and the linear/sigmoidal nodes are the same.



The training data are uniformly distributed on the output of the hidden layer node and so the output of the hidden layer bias is $1/\sqrt{3}$.

The probability density distribution of the training data in the input layer is:

$$p(x) = \frac{\exp(-x+1)}{(1+\exp(-x+1))^2}$$

Thus the power of the input linear node is:

$$\int_{-\infty}^{+\infty} \frac{\exp(-x+1)}{1+\exp(-x+1))^2} dx = \pi^2/3,$$

So the output of the input layer bias node is set equal to
$\sqrt{(3+\pi^2)/3}$. Thus the nodes in each layer have the same power, and in the output layer they are orthogonal.

The output layer desired weight vector is given by $\hat{\mathbf{w}}_0 = (1.5/\sqrt{3}, 0.5)^T$), and the desired hidden layer weight vector is $\hat{\mathbf{w}}_1 = (\sqrt{3/(3+\pi^2)}, 1.0)^T$. The inital weight vectors have the correct values for the hidden layer weights and a zero output layer weight vector.

For each network the learning rate for the output layer is set to $0.7/\lambda_{\max}$ and $0.3 * 36/\lambda_{\max}$

The learning history

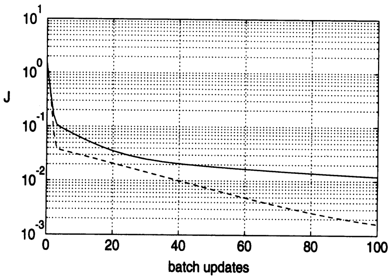


Figure 4.21 MLP initial learning history. The solid line is generated by a standard $(0,1)$ sigmoidal network, whereas the dashed line corresponds to a symmetrical $(-1,1)$ sigmoidal network.

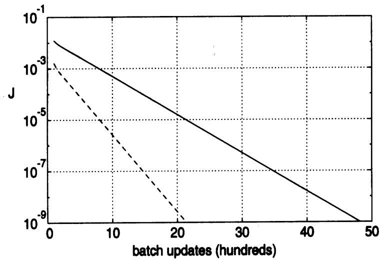


Figure 4.22 MLP final learning history. The faster rate of convergence (dashed line) corresponds to a symmetrical $(-1,1)$ sigmoidal network, whereas the slower rate of convergence (solid line) is generated by a $(0,1)$ sigmoidal network.

The weight update history

Normalised Absolute Weight Errors								
	$(0,1)$ sigmoidal MLP				$(-1,1)$ sigmoidal MLP			
t	$w_{0,0}$	$w_{0,1}$	$w_{1,0}$	$w_{1,1}$	$w_{0,0}$	$w_{0,1}$	$w_{1,0}$	$w_{1,1}$
0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0
10	$1.30e^{-1}$	$2.51e^{-1}$	$9.44e^{-2}$	$1.09e^{-1}$	$4.86e^{-2}$	$4.08e^{-2}$	$3.37e^{-1}$	$5.27e^{-2}$
10^2	$4.60e^{-2}$	$8.25e^{-2}$	$1.36e^{-1}$	$1.65e^{-1}$	$1.94e^{-3}$	$6.81e^{-3}$	$3.07e^{-2}$	$1.38e^{-2}$
10^3	$1.41e^{-3}$	$3.44e^{-3}$	$8.19e^{-3}$	$6.28e^{-3}$	$8.44e^{-7}$	$1.74e^{-5}$	$4.18e^{-5}$	$3.42e^{-5}$

Table 4.2 The absolute value of the normalised errors in the individual weights for a $(0,1)$ and a $(-1,1)$ sigmoidal MLP. The superior rate of weight convergence for the latter network is obvious.