4. Neural Network Classifiers

In this chapter we will utilise the methods from multivariate statistical analysis to investigate the pattern classification algorithms that can be obtained while training artificial neural networks. Our attention will be primarily focused on the similarities and differences between the statistical and neural approaches. For an introduction and a detailed acquaintance with artificial neural networks, the reader is referred to the textbooks of Hertz, Krogh and Palmer (1991), Bishop (1995), Haykin (1999) and others.

In our analysis, particular emphasis will be given to complexity and generalisation. It will be shown that, during training, the perceptron's weights increase along with its complexity. During training of the perceptron, the training is stopped for a moment to allow us to analyse its generalisation and complexity relationships from the statistical point of view. We advocate that the number of parameters of the SLP and MLP classifiers is insufficient to characterise their complexity. We will consider factors that influence the perceptron training process and analyse tools which can be utilised to control its complexity. It will be shown that proper initial values of the perceptron can contain a substantial amount of useful information. In order to save this information one needs to stop training optimally. At the end, we suggest how to use statistical classification algorithms for categorical variables and regularisation techniques to design a "boss" rule (also known as a "combining" or "gating" rule) of the cooperative neural networks.

4.1 Training Dynamics of the Single Layer Perceptron

4.1.1 The SLP and its Training Rule

The non-linear single layer perceptron (SLP) is a greatly simplified mathematical model of an elementary information processing cell in living organisms – a neurone. Following Rumelhart *et al*. (1986), the SLP has been defined as having a number of inputs, $x_1, x_2, \ldots, x_n$, and one output o which is calculated according to Equation (4.1)

$$o = f(\boldsymbol{V}^T\boldsymbol{X} + v_0), \tag{4.1}$$

where v_0, $\boldsymbol{V}^T = (v_1 , v_2 , \ldots , v_n)$ are weights of the discriminant function to be learned, and $f(net)$ is a non-linear activation function, such as functions (1.6) or (1.8) already considered in Section 1.2.

The output of SLP can be used to perform classification of the vector $\boldsymbol{X}$. The decision boundary is linear. When the activation function (1.8) is used, one needs to compare the output o with zero threshold. To find the weights, a certain cost function is minimised. In the feedforward ANN design, the most popular cost function used is the sum of squares, defined in Section 1.2. Instead of the quadratic loss one can use the Kulback – Leibler distance (see e.g. Haykin, 1999). We concentrate here on the quadratic loss. In this chapter, we consider a more general form of the training-set cost with an additional regularisation term

$$cost_t = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{Ni} \left(t_j^{(i)} - f(\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0) \right)^2 + \lambda(\boldsymbol{V}^T\boldsymbol{V} - c^2)^2, \quad (4.2)$$

where: $t_j^{(i)}$ is a desired output (a target) for $\boldsymbol{X}_j^{(i)}$, the j-th training set observation from ω_i, λ and c^2 are parameters of the penalty (regularisation) term. An alternative to this term is the standard "weight decay" term $+\lambda\boldsymbol{V}^T\boldsymbol{V}$.

In artificial neural network training, the gradient iterative procedure is the most popular method for minimising the cost function and finding the weights. At iteration t, the weight vector is updated according to the equation

$$\boldsymbol{V}_{(t+1)} = \boldsymbol{V}_{(t)} - \eta \frac{\partial cost_t}{\partial \boldsymbol{V}}, \quad v_0(t+1) = v_0(t) - \eta \frac{\partial cost_t}{\partial v_0}, \quad (4.3)$$

where $\frac{\partial cost_t}{\partial \boldsymbol{V}}$ is a gradient of the cost function, and η is a learning-step.

4.1.2 The SLP as Statistical Classifier

4.1.2.1 The Euclidean Distance Classifier

Let us use the tanh activation function (1.8) and let the targets $t_j^{(1)} > 0$, $t_j^{(2)} < 0$. Temporarily, we relax the requirement $N_2 = N_1$. Denote

$$k_1 = \frac{N_1}{N_1 + N_2}, \; k_2 = \frac{N_2}{N_1 + N_2},$$

and let $\hat{\boldsymbol{M}} = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{Ni} \boldsymbol{X}_j^{(i)} = k_1 \hat{\boldsymbol{M}}_1 + k_2 \hat{\boldsymbol{M}}_2$ be the centre of the training vectors. Suppose that the following conditions are fulfilled:

E1) the centre $\hat{\boldsymbol{M}}$ is moved to the zero point;
E2) training begins from zero weights;
E3) the target $t_2 = - t_1 N_1/N_2$;
E4) the total gradient training (batch mode) is used.

The gradient of the cost function (4.2) without the regularisation term (λ=0) is

$$\frac{\partial cost_l}{\partial v_0}|_{v_0=v_{0(t)}} = 2(-(t_1 k_1 + t_2 k_2) + v_{0(t)} + (k_1 \hat{\boldsymbol{M}}_1 + k_2 \hat{\boldsymbol{M}}_2)^T \boldsymbol{V}_{(t)}), \tag{4.4}$$

$$\frac{\partial cost_l}{\partial \boldsymbol{V}}|_{\boldsymbol{V}=\boldsymbol{V}_{(t)}} = 2(-(t_1 k_1 \hat{\boldsymbol{M}}_1 + t_2 k_2 \hat{\boldsymbol{M}}_2) + (k_1 \hat{\boldsymbol{M}}_1 + k_2 \hat{\boldsymbol{M}}_2) v_{0(t)} + \mathbf{K}\boldsymbol{V}_{(t)}), \tag{4.5}$$

where $$\mathbf{K} = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} \boldsymbol{X}_j^{(i)} (\boldsymbol{X}_j^{(i)})^T. \tag{4.6}$$

It was assumed that $v_{0(0)} = 0$, $\boldsymbol{V}_{(0)} = \mathbf{0}$ (assumption E2). Therefore, after the first iteration, one has

$$v_{0(1)} = 2\eta(t_1 k_1 + t_2 k_2), \text{ and } \boldsymbol{V}_{(1)} = \boldsymbol{V}_{(0)} - 2\eta \left. \frac{\partial\, cost_t}{\partial \boldsymbol{V}} \right|_{\boldsymbol{V}=\boldsymbol{V}_{(0)}} =$$

$$2\eta(t_1 k_1 \hat{\boldsymbol{M}}_1 + t_2 k_2 \hat{\boldsymbol{M}}_2).$$

The targets satisfy condition $t_2 N_2 = -t_1 N_1$ (assumption E3). Therefore

$$\boldsymbol{V}_{(1)} = 2\eta\, t_1 k_1 \Delta \hat{\boldsymbol{M}}, \; v_{0(1)} = 0, \tag{4.7}$$

where $\Delta \hat{\boldsymbol{M}} = \hat{\boldsymbol{M}}_1 - \hat{\boldsymbol{M}}_2$.

Vector (4.7) is proportional to the weight vector of the Euclidean Distance Classifier designed for centred data, i.e., $\hat{\boldsymbol{M}} = \mathbf{0}$ (assumption E1). The classification according to the numerical sign of the linear discriminant function with weights (4.7) is asymptotically (as $N \to \infty$) optimal when the classes are spherical Gaussian $N_X(X, \boldsymbol{M}_i, \mathbf{I}\sigma^2)$, but also in many other situations. The single-layer perceptron has the useful property of being able to become a comparatively good statistical classifier after just the first training iteration. To achieve this, one has to satisfy conditions E1–E4 enumerated above.

If the number of training vectors from the pattern classes are different ($N_2 \neq N_1$), in order to obtain EDC after the first iteration we have to use *asymmetrical targets*, i.e. for the tanh activation function $t_2 = -t_1 N_1/N_2$.

4.1.2.2 The Regularised Discriminant Analysis

Now we are able to analyse the dynamics of the weight vector after the second and following iterations. Let us use the activation function (1.8), targets $t_j^{(1)} = 1$, $t_j^{(2)} = -1$. Let $N_2 = N_1 = \overline{N} > n/2 + 1$, i.e. the sample covariance matrix $\hat{\Sigma}$ is not singular. Assume that the learning step η is small and after t iterations the weights remain small. Then the weighted sums $|\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0|$ are also small. In such a case, the non-linear activation function (1.8) acts as a linear function, i. e.

$$f(\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0) \approx \boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0.$$

After the second iteration, utilisation of the total gradient adaptation rule (4.3) with the gradients (4.4) and (4.5) results in:

$$v_{0(2)} = 0, \qquad \boldsymbol{V}_{(2)} = \boldsymbol{V}_{(1)} - \eta \frac{\partial\, cost_t}{\partial \boldsymbol{V}}\bigg|_{\boldsymbol{V}=\boldsymbol{V}_{(1)}} =$$

$$\eta\, \Delta \hat{\boldsymbol{M}} - \eta(-\Delta \hat{\boldsymbol{M}} + 2\mathbf{K}\, \boldsymbol{V}_{(1)}) = (\mathbf{I} - (\mathbf{I} - \eta\, \mathbf{K})^2)\, \mathbf{K}^{-1}\, \Delta \hat{\boldsymbol{M}}\,.$$

In further iterations,

$$v_{0(t)} = 0, \qquad \boldsymbol{V}_{(t)} = (\mathbf{I} - (\mathbf{I} - \eta\, \mathbf{K})^t)\, \mathbf{K}^{-1}\, \Delta \hat{\boldsymbol{M}}\,, \tag{4.8}$$

where $\mathbf{K} = \frac{1}{2\overline{N}} \sum_{i=1}^{2} \sum_{j=1}^{\overline{N}} \boldsymbol{X}_j^{(i)} ((\boldsymbol{X}_j^{(i)})^T = \frac{\overline{N}-1}{\overline{N}}\, \hat{\Sigma} + \frac{1}{4}\, \Delta \hat{\boldsymbol{M}}\, \Delta \hat{\boldsymbol{M}}^T$.

To derive (4.8), the matrix $\mathbf{K}$ is assumed to be non-singular. Employing the first terms of expansions

$$(\mathbf{I} - \eta\, \mathbf{K})^t = \mathbf{I} - t\, \eta\, \mathbf{K} + \tfrac{1}{2}\, t(t-1)\eta^2\, \mathbf{K}^2 - \ldots \quad \text{and} \quad (\mathbf{I} - \beta\, \mathbf{K})^{-1} = \mathbf{I} + \beta\, \mathbf{K} + \ldots$$

for small η and t, one can obtain

$$\boldsymbol{V}_{(t)} \approx (t\, \eta \mathbf{K} - \tfrac{1}{2} t(t-1)\eta^2 \mathbf{K}^2))\, \mathbf{K}^{-1} \Delta \hat{\boldsymbol{M}} = t\, \eta(\mathbf{I} - \tfrac{1}{2}(t-1)\, \eta\, \mathbf{K})\, \Delta \hat{\boldsymbol{M}} \approx$$

$$t\, \eta(\mathbf{I} + \tfrac{1}{2}(t-1)\, \eta\, \mathbf{K})^{-1} \Delta \hat{\boldsymbol{M}} = \frac{2t}{(t-1)} \left(\mathbf{I} \frac{2}{(t-1)\eta} + \mathbf{K}\right)^{-1} \Delta \hat{\boldsymbol{M}} =$$

$$\frac{2t}{(t-1)} \left(\mathbf{I}\, \frac{2}{(t-1)\eta} + \left(\frac{\overline{N}-1}{\overline{N}}\, \hat{\Sigma} + \frac{1}{4}\, \Delta \hat{\boldsymbol{M}}\, \Delta \hat{\boldsymbol{M}}^T\right)^{-1} \right) \Delta \hat{\boldsymbol{M}}\,.$$

Sample CM $\hat{\boldsymbol{\Sigma}}$ is assumed to be not singular. Then after some matrix algebra the following is obtained

$$\boldsymbol{V}_{(t)} = \left(\mathrm{I}\frac{2}{(t-1)\eta}\ \frac{\overline{N}}{\overline{N}-1} + \hat{\boldsymbol{\Sigma}}\ \right)^{-1} \Delta \hat{\boldsymbol{M}}\ k_{\mathrm{R}}, \quad v_{0(t)} = 0, \tag{4.9}$$

where k_{R} is a scalar coefficient.

When classifying according to the sign of the discriminant function, the coefficient k_{R} is irrelevant. Therefore, the weight vector $\boldsymbol{V}_{(t)}$ is equivalent to that of the regularised discriminant analysis (2.38*a*) with the regularisation parameter

$$\lambda = \frac{2}{(t-1)\eta}\ \frac{\overline{N}}{\overline{N}-1}.$$

4.1.2.3 The Standard Linear Fisher Classifier

The above expressions indicate that when the number of the training iterations increases, $\left(\mathrm{I}\frac{2}{(t-1)\eta}\ \frac{\overline{N}}{\overline{N}-1} + \hat{\boldsymbol{\Sigma}}\ \right) \rightarrow \hat{\boldsymbol{\Sigma}}$ and the resulting classifier approaches *the standard Fisher linear DF*. The Fisher classifier can be obtained also when we utilise the linear perceptron (here, $f(net) = net$) and equate the derivatives (4.4) and (4.5) to zero and solve the resulting equations with respect to v_0 and $\boldsymbol{V}$. Then for the centred data one has $\hat{v}_0 = 0$, and $\hat{\boldsymbol{V}} = \mathbf{K}^{-1}(\hat{\boldsymbol{M}}_1 - \hat{\boldsymbol{M}}_2)$ (see Section 1.3).

4.1.2.4 The Pseudo-Fisher Classifier

While deriving the weight vector (4.9), it was assumed that the sample covariance matrix is not singular. If the number of training samples $N < n + 2$, the matrix $\hat{\boldsymbol{\Sigma}}$ is singular. Now one can show that with an increase in the number of iterations, the resulting classifier approaches the Fisher linear DF with the pseudo-inversion of the sample covariance matrix. We assume the weights are small, the training data is centred, $t_1 = - t_2 = 1$, and $N_2 = N_1 = \overline{N}$. The weight vector (4.8) can be written in a way that does not require the matrix $\mathbf{K}$ to be non-singular

$$\boldsymbol{V}_{(t)} = \sum_{s=1}^{t} C_t^s\ (-\eta)^s\ \mathbf{K}^{s-1} \Delta \hat{\boldsymbol{M}}\ .$$

Let the orthogonal $n \times n$ matrix $\boldsymbol{\Phi}$ satisfy representation $\boldsymbol{\Phi}^T \mathbf{K}\, \boldsymbol{\Phi} = \begin{bmatrix} \lambda & 0 \\ 0 & 0 \end{bmatrix}$. Then

$$\boldsymbol{V}_{(t)} = \sum_{s=1}^{t} C_t^s (-\eta)^s \boldsymbol{\Phi} (\boldsymbol{\Phi}^T \mathbf{K}\, \boldsymbol{\Phi})^{s-1} \boldsymbol{\Phi}^T \Delta \hat{\boldsymbol{M}} = \sum_{s=1}^{t} C_t^s (-\eta)^s \boldsymbol{\Phi} \begin{bmatrix} \lambda & 0 \\ 0 & 0 \end{bmatrix}^{s-1} \boldsymbol{\Phi}^T \Delta \hat{\boldsymbol{M}} =$$

$$\boldsymbol{\Phi} \left(\sum_{s=1}^{t} C_t^s (-\eta)^s \begin{bmatrix} \lambda & 0 \\ 0 & 0 \end{bmatrix}^s \right) \begin{bmatrix} \lambda^{-1} & 0 \\ 0 & 0 \end{bmatrix} \boldsymbol{\Phi}^T \Delta \hat{\boldsymbol{M}} =$$

$$\mathbf{\Phi}\left(\mathbf{I}-\left(\mathbf{I}-\eta\begin{bmatrix}\lambda & 0\\ 0 & 0\end{bmatrix}\right)^{t}\right)\mathbf{\Phi}^{T}\mathbf{\Phi}\begin{bmatrix}\lambda^{-1} & 0\\ 0 & 0\end{bmatrix}\mathbf{\Phi}^{T}\Delta\hat{\boldsymbol{M}} =$$

$$\mathbf{\Phi}\left(\mathbf{I}-\left(\mathbf{I}-\eta\begin{bmatrix}\lambda & 0\\ 0 & 0\end{bmatrix}\right)^{t}\right)\mathbf{\Phi}^{T}\boldsymbol{V}^{\mathrm{PF}}, \qquad (4.10)$$

where $\boldsymbol{V}^{\mathrm{PF}}=\mathbf{\Phi}\begin{bmatrix}\lambda^{-1} & 0\\ 0 & 0\end{bmatrix}\mathbf{\Phi}^{T}\Delta\hat{\boldsymbol{M}} = \mathbf{K}^{+}\Delta\hat{\boldsymbol{M}}$, and the pseudo-inverse $\mathbf{K}^{+}$ is defined by Equation (2.35).

For small values of η with an increase in t

$$\left(\mathbf{I}-\left(\mathbf{I}-\eta\begin{bmatrix}\lambda & 0\\ 0 & 0\end{bmatrix}\right)^{t}\right)\rightarrow\mathbf{I}.$$

Then $\boldsymbol{V}_{(t)}\rightarrow\boldsymbol{V}^{\mathrm{Fpseudo}}$ (the weight vector of the *Fisher linear DF with pseudo-inversion*). This conclusion is correct while the inputs $net = \boldsymbol{V}^{T}\boldsymbol{X} + v_0$ to the activation function *o(net)* vary in the linear part of the function *o(net)*.

4.1.2.5 Dynamics of the Magnitudes of the Weights

In the ANN literature, it is recommended that one starts training the perceptron with small initial weights. Above we have presented the arguments that in the SLP perceptron training, it is beneficial to begin from zero-valued initial weights. Initially, in both cases, the weights of the perceptron are small. While training, the magnitudes of the weights increase. We can illustrate this phenomenon theoretically.

First, we rewrite the cost function (4.2) without the regularisation term in following way

$$cost_t = \frac{1}{N_1 + N_2}\sum_{i=1}^{2}\sum_{j=1}^{N_i}\left(t_j^{(i)} - f(G_{ij})\right)^2,$$

where $G_{ij} = c(\boldsymbol{V}^{T}\boldsymbol{X}_j^{(i)} + v_0)$, the weights v_0, $\boldsymbol{V}$ are determined by the weights of the Fisher DF (Equation (1.3)), and a constant c controls a magnitude of the weights.

In the high-dimensional case, we can assume a set of the values G_{11}, G_{12}, …, G_{2N} have the Gaussian distribution. To find an optimal value of c we note that the mean values E_1, E_2 and a common standard deviation, sd, of G_{ij} for both classes are:

$$E_i = (-1)^{i+1}\,½\,c\,\hat{\delta}^2, \text{ and } sd = c\,\hat{\delta},$$

where $\hat{\delta}^2 = (\hat{\boldsymbol{M}}_1 - \hat{\boldsymbol{M}}_2)^T \hat{\bar{\Sigma}}^{-1} (\hat{\boldsymbol{M}}_1 - \hat{\boldsymbol{M}}_2)$ is the sample squared Mahalanobis distance.

The search for a minimum of $cost_t$ with linear and non-linear activation functions for limiting target values results in the optimal value: $c_{opt} = 1$. Therefore, the mean values and standard deviations of the weighted sums $\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0$ are

$$\begin{aligned} &\text{for } \hat{\delta} = 1, \quad |E_i| = 0.5, \quad sd = 1, \\ &\text{for } \hat{\delta} = 2, \quad |E_i| = 2, \quad sd = 2, \\ &\text{for } \hat{\delta} = 8, \quad |E_i| = 32, \quad sd = 8. \end{aligned}$$

The above considerations show that in the training process, the weights can increase up to certain values. These values depend on the separability of the training sets of opposite pattern classes, δ: for close classes, we should obtain small weights and, for distant classes, large ones. If $\hat{\delta}$ is infinitely large, then the magnitude of the weights is unbounded.

At the beginning of training, the weights are small, and all weighted sums G_{ij} are small too. Later, the weights and the weighted sums G_{ij} are increasing. At the very end of the training process, we approach a minimum of the cost function. Then for very distant pattern classes (large $\hat{\delta}$) the outputs of SLP for almost all training vectors $\boldsymbol{X}_j^{(i)}$ should become very close to the boundary values of the activation function.

4.1.2.6 The Robust Discriminant Analysis

With an increase in the number of iterations, the magnitudes of the weights should grow. Consequently, a major part of the weighted sums $|\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0|$ becomes large. The outputs of the tanh activation function are bounded by -1 and 1, i.e. $-1 \le f(\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0) \le 1$. As a result, for large weighted sums $|\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0|$ the activation function begins to act as the non-linear function. For large weights, the non-linearity of the activation function reduces the influence of distant atypical training vectors (outliers) and we obtain a robust linear classification rule.

Note, in the finite iteration case, we have a *robust and regularised discriminant function.* This is a new way to design the classification rule, which has no analogue in statistical pattern recognition. The robust discriminant function can be obtained also for the case where the target values differ from the limit ones (e.g. $t_1 = 0.8$ and $t_2 = -0.8$ for tanh activation function (1.8)).

4.1.2.7 The Minimum Empirical Error Classifier

When the targets values are +1 and -1 and the empirical classification error is small the weights may become very large. Then outputs of SLP, $o = f(\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_0)$ become very close to either -1 or +1. Consequently, while minimising the cost

(4.2), roughly, we are minimising the empirical classification error as well (theoretically, we minimise the empirical classification error only in situations where the weights and the weighted sums $|\boldsymbol{V}^T\boldsymbol{X}_j^{(i)} + v_0|$ are extraordinarily large).

In situations where the number of empirical errors is high, the weights cannot become too large. In order to minimise the number of training-set errors one ought to increase the weights purposefully. To do this one may fix certain values of the parameters λ and c^2 in the cost function (4.2) and obtain an "anti-regularisation" effect (Section 4.6.3).

4.1.2.8 The Maximum Margin (Support Vector) Classifier

When the number of training vectors, N, is small in comparison with the number of inputs n then after a certain amount of training, the number of empirical (training-set) errors can become equal to zero. In such a situation, we obtain the zero empirical error classifier. Suppose we incorporate certain requirements in the cost function in order to prevent an excessive growth of the weights' magnitudes (e.g., we add a term $\lambda(\boldsymbol{V}^T\boldsymbol{V} - c^2)^2$ with sufficiently large c^2 to the cost function). Then in order to diminish the cost, the training algorithm should maximise distances from the training vectors to a separating hyperplane, $\boldsymbol{X}^T\boldsymbol{V}_{(t)} + v_{0(t)} = 0$.

Training vectors which are closest to the hyperplane, $\boldsymbol{X}^T\boldsymbol{V}_{(t)} + v_{0(t)} = 0$, have maximal contribution to the cost (4.2). Due to the exponential character of the activation function (1.8), their contribution can be orders of magnitude larger than contributions of more distant vectors. Therefore, the training algorithm reacts to the closest training vectors. Thus, it maximises the margin between the hyperplane $\boldsymbol{X}^T\boldsymbol{V}_{(t)} + v_{0(t)} = 0$ and the closest training vectors. This is the maximal margin classifier. In cases when, instead of working in the original feature space, we map the input vectors $\boldsymbol{X}$ into a high-dimensional feature space Ω_{new} through some non-linear mapping, chosen *a priori*, we have the SV classifier.

4.1.3 Training Dynamics and Generalisation

It has been shown that while satisfying conditions E1 – E4, one can obtain seven statistical classifiers of differing complexity. These are beneficial characteristics of the single layer perceptrons. In Chapter 5, we will utilise these properties to integrate the statistical and neural approaches in designing linear classifiers. In this subsection, we illustrate these properties by describing several characteristic simulation experiments with bi-variate data sets. In these experiments, we demonstrate the influences of the data, and the parameters of the training algorithm on the type of classifier obtained.

Example 1. In Figure 4.1, we have the distribution of two bi-variate Gaussian classes **A** (already depicted in Figure 1.3): two GCCM "signal" classes contaminated with 10% additional Gaussian noise. The noise patterns are indicated by "$*$" and "+", the signal classes by two small ellipses.

In all training experiments reported in this section, we use the centred training data and the starting weight vector $v_{(0)} = 0, \boldsymbol{V}_{(0)} = \mathbf{0}$. We train the single layer perceptron with the sigmoid activation function and targets t_1=1 and t_2=0. We train for t_{max} = 1000 iterations by standard back-propagation in batch mode, with the learning step η = 5. The training-set size is $N_1 = N_2 = \bar{N}$ = 250. After the first iteration (boundary 1 in Figure 4.1), we obtain the EDC (boundary 2) yielding an error rate of 22%. After 250 iterations, the resulting classifier (boundary 3) becomes very close to the Fisher linear discriminant (boundary 4) and yields an error rate of 12%.

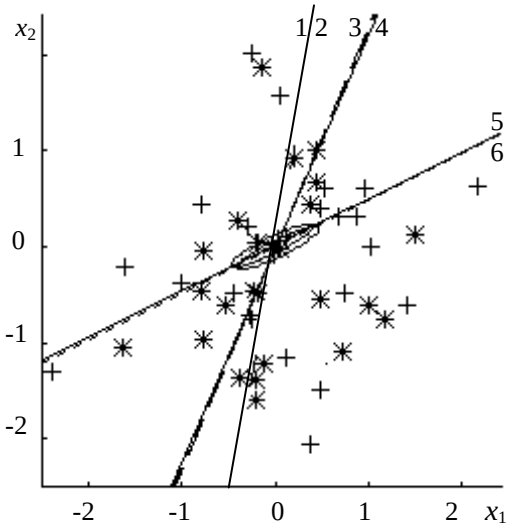

Fig. 4.1. Dynamics of the SLP training process: the distribution of two Gaussian pattern classes contaminated by noise and the positions of the discriminant lines: 1 – SLP after the first iteration, 2 – Euclidean distance classifier, 3 – SLP after 250 iterations. 4 – the Fisher linear DF, 5 – the MEE classifier, 6 – SLP at the end of the training process (Reprinted from *Neural Networks*, 11:283-296, Raudys, Evolution and generalization of a single neurone, 1998, with permission from Elsevier Science).

After 750 more iterations with the learning step η=5, the decision boundary of SLP actually comes extremely close to the boundary of the Fisher linear discriminant (boundary 4). A significant increase in the learning step (up to η=100) helps to move the decision boundary of SLP (boundary 5, after 1000 iterations) to boundary 6 of the minimum empirical error classifier (error rate of 5.5%).

Example 2. More details on designing the maximal margin classifier can be perceived from Figure 4.2. In this bi-variate data model, each class consists of a mixture of two Gaussian densities. Each subclass is distributed on its own straight "data line" in the bi-variate space. All four data lines are parallel. Conventional training of the SLP allows us to obtain the zero empirical error classifier, however, it does not allow us to maximise the margin between the training vectors and the

discriminant boundary $v_0 + v_1x_1 + v_2x_2 = 0$. The reason is simple: after a certain amount of training, the weights become rather large and diminish the gradient of the cost function. Thus, the training process essentially stops.

In order to obtain the maximal margin, one needs to have large weights. One way to do this is to increase the learning step. Constant values of the learning step guarantee the learning process only at the very beginning of iteration, while the weights are small, and the activation function $o(g)$ is unsaturated.

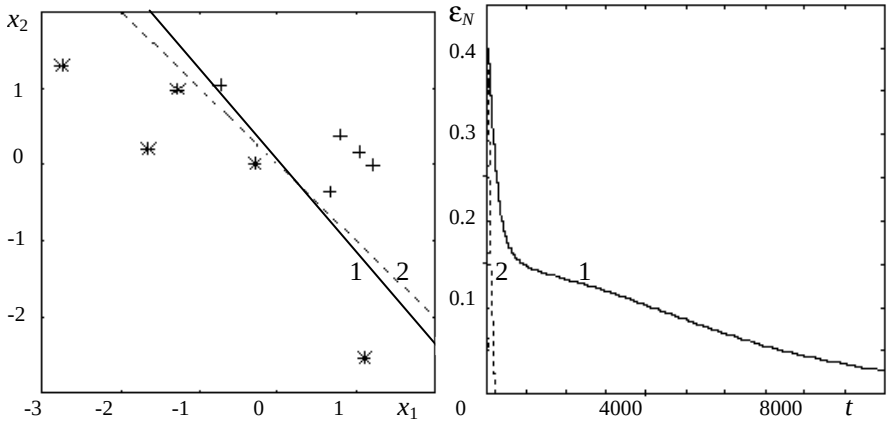

Fig. 4.2. SLP as the maximum margin classifier. *a* shows the training data, a mixture of Gaussian subclasses: 1 – discriminant line after training with $\eta = 0.5$, 2 – discriminant line after training with varying η ($\eta = 0.5 \times 1.05^t$). Only four vectors contribute to the determination of the exact position of the boundary. *b* shows the generalisation error versus *t*, the number of iterations: 1 – $\eta = 0.5$, 2 – $\eta = 0.5 \times 1.05^t$ (Reprinted from *Neural Networks*, 11:283-296, Raudys, Evolution and generalization of a single neurone, 1998, with permission from Elsevier Science).

After training with the exponentially increasing learning step ($\eta = 0.5 \times 1.05^t$), the decision boundary is placed approximately halfway between the closest vectors of opposite pattern classes. This is the maximum margin classifier. The remaining vectors from the other two subclasses do not affect the position of the decision boundary.

Example 3. Details concerning the exponentially increasing learning step technique are presented in Figure 4.3. We present the change in the magnitude of the weights of SLP during 800 iterations for 100-dimensional Gaussian N($\boldsymbol{M}_i$, $\boldsymbol{\Sigma}$) data. The parameters of the data model are $\boldsymbol{M}_2 = -\boldsymbol{M}_1 = \boldsymbol{M} = (m_1, m_2, \ldots, m_{100})^T$. Randomly χ_1^2 distributed components m^2_i were normalised ($\boldsymbol{M}^T\boldsymbol{M} = 4$). The variances of all features were set to unity, $\sigma_{ii} = 1$, and the covariances were set to $\sigma_{ij} = 0.3$. The training set size is set at $\overline{N} = 100$.

In this high-dimensional example, we have two well-separated pattern-classes and a comparatively small number of training vectors. Therefore, the training data

is linearly separable. After achieving the zero empirical error, we obtain a certain margin between the decision boundary and the training vectors. In this experiment, we compare two strategies to control the learning step η. In the first three experiments, the parameter η is kept constant, in the fourth experiment, the parameter η is exponentially increasing with the iteration number t, $\eta = \eta_o \times \gamma^t$. In practice, we need fix a bound for a maximal value η_{max}. We see the value of the learning step η and its behaviour with an increase in the number of iterations affects the magnitude of the weights and indirectly influences the outcome of the training process. Thus, the learning step is one of main factors which influence the statistical properties of the classification rule and its generalisation error.

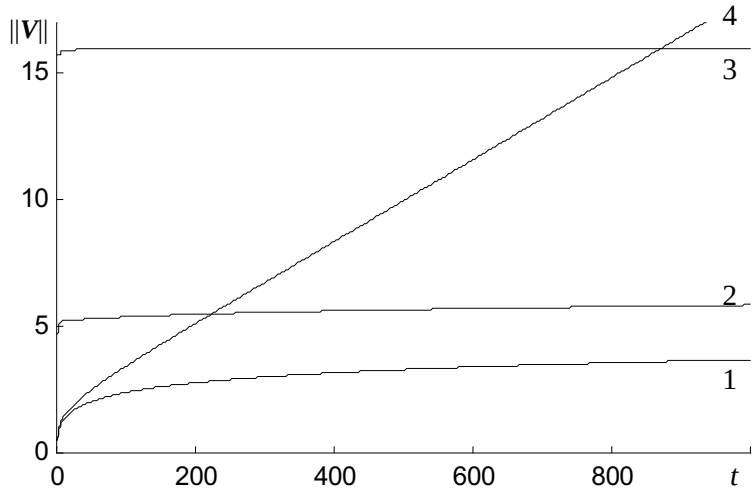

Fig. 4.3. An increase in the magnitude $||\boldsymbol{V}||$ of the weights versus t, the number of iterations: 1 – $\eta = 0.01$, 2 – $\eta = 0.1$, 2 – $\eta = 0.3333$, 4 – $\eta = 0.01 \times 1.05^t$.

4.2 Non-Linear Decision Boundaries

In the previous section, we demonstrated that in its dynamic evolution, SLP can realise seven linear classification algorithms of different complexity. In order to obtain non-linear boundaries, we have either to utilise more complicated algorithms or prior to training, perform non-linear feature transformations.

4.2.1 The SLP in Transformed Feature Space

A standard approach to obtaining non-linear decision boundaries by means of simple algorithms is to map the input vectors $\boldsymbol{X}$ into a high-dimensional feature space through some non-linear mapping, chosen *a priori*

$$y_\beta = f(\boldsymbol{X}, \mathbf{Y}_\beta), \qquad \beta = 1, 2, \ldots, m,$$

where m is the dimensionality of the new space, and $\mathbf{Y}_1, \mathbf{Y}_1, \ldots, \mathbf{Y}_m$ is a set of known vector parameters.

In this *m*-variate space, one constructs a linear classifier which, in the original low-dimensional space, forms the non-linear decision boundary. Two classical examples are the polynomial classifiers of Schuermann (1977) and the support vector (SV) machines of Vapnik (1995). Below, an example using the SLP is presented.

Example 4. In Figure 4.4 we have a distribution of two unimodal bi-variate populations (100 points from each pattern class) with slightly asymmetrical distribution densities of the features. To discriminate the pattern classes we used the standard quadratic DF at first (decision boundary Q).

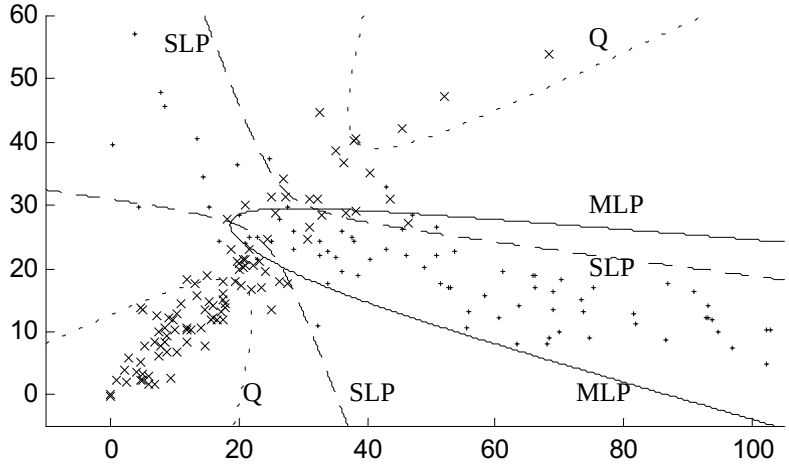

Fig. 4.4. The non-linear decision boundaries from the standard quadratic discriminant function Q (six weights, 19.5% of errors), the multilayer perceptron MLP with two hidden units (nine weights, 11.5% of errors), and the SLP in the transformed five-variate polynomial feature space (six weights, 9.5% of errors).

In order to obtain the better decision boundary, we used MLP (decision boundary MLP). In addition, we mapped the data into five-variate polynomial feature space: $y_1 = x_1$, $y_2 = x_2$, $y_3 = x_1^2$, $y_4 = x_2^2$, $y_5 = x_1 x_2$ and trained the SLP in order to obtain the minimal number of errors while classifying 200 training vectors. We obtained the same type of quadratic decision boundary as in the first case. The weights, however, were different and resulted in a different decision boundary with half the number of classification errors.

In the high-dimensional feature space, the new features were highly correlated. Therefore, to assure convergence, we had to whiten the data (see Section 5.5.1 for details) and to use the antiregularisation technique (see Section 4.6.3 for details).

The polynomial feature transformation is the first and most popular choice for forming "the high-dimensional feature space through some non-linear mapping, chosen *a priori*". Another popular class of functions is radial basis functions:

$$y_\beta = \kappa(\boldsymbol{X}, \boldsymbol{C}_\beta). \tag{4.11}$$

As a simple example, consider the spherical Gaussian density

$$\kappa(\boldsymbol{X}, \boldsymbol{C}_\beta) = N_X(X, \boldsymbol{M}_i, \mathbf{I}\sigma_\beta^2) = \prod_{j=1}^{n} (2\pi)^{-1/2} \sigma_\beta^{-1} e^{-1/2 (x_j - c_{\beta j})^2 / \sigma_\beta^2}, \quad (4.12)$$

where σ_β is a radius of β^{th} radial function and $\boldsymbol{C}_\beta = (c_{\beta 1}, c_{\beta 1}, \ldots, c_{\beta n})^T$ is its centre vector.

In principle, more complicated radial basis functions can be chosen. One more class of mapping functions is non-linear transformations of the weighted sums

$$y_\beta = f(\boldsymbol{X}^T \boldsymbol{V}_\beta + v_{0\beta}), \quad (4.13)$$

where $f(net)$ is a non-linear function, e.g. $f(net) = \tanh(net)$, $v_{0\beta}$, and $\boldsymbol{V}_\beta^T = (v_{\beta 1}, v_{\beta 2}, \ldots, v_{\beta n})$ are coefficients of the β^{th} mapping function.

In this approach, to design the non-linear classifiers, the parameters σ_β (the radius), $c_{\beta 1}, c_{\beta 1}, \ldots, c_{\beta n}$ (coordinates of the centre vector), and $v_{0\beta}, v_{\beta 1}, v_{\beta 2}, \ldots, v_{\beta n}$ (the coefficients of the mapping function) are chosen *a priori*. In training the classifier, one needs to find weights $w_0, w_1, \ldots, w_m$, of the final linear classifier

$$output = \sum_{\beta=1}^{m} w_\beta y_\beta + w_0. \quad (4.14)$$

Instead of using the SLP to find the weights in the new high-dimensional feature space, one can utilise any of a number of known linear classifiers. Schuermann utilised the standard Fisher classifier with the pseudo-inversion of the covariance matrix. Vapnik used the linear maximal margin classifier. The SLP is advantageous in the sense that, while training the perceptron in an adaptive way, we can obtain both linear classifiers just mentioned and five others (Section 4.1). In the new high-dimensional feature space, we face a serious additional difficulty: the new features are highly-correlated and the SLP classifier's training can become very slow. In chapter 5 we will discuss peculiarities of training SLP in the high-dimensional highly-correlated feature space.

4.2.2 The MLP Classifier

The MLP classifier is characterised by a large number of hidden layers and output layer weights. Theory shows that the standard MLP with one hidden layer and smooth non-linear activation functions can realise an arbitrary non-linear decision boundary. For this we need to have a sufficient number of hidden units. The information processing scheme of the MLP classifier with one hidden layer is similar to (4.14) with the non-linear transformations of the weighted sums (4.13). A main difference is that in MLP the coefficients of the mapping functions ($v_{0\beta}, v_{\beta 1}, v_{\beta 2}, \ldots, v_{\beta n}$, $\beta = 1, 2, \ldots, h$) should be found from the training data. Previously, in this book, we have presented three examples of non-linear decision boundaries obtained by utilising MLP (Figures 1.5, 2.1 and 4.4).

A standard method of finding the weights is to use the gradient descent optimisation (a back propagation method) where we calculate derivatives of the cost function and find the weights in an iterative way. To perform the optimisation task we need to fix starting values for the weight vector. This stage of the training process is called "the weights initialisation". In Section 4.5.2 we will show that the weights initialisation is very important in determining the performance of the neural network. In this book, we focus on the complexity and generalisation problems. Therefore, for standard details concerning back propagation and other training procedures the reader is referred to the excellent textbooks of Hertz, Krogh and Palmer (1991), Bishop (1995) and Haykin (1999).

4.2.3 Radial Basis Function Networks

The information processing scheme of the Radial Basis Functions (RBF) network is similar to (4.14) with the non-linear transformations of the weighted sums (4.11) and (4.12). A main difference consists in that, in RBF, the coefficients of the mapping functions should be found from the training data. The discriminant function used to make the decision is similar to that discussed for the statistical classifier (2.44) derived for the mixtures of Gaussian or non-Gaussian spherical densities (Section 2.6.1). In the statistical approach, in order to find the unknown coefficients of the decision rule we use statistical parameter estimation methods. Typically, these parameters are different for each pattern class. In the neural network approach, we have to choose the cost function and minimise it. Here, in determining the parameters of the network, training vectors of all pattern classes are used simultaneously. This is one of the advantages of the RBF networks over the statistical classifiers. Equation (4.15) is an example of the cost function for the two category case

$$cost_t = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} \psi\Big(t_j^{(i)} - \sum_{\beta=1}^{m} v_{i\beta}\, \kappa(\boldsymbol{X}_j^{(i)}, \boldsymbol{C}_{i\beta}, \sigma_{i\beta})\Big). \qquad (4.15)$$

where $\psi(c)$ is a loss function (e.g. quadratic, robust, etc.).

In the RBF approach, an important problem is to choose the targets, $t_j^{(i)}$. One possibility is to use zeros and ones that indicate the class membership of each training vector $\boldsymbol{X}_j^{(i)}$. As the starting values of the unknown coefficients $v_{i\beta}$, $\sigma_{i\beta}$ and $c_{i\beta 1}, c_{i\beta 2}, \ldots, c_{i\beta n}$ the standard statistical mixture decomposition or cluster analysis method can be used (for cluster analysis, see Fukunaga, 1990, Duda *et al*., 2000).

In a modification of RBF, as the output of the network one can use the ratio

$$P_s(\boldsymbol{X}) = \frac{\sum_{\beta=1}^{m} v_{s\beta}\, \kappa(\boldsymbol{X}, \boldsymbol{C}_{s\beta}, \sigma_{s\beta})}{\sum_{i=1}^{2} \sum_{\beta=1}^{m} v_{i\beta}\, \kappa(\boldsymbol{X}, \boldsymbol{C}_{i\beta}, \sigma_{i\beta})}. \qquad (4.16)$$

If one supposes that expression $\sum_{\beta=1}^{m} v_{s\beta}\ \kappa(\boldsymbol{X},\ \boldsymbol{C}_{s\beta},\ \sigma_{s\beta}))$ approximates the class conditional density $f_s(\boldsymbol{X})$, then following the statistical pattern recognition theory, one can conclude that expression (4.16) will represent the *a posteriori* probability of *s*-th class. Use of (4.16) in the cost function (4.15) results in a differentiable function with an easy-to-find gradient. This approach can be used for formally introducing a rejection option into the algorithm. Some authors call such modifications of the RBF networks *probabilistic neural networks*.

4.2.4 Learning Vector Quantisation Networks

A simplification of the numerical calculation process in the RBF networks during the recognition phase follows if, instead of the sum $\sum_{\beta=1}^{m} v_{s\beta}\kappa(\boldsymbol{X},\boldsymbol{C}_{s\beta},\ \sigma_{s\beta}))$, one analyses each of the *m* components separately as an independent new subclass. Then a piecewise-linear discriminant function similar to (2.46) can be constructed. Such types of decision-making is called the learning vector quantisation network. This scheme is very similar to the *piecewise-linear classifiers* discussed in Section 2.6.2. A decision-making procedure which performs these calculations has been depicted in Figure 2.4. To estimate the unknown centres, $\boldsymbol{C}_{s\beta}$, and radiuses, $\sigma_{s\beta}$, one has to minimise a chosen cost function. As the starting values of the unknown coefficients, one can utilise the coefficients of the piecewise-linear discriminant function discussed in Section 2.6.2. Various approaches, including old ones traditionally used to design the statistical pattern classifiers, can be used to find the centres $\boldsymbol{C}_{s\beta}$. These methods include the cluster analysis methods, mixture decomposition methods and the great variety of decision tree design methods (Section 2.8.3). In the neural net approach, however, to find the coefficients, training vectors of all pattern classes take part simultaneously.

4.3 Training Peculiarities of the Perceptrons

The essential property of the cost functions utilised in ANN training is the non-linearity of the activation function. The non-linearity helps to obtain non-linear smooth decision boundaries. However, at the same time it burdens the iterative training process. One more training difficulty arises in cases where the training data are almost singular, i.e. we have an immense difference between the smallest and largest eigenvalues of the data covariance matrix.

4.3.1 Cost Function Surfaces of the SLP Classifier

The non-linear SLP classifier realises a hyperplane in the *n*-dimensional feature space, Ω. The position of this hyperplane is determined by *n* independent parameters. The SLP classifier, however, has *n*+1 weights that are determined in

the training process. Consequently, one weight is excessive. Therefore, in a multivariate weight space, the cost function has a deep and long trench.

Example 5. In Figure 4.5*a*, we present two sections of the sum of squares cost function surface. We consider SLP which determines a discriminant function $g(x_1) = v_1x_1+v_0 = w_{m1}(x_1+ w_{m0})$ utilised to classify uni-variate observations into two populations. The graphs are presented as the $cost_t$ versus w_{m0} for two values of a coefficient w_{m1} (a solid graph for $w_{m1} = 100$, here we have almost the hard-limiting activation function, and a dotted graph for $w_{m1} = 20$, here we have the smoother soft-limiting activation function).

Apparently, for the hard-limiting activation function ($w_{m1} = 100$) each training observation contributes a separate step into the cost function surface. We have stated earlier that the relative slope of the cost function depends on the magnitude of the weights. Therefore, near the origin ($w_{m1} = 20$, the small weights) the activation function acts almost as a linear one and the cost function is smooth. With an increase in the magnitude of the weights, the activation function acts almost as a hard-limiting function. Then the steps exhibit themselves more clearly. It becomes quite obvious from 2D and 3D plots of the cost function (Figure 4.5*bc*).

In this figure, we see four clear minima – four ditches in the landscape of the cost function. In spite of simplicity of the classification problem, the effect of multi-extremality of the cost function is obvious: the cost function is composed of numerous ditches and long hills spread along, like the rays from the origin of the co-ordinate axis.

4.3.2 Cost Function Surfaces of the MLP Classifier

In comparison with the SLP, the multilayer perceptrons with n inputs and h hidden neurones have more weights to be adjusted. Therefore, the analysis and understanding of the MLP cost function surfaces in an $(n+1)h+(h+1)$-variate weight space Θ_{MLP} are much more complicated. If the empirical error is zero, then, as a rule, the global minimum of the cost function surface is expressed more clearly.

Example 6. We consider the classification problem with two bi-variate pattern classes, where the vectors of the first pattern class are inside another one: vectors of bi-variate spherical Gaussian distribution N(0, **I**) are inside a circle $(x_1^2 + x_2^2 = 1.1)$ and vectors of the second class are outside a larger circle $(x_1^2 + x_2^2 = 1.45)$. It is data **SF2**. The number of training vectors is $N_1 = N_2 = 50$. The weights were initialised randomly in an interval (-3, 3) and MLP with three sigmoid neurones in the hidden layer was trained 5000 cycles by the stochastic back propagation method with targets 0.9 and 0.1 and learning step $\eta = 0.2$.

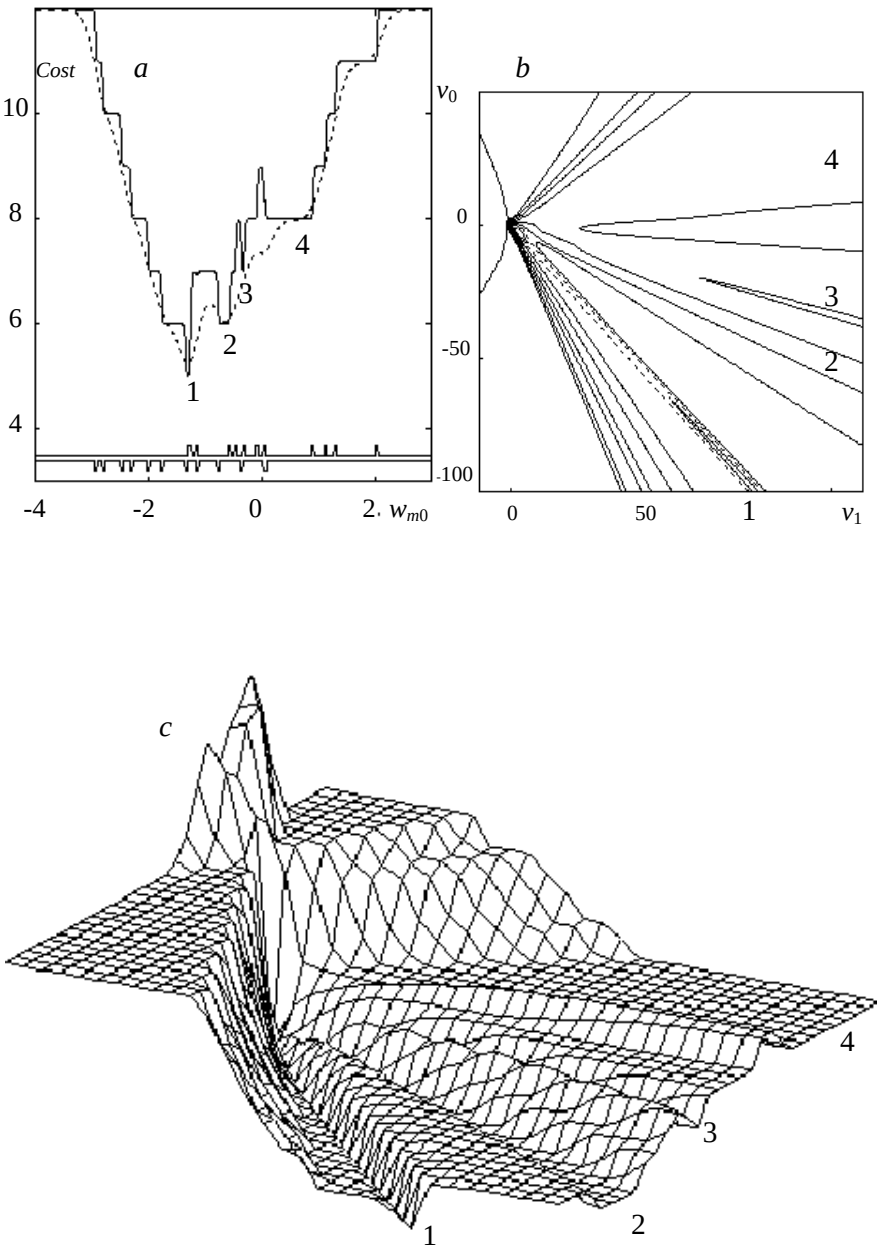

Fig. 4.5. *a* – two sections of the cost function of the sum squares versus w_{m0}, the threshold coefficient of DF, for the hard-limiting activation function (solid line) and for the soft-limiting activation function (dotted line); twelve uni-variate Gaussian N(0,1) and N(1.68,1) training observations from each class are depicted by positive and negative impulses; *b* – 2D plot of the cost function; *c* – 3D plot of the cost function: long trenches (the minima) are marked by numbers 1 (the global minimum), 2, 3, and 4 (the local minima).

In the back propagation stochastic gradient training, the weights wandered for a while around an origin of the co-ordinate axis until zero empirical error was achieved (≈250 cycles). Afterwards, the weights began increasing steadily until they reached some "local trench" and moved further along this trench. We see changes in nine hidden layer weights and the output layer weights during the training in five weight trajectory plots, presented in Figure 4.6.

In Figure 4.6, the weights values are plotted on the axes *x* and *y*, however, they are different for each of the 5 plots:

- plots 1, 2, and 3 represent the weight pairs - v_{j1} versus v_{j2} of three hidden layer neurones (j = 1, 2, 3);
- plots 4 and 5 represent the weight pairs w_1 versus w_3 and w_2 versus w_3 of the output-layer neurone.

The most distant points of each of the five weight trajectory plots correspond to the weights obtained after 2000 additional training cycles, when the targets 1.0 and 0.0 were used.

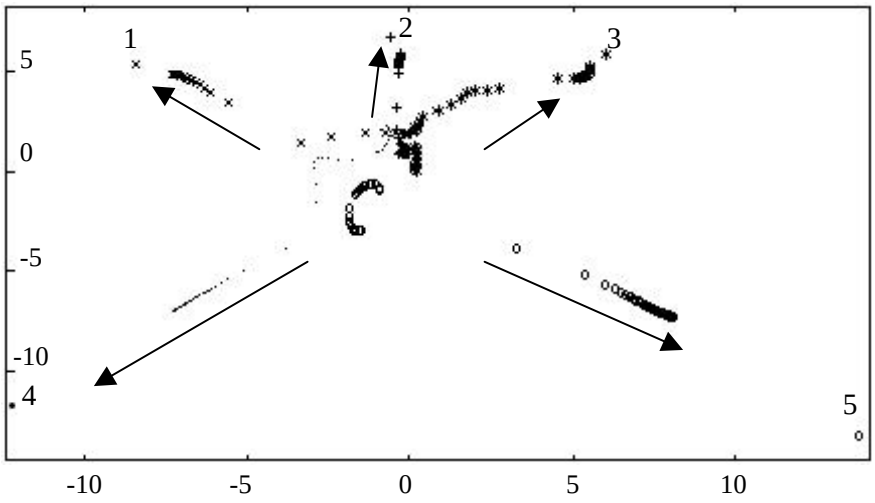

Fig. 4.6. "Explosion of universe" in MLP training: 1, 2 and 3 show the hidden layer weights; 4, 5 show the output layer weights (for each "weight history" the co-ordinates are different).

This example demonstrates that the local minima positions depend on the target values. A linear proportional increase in the values of the weights indicates that approximately correct positions of the individual hyperplanes have already been found during the preceding training iterations. In subsequent iterations, the network increases the magnitudes of the weights without changing the positions of discriminating hyperplanes notably. An increase in the magnitudes of the weights of the hidden layer neurones sharpens the curvatures of the decision boundary.

Thus, the control of the magnitudes of the weights permits us to obtain optimal curvatures of the decision boundary near the intersection of two or more of the discriminating hyperplanes. We have presented an illustration in Section 1.2 (Figure 1.5).

When the empirical error is zero, the output-layer neurone performs decisions in a linearly separable space. Therefore, we do not have the "curvature" problems there. Consequently, the output-layer weights have a tendency to increase without bound. We can notice that after 2000 additional training cycles with the targets 1.0 and 0.0, points 4 and 5 of the output neurone move much farther than the points 1, 2, and 3, corresponding to the hidden neurones.

As in the SLP classifier, each hidden neurone is characterised by n+1 weights, however the position of the hyperplane corresponding to this neurone is determined by n independent parameters. Thus, one parameter in each neurone is excessive in determining the position of the hyperplane. In such a way, at the end of the training process, the cost function surface of MLP has a multidimensional flat area. For the single layer perceptron, we had the one-dimensional "long narrow ditch". In that case, the BP training is extremely slow.

Example 7. In Figure 4.7, we have 2D and 3D plots of the cost function surface of an *insufficiently trained* MLP with eight hidden units. The weights are still small. In the two-dimensional space of two hidden unit weights, there exists only one, global, minimum point and there are no local minima. However, the error surface consists of a great number of flat plane surfaces, steep slopes and wide or narrow ditches, and, in fact, it is very difficult to find the global minimum point.

We have depicted only the two-dimensional section of the cost function surface. The number of weights is much larger actually and the situation is much more complex. It is not easy to find the right direction in which to move in this perplexity of ditches and plateaux. In the BP training with a constant learning step, the training algorithm gradually stops at one flat local minimum. Numerous simulation studies with data artificially generated by a teacher MLP indicate that despite the fact that, in principle, the MLP can separate the training data without error, often the back propagation training fails to do this.

One more specific peculiarity of the MLP classifiers is that there exist a number of different combinations of the network weights that result in approximately the same classification accuracy. The amount of satisfactory solutions is usually significantly larger when the number of neurones in the hidden layer is high, e.g. interchange of two hidden neurones and corresponding output layer weights does not change the decision boundary.

When the training vectors of the opposite pattern classes overlap, larger numbers of local ditches can be found. The situation is similar to that of the single-layer perceptron already discussed at the beginning of this section. One more characteristic difficulty in the MLP training arises when several hidden neurones become very similar. Each new initialisation and subsequent training of MLP can lead to a new configuration of the hyperplanes produced by the hidden neurones. Therefore, one should train the perceptron many times, each time starting from different initial weights.

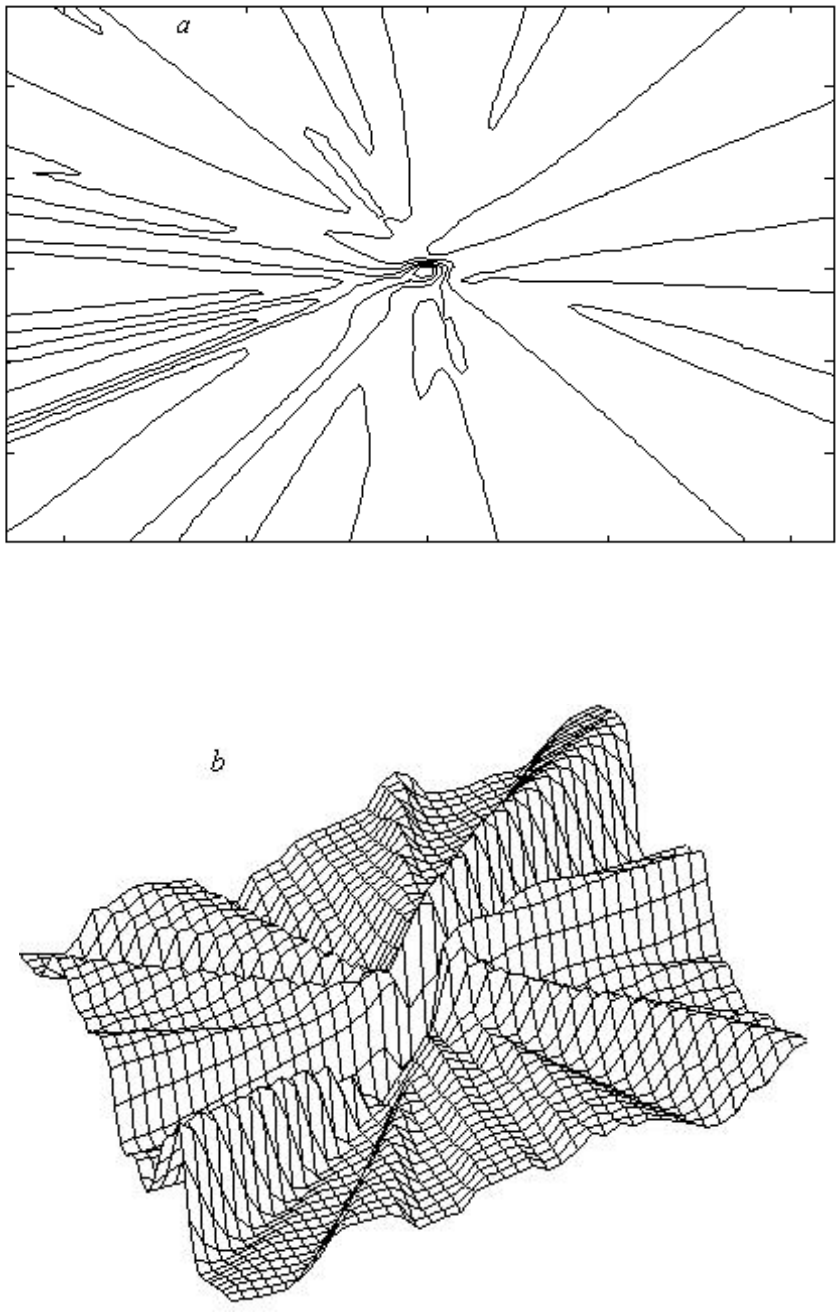

Fig. 4.7. 2D and 3D cost function plots of an undertrained MLP (with small weights) with eight hidden units.

4.3.3 The Gradient Minimisation of the Cost Function

The gradient minimisation of the cost function is favourable in the sense that, in SLP training, it is possible to obtain regularised discriminant analysis (RDA). In principle, to speed up the optimisation procedure one can utilise more sophisticated second order methods (e.g., a Newton method or its modifications), however, in this case, we bypass RDA. In this section we consider an unfavourable property associated with the eigenvalues of the training data when we employ the traditional gradient (back-propagation) training procedure.

Such a situation arises when the training-set size N is small in comparison with n, the number of inputs, when the original features are highly correlated and when we map the input vectors into a high-dimensional space. Then the new features become highly correlated and the sample covariance matrix becomes singular or almost singular. A number of eigenvalues of this matrix can be small or equal to zero.

As in the previous sections, consider the linear SLP and the traditional sum of squares cost function and the total gradient descent optimisation procedure. Using simple matrix algebra, we can rewrite the sum of squares cost function in the following way

$$cost_t(\boldsymbol{V}) = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_j^{(i)} - (\boldsymbol{V}^T \boldsymbol{X}_j^{(i)} + v_o))^2 = cost_{min} + (\boldsymbol{V}-\hat{\boldsymbol{V}})^T \mathbf{K} (\boldsymbol{V}-\hat{\boldsymbol{V}})$$

$$= cost_{min} + (\boldsymbol{V}-\hat{\boldsymbol{V}})^T \boldsymbol{\Phi}\boldsymbol{\Phi}^T \mathbf{K} \boldsymbol{\Phi}\boldsymbol{\Phi}^T (\boldsymbol{V}-\hat{\boldsymbol{V}}) = cost_{min} + \boldsymbol{U}^T \boldsymbol{\Lambda} \boldsymbol{U} = cost_t(\boldsymbol{U}), \quad (4.17)$$

where $\hat{\boldsymbol{V}} = \mathbf{K}^{-1}(\hat{\boldsymbol{M}}_1 - \hat{\boldsymbol{M}}_2)$, $\boldsymbol{\Phi}$ is an orthogonal eigenvectors matrix of the matrix

$$\mathbf{K} = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} \boldsymbol{X}_j^{(i)} \boldsymbol{X}_j^{(i)\,T}, \text{ such that } \quad \mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^T,$$

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$ is an eigenvalues matrix of $\mathbf{K}$, and $\boldsymbol{U} = \boldsymbol{\Phi}^T(\boldsymbol{V}-\hat{\boldsymbol{V}})$. Note,

the term $cost_{min} = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_j^{(i)} - (\hat{\boldsymbol{V}}^T \boldsymbol{X}_j^{(i)} + \hat{v}_o))^2$ does not depend on $\boldsymbol{U}$.

Representation (4.17) allows us to express the gradient in a very simple form

$$\frac{\partial\ cost_t}{\partial \boldsymbol{U}}\Big|_{\boldsymbol{U}=\boldsymbol{U}_{(t)}} = 2\ \Lambda\ \boldsymbol{U}_{(t)}.$$

Then the gradient training rule of the transformed weight vector $\boldsymbol{U} = \boldsymbol{\Phi}^T(\boldsymbol{V}\text{-}\hat{\boldsymbol{V}})$ is

$$\boldsymbol{U}_{(t+1)} = \boldsymbol{U}_{(t)} - \eta\ \frac{\partial cost_t}{\partial \boldsymbol{U}} = \boldsymbol{U}_{(t)} - \eta 2\ \Lambda\ \boldsymbol{U}_{(t)} = (\mathbf{I} - 2\ \eta\ \Lambda)\ \boldsymbol{U}_{(t)}. \tag{4.18}$$

In order to ensure convergence to zero, the following conditions should be satisfied

$$|\ 1 - 2\eta\lambda_i\ | < 1 \qquad \text{for} \quad i = 1, 2, \ldots, n. \tag{4.19}$$

Let λ_{max} be the maximal eigenvalue of **K**. Then condition (4.19) can be formulated as

$$0 < \eta < 1/\lambda_{max}. \tag{4.20}$$

The convergence time depends on all spectra of eigenvalues of the matrix **K**. In order to obtain the fastest convergence it is suggested that one should use

$$\eta_{opt} = \frac{1}{\lambda_{min} + \lambda_{max}} \quad \text{or} \quad \eta_{opt} = \mu\,\frac{1}{2\lambda_{max}}, \text{ where } 0 < \mu < 1. \tag{4.21}$$

We see that in situations where the training data is almost singular, i.e. we have an immense difference between the smallest and largest eigenvalues of the data covariance matrix, it is difficult, if not impossible, to ensure fast BP convergence of the training algorithm.

4.4 Generalisation of the Perceptrons

4.4.1 Single Layer Perceptron

4.4.1.1 Theoretical Background

If the training conditions E1 – E4 are satisfied, then in back-propagation, one can obtain seven statistical classifiers with different generalisation error – training-set size properties. At first we have EDC with generalisation error (3.7), then RDA with (3.18), and the Fisher classifier with (3.10). If the training-set is very small, then we approach the Fisher classifier with the pseudo inversion of the covariance matrix (Equation (3.17)). Later, with an increase in the magnitudes of the weights, we approach the minimum empirical error classifier discussed in Section 3.6.4. Consequently, the generalisation error of the SLP classifier dramatically depends on the method utilised to train SLP. In the case of a very good initialisation

(conditions E1 – E4 and spherically Gaussian data), the generalisation error of SLP can be determined by that of EDC. Suppose the weights are initialised randomly in a wide interval. Then the generalisation error of SLP is much higher. In the zero empirical error case, it can be determined by the left columns of Table 3.3.

An important aspect of the SLP classifier training is that the generalisation errors of some classifiers (EDC and ZEE) depend on a hidden (intrinsic, effective) dimensionality of the data. Unfortunately, the definition of the "intrinsic dimensionalities" for each classifier is different. The type and statistical properties of the SLP classifier obtained in the back propagation training, depend on the number of iterations, initialisation, learning step and peculiarities of the data. Therefore, most often we do not know which particular type of statistical classifier we have at each moment. Hence, the numerical values of the generalisation error and estimates of the necessary training-set size discussed in Chapter 3 are difficult to use practically. Whereas the learning quantity $\kappa = \bar{\varepsilon}_N^{\mathrm{SLP}} / \varepsilon_\infty^{\mathrm{SLP}}$, the ratio between the generalisation and the asymptotic errors, depends on the training conditions, it is impossible to express the generalisation error of SLP by a single formula. Thus, in order to guide oneself in the SLP training process, the researcher needs to take into account that in each particular stage of the training process, *different generalisation error equations* are applicable.

4.4.1.2 The Experiment Design

In order to illustrate the effect of the training conditions on the type of classification rule and its generalisation error, we present several experimental results. In the simulations, we used the GCCM and SG data, the batch-mode BP training algorithm and the sum of squares cost function with the *sigmoid* activation function (1.6). In the experiments, if not stated otherwise, the conditions E1 – E4 were fulfilled. Most often we used the target values $t_1 = 1$ and $t_2 = 0$. Close targets, e.g. $t_1 = 0.55$ and $t_2 = 0.45$, force the sigmoid activation function to act as the linear function. Thus, for $t_1 = 0.55$ and $t_2 = 0.45$, after minimising the cost function we should obtain the Fisher DF. To consider a more realistic situation, in some of our experiments, we analysed the proximal targets $t_1 = 0.9$, $t_2 = 0.1$, the values recommended by Rumelhart *et al*. (1986).

At different moments, the training process was stopped for an instant and the generalisation error calculated. In this particular analysis, we know that the true distribution of the data is Gaussian. Therefore, to find the generalisation error ε_N we do not need an independent test set – we can calculate ε_N analytically.

Let the sample estimates of weights of the linear discriminant function (DF) be $\hat{v}_0$ and $\hat{\boldsymbol{V}}$, the true means be $\boldsymbol{M}_1$, and $\boldsymbol{M}_2$, and the CM, $\bar{\boldsymbol{\Sigma}}$. Then simple matrix algebra allows us to calculate the conditional means $\hat{\boldsymbol{V}}^T \boldsymbol{M}_i + \hat{v}_0$ (i=1,2), and the variance $\hat{\boldsymbol{V}}^T \bar{\boldsymbol{\Sigma}} \hat{\boldsymbol{V}}$ of the discriminant function $g(\boldsymbol{X}) = net = \hat{\boldsymbol{V}}^T \boldsymbol{X} + \hat{v}_0$. The linear discriminant function is a Gaussian random variable. Therefore, the conditional generalisation error (conditioned on $\hat{v}_0$ and $\hat{\boldsymbol{V}}$) is

$$\varepsilon_N^{\mathrm{SLP}} = \frac{1}{2}\Phi\{-\frac{\hat{v}_0 + \hat{\boldsymbol{V}}^T \boldsymbol{M}_1}{\sqrt{\hat{\boldsymbol{V}}^T \overline{\boldsymbol{\Sigma}} \hat{\boldsymbol{V}}}}\} + \frac{1}{2}\Phi\{\frac{\hat{v}_0 + \hat{\boldsymbol{V}}^T \boldsymbol{M}_2}{\sqrt{\hat{\boldsymbol{V}}^T \overline{\boldsymbol{\Sigma}} \hat{\boldsymbol{V}}}}\}. \tag{4.22}$$

This expression allows us to calculate the generalisation error analytically.

4.4.1.3 The SLP and Parametric Classifiers

If the empirical classification error is small, the target value has a significant influence on the magnitudes of the weights and on the result obtained. Thus, the target values can guide SLP both to the parametric and to nonparametric (structural) classification rules. The generalisation error graphs in Figure 1.10 are a nice illustration. The graphs differ in the target values used. Recall that for Figure 1.10 the 20-variate GCCM data **C** was used and the training-set size was $\overline{N}$ = 14 vectors from each class. In order to make the training process faster, a slightly increasing learning step (η=10 ×1.0005^t) was utilised. In both cases, after the first iteration we have EDC with $\varepsilon_N^{\mathrm{SLP}}$ = 0.058. At the beginning of training, the generalisation error decreases: there we have the linear regularised discriminant analysis. The different target values, however, lead to different classification rules later. With targets 0.9 & 0.1 (graph 1) we are approaching the standard Fisher DF with $\varepsilon_N^{\mathrm{SLP}}$ =0.093. With targets 1 & 0 (graph 2) we move towards the minimal empirical error and the maximal margin classifiers. We see that for an extremely small training-set size (n=20, N=28) and good initialisation (after the first iteration we have EDC whose weight vector serves as a very successful initialisation) case, the targets 1 & 0 result in significantly smaller generalisation error. Thus, the targets 0.9 & 0.1 can help to obtain parametric statistical classifiers.

Figure 4.8 demonstrates experimental learning curves: the expected generalisation error versus training-set size. Each curve is an average value obtained from 50 randomly selected training sets (the 20-variate GCCM data **C**). The SLP was trained with the targets 0.9 & 0.1 and compared with the learning curves of EDC, the Fisher linear classifier and the regularised DA (with the optimal value of λ). In Section 3.4.8 (Figure 3.4), we demonstrated that the learning curve of the Pseudo-Fisher classifier exhibits *peaking behaviour* in the interval ($1< N < n$). In the experiment with data **C**, the Fisher classifier also shows a minimum close to $n/2$=10. The experiment (Figure 4.8*b*, curves 2 and 3) confirms theoretical findings. The learning curve of the non-linear SLP with targets 0.9 & 0.1 (2*a* in Figure 4.8*a*) also exhibits a clear *peaking behaviour* in the interval ($1 < N < n$). The shape of this curve resembles experimental (curves 3 and 2 in Figure 4.8*b*) and theoretical (Figure 3.4) curves for the standard Fisher DF and the Pseudo-Fisher DF. The learning curve of SLP after the first iteration, (curve 1 in Figure 4.8*a*), coincides with that of the EDC, (curve 1 in Figure 4.8*b*). Both experimental curves are very close to the theoretical one. Both RDA with optimal regularisation parameter λ_{opt}, (curve 4 in Figure 4.8*b*) and SLP with optimal number of iterations t_{opt} (curve 4 in Figure 4.8*a*), result in the lowest classification errors. Note, λ_{opt} and t_{opt} were determined individually for each training-set.

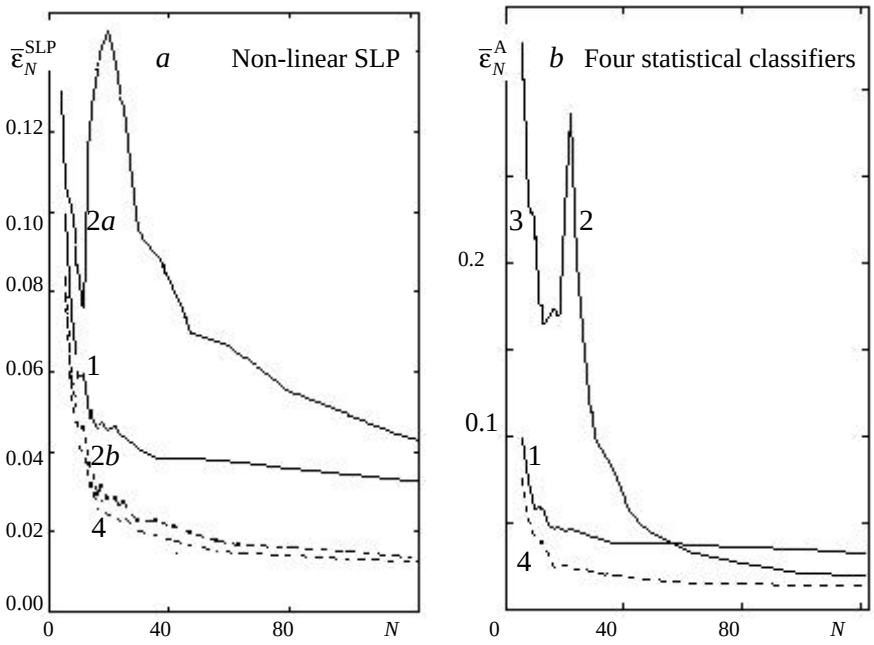

Fig. 4.8. Scissors effect in practice: the generalisation error versus the training set size N. (*a*) the non-linear SLP: 1 – after the first iteration, 2*a* – after 500 iterations (targets 0.9 & 0.1), 2*b* – after 100 iterations (targets 1 & 0), 4 – after the optimal number of iterations; (*b*) the statistical classifiers: 1 – EDC, 2 – Fisher DF, 3 – Pseudo-Fisher DF, 4 –regularised DA for optimal λ (Reprinted from *Neural Networks*, 11:297–313, Raudys, Evolution and generalization of a single neurone, 1998, with permission from Elsevier Science).

For $N > 1.5\,n$, the experimental learning curve 2 in Figure 4.8*b* becomes very close to the theoretical curve 2 for the Fisher DF in Figure 3.4. The same can be said about the experimental learning curve 4 in Figure 4.8*a* of SLP after the optimal number of iterations t_{opt} (targets 1 & 0), and curve 4 in Figure 4.8*b* for the regularised DA with the optimal value of the regularisation parameter λ_{opt}. The curve 2*b* in Figure 4.8*a* corresponds to targets 1 & 0. In this case, the target values force the activation function to act as the non-linear function. For this type of data, the targets 1 & 0 allow us to obtain essentially smaller generalisation errors, and confirms that the *target values are very important*.

Figure 4.8*b* demonstrates an obvious scissors effect: for small training sets (up to $N \approx 60$) it is preferable to use the simple structured EDC instead of the complex Fisher linear DF, and vice versa. That is, in large training-set cases, the Fisher linear DF can be used more efficiently. In principle, similar conclusions can be made for the SLP:

- in the small training-set case, it is preferable to train the SLP for a small number of iterations,
- in the large training-set case, one needs to use more iterations.

Other regularisation factors, such as target and learning step values, affect the training process simultaneously and the problem of finding optimal values for all of these parameters is not easy. Theoretical considerations concerning the effective dimensionality n^* of EDC indicate situations where SLP can be perfectly trained by using particularly small training sets. In Section 3.3 we have analysed an exceptional case: the 100-variate GCCM data model **D** with effective dimensionality n^* close to 1: $n^* \approx 1.05$, and asymptotic error $\varepsilon_\infty^E = 0.03$. Theoretical calculation gives the generalisation error of EDC $\bar{\varepsilon}_N^E = 0.0318$. This theoretical estimate advocates that for this data, the SLP classifier also can be trained with an exceptionally small training-set. In a series of 10 experiments with training sets containing five 100-variate vectors from each class, a very small generalisation error was obtained: the EDC yielded, on average, 0.039 error with the standard deviation 0.009. The same result was obtained for the SLP after the first iteration.

Similar theoretical considerations concerning the effective dimensionality n^* indicate situations where it is difficult to train the SLP classifier. The 100-variate GCCM data model **E** with $n^* \approx 10^{10}$ is an excellent example. Theoretical calculation gives the generalisation error of EDC $\bar{\varepsilon}_N^E = 0.4997$. In a series of 10 experiments with training sets of size $\bar{N} = 200$, a very high generalisation error was obtained - EDC yielded, on average, 0.4997. After the first iteration, SLP gave the same result. The Fisher DF, however, yielded a "reasonable" error 0.058, i.e., only 1.93 times higher than the asymptotic error $\varepsilon_\infty^E = 0.03$.

Thus, "almost singular" data is a very difficult problem for the BP training (see Section 4.3.3). In such a situation, a "decorrelating" transformation $\boldsymbol{Y}=\mathbf{T}\boldsymbol{X}$ is very helpful. An $n \times n$ transformation matrix can be obtained from $\hat{\boldsymbol{\Sigma}}$, an estimate of the covariance matrix: $\mathbf{T} = \Lambda^{-1/2}\Phi^T$, where Φ is an orthonormal $n \times n$ matrix such that $\Phi^T \hat{\boldsymbol{\Sigma}} \Phi = \Lambda$ (diagonal matrix of the eigenvalues). Then, in a new space Ω_Y, again we obtain the EDC after the first iteration, however, this classification rule is equivalent to Fisher's rule in the original Ω_X space. We will present more details in Chapter 5.

4.4.1.4 The SLP and Structural (Nonparametric) Classifiers

In non-linear SLP training, the weights can become very large. In such cases, SLP can become similar to the minimum empirical error and the maximum margin classifiers. In the random search optimisation (the Gibbs algorithm) training, the generalisation error of the zero empirical error classifier depends on the prior distribution of the weight vector (Section 3.6.4). We hypothesise that in the non-random (gradient descent) training, the starting position $v_{0(0)}$, $\boldsymbol{V}_{(0)}$ of the weight vector can affect the generalisation error too. Simulation experiments confirm this hypothesis.

Let the pattern classes be spherically Gaussian $N_X(\boldsymbol{M}_1, \mathbf{I})$, $N_X(\boldsymbol{M}_2, \mathbf{I})$. Then the optimal Bayes decision rule is the Euclidean distance classifier with weight vector $\boldsymbol{V}^E = \Delta\boldsymbol{M} = \boldsymbol{M}_1 - \boldsymbol{M}_2$, $v_0 = -\frac{1}{2}(\boldsymbol{M}_1 + \boldsymbol{M}_2)^T \boldsymbol{V}^E$. The predictive Bayes approach to

designing the classification rules (Section 2.4.2) says that the sample based EDC is the optimal decision rule when the prior distribution of the vector $\Delta \boldsymbol{M}$ is spherically Gaussian. Therefore, one can expect that the sample based vector $(\hat{\boldsymbol{V}}^{\mathrm{E}},\ \hat{v}_0)$ can become a good starting position in the adaptive training of the SLP classifier. If the training conditions E1 – E4 are fulfilled, then just after the first iteration we have EDC. In other words, just after the first iteration we can obtain the best classifier, and do not need to train the perceptron any more. If we start training from random weights or if the pattern classes differ from the spherical Gaussian model, we will not have the best classifier after the first iteration.

Two learning curves, depicted in Figure 1.9, are typical examples of experimental results obtained with the spherical Gaussian data. When starting from zero initial weights, just after the first iteration we have the best weight vector – the sample EDC. Hence, we have a constant increase in the generalisation error later (curve 2). After starting from a randomly generated, inexact initial weight vector, the training process "corrects" the initial weight vector. Therefore, the generalisation error reduces at first. Later on, however, the training procedure leads to the maximal margin classifier. The maximal margin (support vector) classifier is far from being the best one for the spherical Gaussian patterns. Therefore, after reaching the minimum, we obtain an increase in the generalisation error, and gradually we approach the learning curve, which is obtained in the zero weight initialisation case (curve 1, in dots, iterations 26 – 150 only).

The minimum of curve 1 with random initialisation is notably higher than the minimum of the generalisation error with the zero weight initialisation. This experiment witnesses once more that random initialisation and the search for the *maximal* margin is not always the best strategy in classifier design. The correct initialisation can reduce the generalisation error. Thus, almost correct perceptron weights can store a large amount of useful information. In Section 4.5.2 we will consider this important problem in more detail.

Simulation experiments indicate that with an increase in the width of the weights initialisation interval, the generalisation error of the SLP classifier increases and approaches the theoretical value of the mean generalisation error of the ZEE classifier calculated for the random Gaussian priors (Section 3.6.4). If the initialisation interval becomes too wide, then just before the first iteration we can have very large weights and the saturation of the activation function. Then the perceptron does not learn or learns very slowly. In such situations, we can have a very high classification error.

We see that the number of weights is far from being a *sufficient parameter* to determine the small sample properties of the single layer perceptron. The parameters of the perceptron and the training conditions are very important too.

4.4.2 Multilayer Perceptron

The non-linearity of the activation function, the continuous evolution of the output layer SLP and the expansion of the hidden layer weights make a precise analysis of the generalisation error of the MLP classifiers almost impossible. The theoretical considerations, however, can explain only the generalisation behaviour

of the SLP in the output layer. An accurate investigation of the generalisation error of the MLP can be performed by means of simulation. In this section, we present theoretical and experimental evidence that advocates that, while determining the generalisation error of the MLP classifier, the number of network weights is a deficient parameter of the complexity.

4.4.2.1 Weights of the Hidden Layer Neurones are Common for all Outputs

In Chapter 3, it was demonstrated for the parametric statistical classification rules that the parameters of the class distribution density functions that are common for both pattern classes asymptotically (when the number of training samples and the dimensionality are increasing) do not affect the expected probability of misclassification. In fully connected feedforward multilayer perceptrons, the output of each hidden neurone is transferred to all output layer neurones. Any deviation, $\Delta v_{\beta\alpha}$, of each single hidden layer neurone weight, $v_{\beta\alpha}$, affects all outputs of the network. Consequently, it can be said that the weights of the hidden layer neurones of MLP are common to all pattern classes.

An analogy with statistical pattern classification hints that the theoretical results concerning parametric statistical classifiers possibly may be extended to the multilayer perceptrons. Unfortunately, there are no such theoretical results for the multilayer perceptrons so far. Simulation experiments, however, advocate that in certain situations, this guess is correct.

The simple data. In Figure 4.9*a* we have two graphs which show the mean generalisation error versus the number of dimensions, n, obtained for very simple structured data − two spherical Gaussian classes. The lower graph corresponds to the multilayer perceptron with five units ($h = 5$) in the hidden layer trained by the stochastic gradient BP algorithm, sigmoid activation function, targets $t_1 = 0.9$ and $t_2 = 0.1$. The upper graph corresponds to the standard Fisher linear DF. The same training data sets were used to train both classification algorithms. Apparently, for this kind of the data the MLP classifier with $(n+1)h+(h+1)$ weights is less sensitive to the dimensionality of the input vector than is the standard Fisher DF with only $n+1$ weights. It agrees with the above guess, based on the analogy with the statistical classifiers. Thus, one may speculate that the accuracy of the weights that are common for all outputs are less important than the accuracy of the weights of the final (output) layer.

In addition to formal statistical analysis, there is an intuitive explanation. For very simple distributions of the pattern classes, the neurones of the hidden layer are similar. Therefore, a minor inaccuracy in determination of one weight of the hidden-layer neurone can be reduced by effects of other similar hidden units, and, most importantly, by subsequent tuning of the weights of the output layer. In the above example, we have only six weights to be adjusted in the output layer neurone and this number does not change with an increase in the number of inputs. Thus, for simple distributions of the pattern classes, the MLP classifier adapts to the simplicity of the problem and can perform satisfactorily even in a case where the number of weights is significantly larger than the number of training vectors.

Note that in the example described above, there were no clear local minima effects.

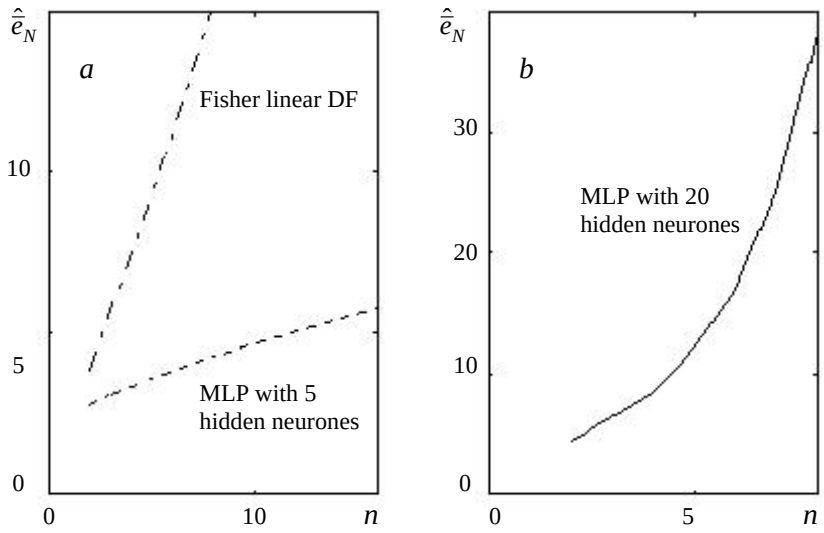

Fig. 4.9. Generalisation error (in percentage) versus dimensionality, n, in: *a*) an easy duty and *b*) heavy duty cases.

The complex data. In a case of classification with pattern classes having complex-shaped distributions, the hyperplanes that correspond to different hidden layer neurones are different. Thus, a small alteration of the hidden unit weight can have opposite effects on different outputs of the network and cannot be compensated by tuning the weights of the output layer. Contrary to the case of simply-shaped pattern classes, in this situation, the network can become more sensitive to inaccuracies in the hidden layer weights.

The heavy-duty spherical non Gaussian classes **SF2** considered earlier can serve as an example of such a situation. Here, vectors of the first class are inside the second one: vectors from a bi-variate spherical Gaussian distribution N_X (0, **I**) inside the circle, $(x_1^2 + x_2^2 = 1.1)$, were chosen for class ω_1, and vectors outside the larger circle, $(x_1^2 + x_2^2 = 1.45)$, were chosen for class ω_2. To make the training task more difficult, in this simulation experiment, we added six Gaussian N_X (0, **I**) zero mean uninformative features. The Bayes probability of incorrect classification is zero, however, the pattern classes are non-linearly separable. We will use this eight-variate data once more (in Section 4.6.8) and will call it **SF8**. The training sets were composed of $\bar{N} = 70$ vectors from each pattern class, and the test set contained $Nt = 1000 = 500 + 500$ vectors. The graph in Figure 4.9*b* shows that in this heavy-duty case, the generalisation error of MLP increases exponentially with an increase in the number of the features. Its small sample

behaviour is more similar to the behaviour of the nonparametric local classifiers (*k-NN*, Parzen window and multinomial classifiers).

Evidently, MLP can adapt to the complexity of the pattern classification problem. In the "easy-duty" case, the multilayer perceptron recognises the simplicity of the problem and exhibits even better small training-set behaviour as the statistical classifiers do. Therefore, *the behaviour of MLP is dual: it depends on the data to be classified*. The MLP classifier can behave similarly to the parametric or the nonparametric local classifier.

4.4.2.2 Intrinsic Dimensionality Problems

Many researchers claim that the "real dimensionality" of real world data is usually much smaller than the number of features. This hypothesis is based on an assumption that in real life, only a few unknown independent factors influence the variability of a large number of the features measured. This unknown, but intuitively supposed, dimensionality of the data is called *the intrinsic dimensionality*. In Chapter 3, it was demonstrated that the small sample properties of EDC, the Parzen window classifier, and the zero empirical error classifier depend on the intrinsic (effective, hidden) dimensionality of the data. There exist theoretical and experimental arguments that it is also true for the non-linear SLP classifier. Unfortunately, there is no unique definition of the "intrinsic dimensionality". For EDC, it was the parameter n^* (Equation (3.7)), for the PW and zero empirical error classifiers it was r, the number of non-zero eigenvalues of the GCCM model. In the GCCM data model case, the Fisher classifier, however, is sensitive to all features. Consequently, the dimensionality, r, of the subspace where the data points are situated is not a sufficient definition of the intrinsic dimensionality. A rigorous definition depends both on a true distribution of the pattern classes and on the type of the classifier used.

Example 8. In Table 4.1 we have average values (upper rows) and standard deviations (lower rows) of the generalisation errors of the MLP classifiers for three pairs of 100-variate GCCM data models with Mahalanobis distance $\delta = \delta^* = 3.76$ (asymptotic error is 0.03). The three data models have different effective dimensionality parameters ($n^* = 1.05$, 100 or 10000). The number of hidden neurones was either $h = 10$ or 100 (1021 or 10201 the perceptron's weights), and the target values were: $t = 0.7$ and 0.3; $t = 0.9$ and 0.1; or $t = 1.0$ and 0.0 (the standard sigmoid activation function was used). A standard gradient descent (back-propagation) MLP training algorithm from a Matlab neural networks toolbox was used to train the networks. The generalisation errors were evaluated in 10 experiments with 10 randomly chosen training sets containing $\overline{N} = 10$ multivariate vectors from each of two pattern classes. Note, $N << n=100$. To estimate the generalisation error, 500+500 independent test set vectors were used. For each set of experimental conditions in Table 4.1, we present two generalisation error values:

- an optimally stopped perceptron (on the left of "/"), and
- at the end of the training process – after 2,000 iterations (on the right, after "/").

Table 4.1. Mean values (upper rows) and standard deviations (lower rows) of the generalisation errors of the MLP classifiers for three pairs of Gaussian pattern classes sharing a common covariance matrix. Dimensionality $n = 100$, training-set size $N = 10 + 10$.

n^*	h = 10 *target* = 0.7	h = 10 *target* = 0.9	h = 10 *target* = 1.0	h = 100 *target* = 0.7	h = 100 *target* = 0.9	h = 100 *target* = 1.0
1.05	.173/.214	.061/.070	.046/.050	.062/.091	.049/.077	.046/.069
	.059/.070	.020/.024	.014/.016	.021.035	.015/.027	.013/.020
100	.379/.435	.377/.433	.374/.435	.397/.411	.316/.323	.249/.308
	.065/.053	.062/.049	.058/.045	.053/.049	.047/.044	.036/.037
10000	.476/.486	.462/.471	.455/.463	.463/.481	.463/.481	.463/.487
	.015/.022	.025/.028	.025/.026	.028/.033	.023/.026	.022/.023

Table 4.1 confirms that the data type (the effective dimensionality n^*) and the training conditions (the target values and stopping moment) are very important. The simulation results obtained for the spherical data ($n^* = n = 100$) can be explained by the theoretical results presented in Table 3.1. For the spherical data any hidden neurone does not have any preference over other ones. Thus, one may make an assumption that after training, the weights of the hidden layer neurones become random. Assume the weights of the hidden layer are independent zero-mean random variables and the activation functions of the hidden layer neurones behave as linear functions. For simplicity, suppose the distribution of the weights is Gaussian. Then for n-variate spherically Gaussian data, it is possible to calculate the Mahalanobis distance of the populations at outputs of the hidden layer with h neurones $\delta_{\text{hidden}} \approx \delta\sqrt{h/(n+h-1)}$.

For 10 hidden units we have $\delta_{\text{hidden}} \approx 1.14$ with asymptotic error $\varepsilon_\infty \approx 0.29$, and for 100 hidden units we have $\delta_{\text{hidden}} \approx 2.666$ with $\varepsilon_\infty \approx 0.091$. Under the random weights assumption, for $\delta_{\text{hidden}} \approx 1.14$ and n=10 (h = 10) Equation (3.7) for EDC gives the expected error $\bar{\varepsilon}_N^{\text{E}} = \approx 0.36$. For $n = 100$ (h = 100) and $\delta_{\text{hidden}} \approx 2.666$ we calculate $\bar{\varepsilon}_N^{\text{E}} \approx 0.25$. Note, in this analysis, the number of hidden units, h, plays role of dimensionality in the output layer SLP. These $\bar{\varepsilon}_N^{\text{E}}$ values are fairly close to the average experimental values of the optimally stopped MLP obtained for h = 10 and h = 100 and the target values 1 & 0 ($\hat{\bar{\varepsilon}}_N^{\text{SLP}}$ = 0. 374 and 0.249).

The experimental values at the end of the training process are closer to the pessimistic theoretical generalisation error estimates of the ZEE classifier. Calculation according equations presented in Section 3.6.4 for ZEE classifier with random Gaussian priors for h=10 and $\delta_{\text{hidden}} \approx 1.14$ gives $\bar{\varepsilon}_N^{\text{ZEE}} = 0.47$. For h=100 and $\delta_{\text{hidden}} \approx 2.666$ we calculate $\bar{\varepsilon}_N^{\text{ZEE}} = 0.41$. For $n^* = 100$ Table 4.1 shows only slightly smaller error values at the end of training: $\hat{\bar{\varepsilon}}_N^{\text{SLP}} = 0.435$ and 0.308.

For target values 0.7 & 0.3, the weights of the output-layer neurone are small. Consequently, the activation function of this neurone actually behaves like a linear function. Therefore, in this case, we obtain a classifier which is similar to the standard Fisher (when $h < N$) or the Pseudo-Fisher classifier (when $h > N$). In Section 3.4.8, we have shown that if dimensionality exceeds the training-set size, then the generalisation error of the Pseudo-Fisher classifier is notably smaller than 0.5). In the experiment we obtain similar values.

For data with small n^* (n^*=1.05), EDC gives a small generalisation error. The same can be said about the MLP classifier. We also notice a parallelism between the performances of EDC and SLP classifiers for data with high intrinsic dimensionality (n^* = 10,000): both classifiers result high generalisation errors.

The above results indicate that one cannot reject a possibility that in certain pattern classification tasks, the weights of the hidden layer neurones of the MLP classifier to some extent are determined in a random manner. In cases where this is true, the number of weights in the hidden layer does not serve as a characteristic of the complexity of the network. Moreover, the hidden layer weights are common to all outputs. In this situation, the number of weights in the output neurone is the more important characteristic. On the other hand, owing to a constant evolution of the output neurone during the iterative training process, the number of weights in the output layer of the network also becomes an insufficient characteristic to determine a real effective complexity of the network.

4.4.2.3 An Effective Capacity of the Network

In Section 4.3.2, it was demonstrated that the local minima and flat areas of the cost function often make difficult the task of designing complex-shaped decision boundaries. Often, known optimisation methods do not allow one to realise a large number of dichotomies as predicted by VC dimension. In the output layer of the MLP, we have the single layer perceptrons. The complexity of the SLP classifier depends on the number of training iterations, the data model, peculiarities of the cost function, the training algorithm and its parameters. Therefore, the number of weights to be adjusted in the training process is a very rough and *insufficient characteristic of the network complexity*.

Kraaijeveld and Duin (1994) experimentally investigated a number of dichotomies that can be realised by MLP and back propagation training. They found that a practical, effective VC dimension is much smaller than that theoretically predicted by using the upper and even lower bound. We present below data from Kraaijeveld and Duin which show the dependence of the "effective capacity" – the average number of dichotomies obtained in the simulation study – versus h, the number of hidden units of MLP and n, the number of features.

Table 4.2. The estimated effective capacity d^* of MLP as a function of h, the number of hidden units. Dimensionality n = 2.

h	1	2	5	10	20	50	100
# of free parameters	5	9	21	41	81	201	401
effective capacity d^*	3	5	6	7	7	6	5

Table 4.3. The estimated effective capacity d^* as a function of n; $h = 100$.

n	1	2	5	10	20
# of free parameters	301	401	701	1201	2201
effective capacity d^*	4	5	7	11	15

We see that "the estimated effective capacity" may be orders of magnitude smaller than the theoretical capacity of the network. This proves once more that "the searching capabilities of the back propagation algorithm are very limited" (Kraaijeveld and Duin, 1994). In SLP training, we move from the simplest EDC to the regularised DA, up to the most complex maximal margin classifier. Undoubtedly, a similar complexity change effect exists in MLP training.

4.5 Overtraining and Initialisation

A characteristic phenomenon in ANN training is overtraining (overfitting). With an increase in the number of iterations, the generalisation error decreases at first, reaches a minimum and then begins to increase. We come across such network behaviour in Figures 1.9 and 1.10. The overtraining effect can be explained by the fact that in a finite training-set case, the network adapts to the training data too much. In this section, this important ANN training peculiarity is considered more thoroughly. We advocate that, this effect depends on:

- adaptation to the training data,
- the initial weight values of the perceptron and on
- the constant changing of the cost function in the BP training process.

4.5.1 Overtraining

One rational explanation of the overtraining effect of the single layer classifier is based on the fact discussed in Section 4.1: during the iterative training process, we obtain several classification rules of increasing complexity. In the finite training-set case, one of the classifiers results in the smallest generalisation error and appears to be the best. Thus, we need to choose a classifier of proper complexity. In Section 4.4.1.1, we hypothesised that in the small training-set case, it is preferable to train the SLP for a short time and, in the large training-set case, one needs to use more iterations. Therefore, with very large training-set sizes, one may train SLP until the minimum empirical error is obtained. When the data are spherically Gaussian (SG), however, it is reasonable to satisfy conditions E1 – E4 and train the perceptron for only one iteration. We have already seen such an example in Figure 1.9 where further training of SLP increases the generalisation error more than two times.

Theoretical considerations presented in Chapter 3 and in Section 4.1 can give conditions where the overtraining effect is very noticeable. For example, Table 3.1 indicates that for 50-variate spherical Gaussian data with Mahalanobis distance

δ = 4.56 (the Bayes and the asymptotic errors are 0.01) and training-set size $\overline{N}$ = 30, one can expect after the first iteration to obtain the generalisation error 0.016 (EDC). If one uses the targets 0.9 and 0.1, then in further training, SLP will approach the Fisher classifier with more than 10 times the generalisation error (0.197). If one uses targets 1 and 0, then in further training one will get the zero empirical error classifier with a slightly smaller generalisation error – 0.124 (Table 3.3). In both cases we obtain tremendous overtraining.

The overtraining effect can be explained also by the fact that there exist two different cost functions in the training process:

- $cost_t$, to be minimised during the training process, is based on the *training pattern vectors* and
- $cost_{gen}$ of the classification error measured on the *general population* (in practice, it is measured on a separate test or validation set); we had named this the conditional classification error or the generalisation error.

Each cost function has its own landscape in the multivariate weight space, and minimisation of the first function does not imply the minimisation of the second one.

Example 9. In Figure 4.10 in two-variate weight space, we have depicted a final weight vector $\hat{\boldsymbol{V}}_1$ that corresponds to a minimum of the sample-based cost function $cost_t$ and a true minimum $\boldsymbol{V}^*$ that corresponds to the true (asymptotic) classification error. Suppose, the classification error depends only on the distance of the weight vector from the ideal value $\boldsymbol{V}^*$. Let us start training from initial weight $\boldsymbol{V}_0^1$ and move to the sample-based minimum $\hat{\boldsymbol{V}}_1$ along a straight line. Then a minimum classification error will be obtained at $\hat{\boldsymbol{V}}_{opt}^1$, the closest point to $\boldsymbol{V}^*$. Thus, in a segment $\hat{\boldsymbol{V}}_{opt}^1$ - $\hat{\boldsymbol{V}}_1$ we have overtraining.

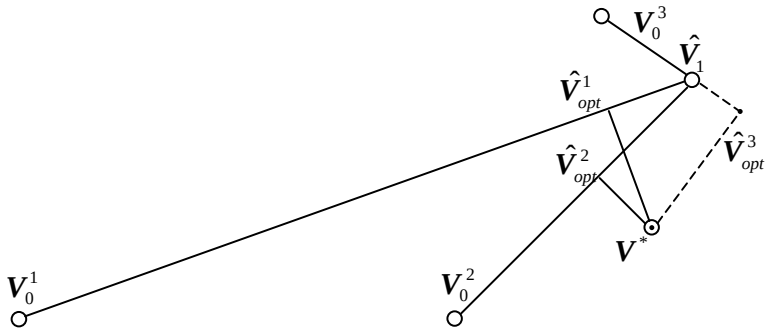

Fig. 4.10. Effect of initial weights on overtraining. We have overtraining when starting from $\boldsymbol{V}_0^1$ or from $\boldsymbol{V}_0^2$. However, starting from $\boldsymbol{V}_0^3$ does not lead to overtraining (α_{opt}>1, see Equation 4.33).

It is a consequence that instead of minimising the true cost, $cost_{gen}$, with the minimum in $\boldsymbol{V}^*$, we minimise the sample-based cost, $cost_t$, with the minimum in another place ($\hat{\boldsymbol{V}}_1$). If we start with $\boldsymbol{V}_0^2$ and move straight to $\hat{\boldsymbol{V}}_1$, we need to stop at $\hat{\boldsymbol{V}}_{opt}^2$. In comparison with $\hat{\boldsymbol{V}}_{opt}^1$, vector $\hat{\boldsymbol{V}}_{opt}^2$ results in a smaller generalisation error. If we start with $\boldsymbol{V}_0^3$ and move straight to $\hat{\boldsymbol{V}}_1$, we need to train until the end (the minimum of the cost function). In this case, we have no overtraining.

This graphical example advocates that the overtraining effect depends on the initial values. In the next subsection, we will perform more complete analytical investigation.

4.5.2 Effect of Initial Values

Figure 4.10 demonstrates that overtraining appears only in a part of training experiments and that it depends on the configuration of the starting, $\boldsymbol{V}_0$, final, $\hat{\boldsymbol{V}}_1$, and ideal, $\boldsymbol{V}^*$, weight vectors. In this section, we will show that in high-dimensional cases, the *overtraining almost always occurs*. To simplify analysis, consider an elementary linear model

$$y = \boldsymbol{X}^T \boldsymbol{V}^* + \xi = t, \tag{4.23}$$

where y is the output, $\boldsymbol{V}$ is the weight vector, $\boldsymbol{X}$ is n-dimensional input vector, $\boldsymbol{X} \sim N_X(\mathbf{0}, \mathbf{I})$, ξ is Gaussian $N(0, \sigma^2)$ independent noise and t stands for the target value.

Let us have the training set $(\boldsymbol{X}_1, y_1)$, $(\boldsymbol{X}_2, y_2)$, ..., $(\boldsymbol{X}_N, y_N)$ and let us utilise the standard sum of squares cost function in order to evaluate an unknown weight vector. Minimisation of the sum of squares cost results in

$$\hat{\boldsymbol{V}}_1 = \left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1} \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{X}_i y_i = \left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1} \frac{1}{N} \sum_{j=1}^{N} (\boldsymbol{X}_j \boldsymbol{X}_j{}^T \boldsymbol{V}^* + \boldsymbol{X}_j \xi_j),$$

where we have used $y_j = \boldsymbol{X}_j^T \boldsymbol{V}^* + \xi_j$. According to definition, $E\xi_j = 0$.

Consequently,

$E\hat{\boldsymbol{V}}_1 = \boldsymbol{V}^*$, $\hat{\boldsymbol{V}}_1 - \boldsymbol{V}^* = \hat{\boldsymbol{V}}_{1*} = \left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1} \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{X}_i y_i$ and the covariance matrix of the weight vector $\hat{\boldsymbol{V}}_1$

$$\boldsymbol{\Sigma}_{V_1} = E\hat{\boldsymbol{V}}_{1*} \hat{\boldsymbol{V}}_{1*}{}^T = E\left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1} \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{X}_i y_i \; \frac{1}{N} \sum_{i=1}^{N} y_i \boldsymbol{X}_i{}^T \left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1} =$$

$$E\left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1} \frac{\sigma^2}{N^2} \sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^T \left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1} = \frac{\sigma^2}{N} E\left(\frac{1}{N} \sum_{j=1}^{N} \boldsymbol{X}_j \boldsymbol{X}_j{}^T \right)^{-1}.$$

The standard multivariate statistical analysis gives

$$E\left(\frac{1}{N}\sum_{j=1}^{N}\boldsymbol{X}_j\boldsymbol{X}_j{}^T\right)^{-1} = \frac{N}{N-n-1}\mathbf{I}\,. \tag{4.24}$$

Thus, the asymptotic (when $N \to \infty$) distribution of the final weight vector

$$\hat{\boldsymbol{V}}_1 \sim N_{V_1}(\boldsymbol{V}^*, \boldsymbol{\Sigma}_{V_1}) \text{ with } \boldsymbol{\Sigma}_{V_1} = \frac{\sigma^2}{N-n-1}\mathbf{I}. \tag{4.25}$$

Let $(\hat{\boldsymbol{V}}_1 - \boldsymbol{V}^*)^T(\hat{\boldsymbol{V}}_1 - \boldsymbol{V}^*) = d_1^2$, which is a squared error – a characteristic of the accuracy of determination of the final weight vector $\hat{\boldsymbol{V}}_1$. For randomly generated training data $(\boldsymbol{X}_1, y_1)$, $(\boldsymbol{X}_2, y_2)$, ..., $(\boldsymbol{X}_N, y_N)$ one can consider $\hat{\boldsymbol{V}}_1$ as a random vector. The mean squared error of $\hat{\boldsymbol{V}}_1$ is

$$mse(\hat{\boldsymbol{V}}_1) = E\,(\hat{\boldsymbol{V}}_1 - \boldsymbol{V}^*)^T(\hat{\boldsymbol{V}}_1 - \boldsymbol{V}^*) = \text{tr}\,\boldsymbol{\Sigma}_{V_1} = \frac{\sigma^2 n}{N-n-1}\,. \tag{4.26}$$

Let us start training with the initial weight vector $\boldsymbol{V}_0$. The covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{V}_1}$ is proportional to the identity matrix (Equation (4.25)). Therefore, without loss of generality we can assume that only the first component of the starting vector differs from the ideal vector $\boldsymbol{V}^*$, i.e. $\boldsymbol{V}_0 - \boldsymbol{V}^* = (D, 0, \ldots, 0)^T$. In order to analyse the accuracy of the linear move from $\boldsymbol{V}_0$ to $\hat{\boldsymbol{V}}_1$, let us introduce a *weighted estimate*

$$\hat{\boldsymbol{V}}_{new} = \alpha\,\hat{\boldsymbol{V}}_1 + (1-\alpha)\,\boldsymbol{V}_0, \tag{4.27}$$

where α is a weighting factor ($0 \le \alpha \le 1$).

Then the squared error of the new estimate, $\hat{\boldsymbol{V}}_{new}$, is

$$(\hat{\boldsymbol{V}}_{new} - \boldsymbol{V}^*)^T(\hat{\boldsymbol{V}}_{new} - \boldsymbol{V}^*) = (\alpha\,\hat{\boldsymbol{V}}_1 + (1-\alpha)\,\boldsymbol{V}_0 - \boldsymbol{V}^*)^T(\alpha\,\hat{\boldsymbol{V}}_1 + (1-\alpha)\,\boldsymbol{V}_0 - \boldsymbol{V}^*) =$$

$$(\alpha\,\hat{\boldsymbol{V}}_{1*} + (1-\alpha)\,(\boldsymbol{V}_0 - \boldsymbol{V}^*))^T(\alpha\,\hat{\boldsymbol{V}}_{1*} + (1-\alpha)\,(\boldsymbol{V}_0 - \boldsymbol{V}^*)) =$$

$$(\alpha\,\zeta_1 + (1-\alpha)\,D)^2 + a^2\sum_{j=2}^{n}\zeta_j^2 = (D + \alpha\,(\zeta_1 - D))^2 + a^2\Theta\,, \tag{4.28}$$

where $\zeta_1, \ldots, \zeta_n$ are components of the vector $\hat{\boldsymbol{V}}_{1*} = \hat{\boldsymbol{V}}_1 - \boldsymbol{V}^*$, and $\Theta = \sum_{j=2}^{n}\zeta_j^2$.

In order to find the optimal weighting factor, α_{opt} we equate a derivative of (4.28) to zero:

$$2(D + \alpha\,(\zeta_1 - D))\,(\zeta_1 - D) + 2\alpha\Theta = 0,$$

and obtain

$$\alpha_{opt} = \frac{D(D-\zeta_1)}{(D-\zeta_1)^2 + \Theta}. \tag{4.29}$$

By scaling the variables ζ_1 and Θ by the factor $\frac{N-n-1}{\sigma^2}$ we can represent α_{opt} as a function of two independent, Gaussian (ξ) and chi-square (χ^2_{n-1}), random variables

$$\alpha_{opt} \sim \frac{D(D-\xi\sigma/\sqrt{N-n-1})}{(D-\xi\sigma/\sqrt{N-n-1})^2 + \sigma^2/(N-n-1)\chi^2_{n-1}} \tag{4.30}$$

Thus, the optimal weighting factor α_{opt} depends on two random variables ξ and χ^2_{n-1}. Consequently, α_{opt} can be analysed as a random variable. Representation (4.30) allows us to derive the distribution density of α_{opt} and to calculate the mean and variance of α_{opt}. An expression for the mean is rather simple

$$E\alpha_{opt} = \frac{1}{1+\frac{\sigma^2 n}{(N-n-1)D^2}}. \tag{4.31}$$

Equation (4.31) shows that, in inaccurate initial weights case (large D), the final weight vector, $\hat{\boldsymbol{V}}_1$, ought to play more important role ($E\alpha_{opt}$ is close to 1). In SLP training it means: we need to train the perceptron almost until the end.

In order to compare the accuracy of the initial $\boldsymbol{V}_0$ and final $\hat{\boldsymbol{V}}_1$ weights, it would be desirable to utilise the same units. In (4.26), the accuracy of the weight vector $\hat{\boldsymbol{V}}_1$ was determined in terms of the training-set size. One can use (4.26) and express the accuracy of $\boldsymbol{V}_0$ in terms of an *imaginary training-set* size, N_0, that should give a certain squared error of $\boldsymbol{V}_0$:

$$(\boldsymbol{V}_0 - \boldsymbol{V}^*)^T(\boldsymbol{V}_0 - \boldsymbol{V}^*) = D^2 = \frac{\sigma^2 n}{N_0 - n - 1}. \tag{4.32}$$

Use of (4.32) in (4.31) results in

$$E\alpha_{opt} = \frac{N-n-1}{N_0+N-2n-2}. \tag{4.33}$$

From Equation (4.33), it is seen that with an increase in the accuracy of the determination of the initial weight vector (in this representation, with an increase in the imaginary sample size N_0), we need to scale the final $\hat{\boldsymbol{V}}_1$ weight with the smaller coefficient, α. In perceptron training, this means we have to stop training earlier. For N =30 and N_0 = 22, 26, 30, 40 we have following mean values of the weighting factor: α_{opt} = 0.90, 0.65, 0.50, 0.32.

Now we will evaluate the gain that can be obtained from optimal weighting the initial and final vectors. Inserting the mean value of the weighting factor (4.33) into the expression for the generalisation error of the weighted estimate (4.28), we obtain the mean generalisation error

$$mse(\hat{\boldsymbol{V}}_{new\&opt}) = \frac{\sigma^2 n}{N_0 + N - 2n - 1}. \tag{4.34}$$

Suppose for a moment that all N_0 vectors from the imaginary training-set can be utilised for training the perceptron. Then from Equation (4.26), we conclude that the generalisation error would be

$$mse(\hat{\boldsymbol{V}}_{opt}) = \frac{\sigma^2 n}{N_0 + N - n - 1}. \tag{4.35}$$

Comparison of (4.34) with (4.35) indicates that if instead of all N_0 training vectors we use only the weight vector $\boldsymbol{V}_0$, we are losing just n training vectors. If N_0 and N are large, ratio n/N_0 is small. Thus, the initial weight vector $\boldsymbol{V}_0$ can carry almost all information contained in the N_0 vectors of the imaginary training-set. To save this information we need to stop training in time.

Up to now, we considered only mean values. Figure 4.10 demonstrated that the overtraining phenomenon appears only in some of training experiments. Analysis of variances of optimal weighting factor α_{opt} (Equation 4.30) leads to the conclusion that in high-dimensional cases, overtraining almost always occurs. In Figure 4.11, we have four distribution density functions of α_{opt} found from the representation (4.30) by numerical methods for distinct n, N, and four initial weights of differing accuracy (N_0 = 22, 26, 40 and 212). From Figure 4.11, we see that for n = 20, N_0 = 22, and N = 30, the weighting factor $\alpha_{opt} > 1$ in 7.5% cases. This means that in 7.5% of cases, the overtraining will not be observed. A variance of the distribution of α_{opt} diminishes with simultaneous increase in the number of features, n, and the training-set size (N_0 and N_1). E.g. for n = 200, N_0 = 212 and N =300, we obtain the same value of $E\alpha_{opt}$ (0.90). In the latter case, however, we have a smaller variance and the overtraining effect will always be observed.

Large variances of the distribution of α_{opt} indicate that in small dimensional cases, in order to utilise all information contained in $\boldsymbol{V}_0$, we need to stop individually for each pair of $\boldsymbol{V}_0$ and $\hat{\boldsymbol{V}}_1$. In principle, in some cases, the individually optimal stopping allows us to obtain a generalisation error smaller than (4.35).

In order to stop training optimally, however, we ought to utilise certain *additional information*. The possibility of making use of the information contained in the initial weight vector is a peculiarity both of the prediction and the classification problems. In Chapter 3 it was demonstrated that, in the classification task, the analytical expressions for the generalisation error are much more complicated than Equation (4.26). Consequently, analytical expressions for α_{opt} should be even more complex than (4.31).

As a conclusion we claim that the initial weights can contain useful information. This information can be utilised in improving the generalisation performance of the network**.** To save this information we need to stop training in time.

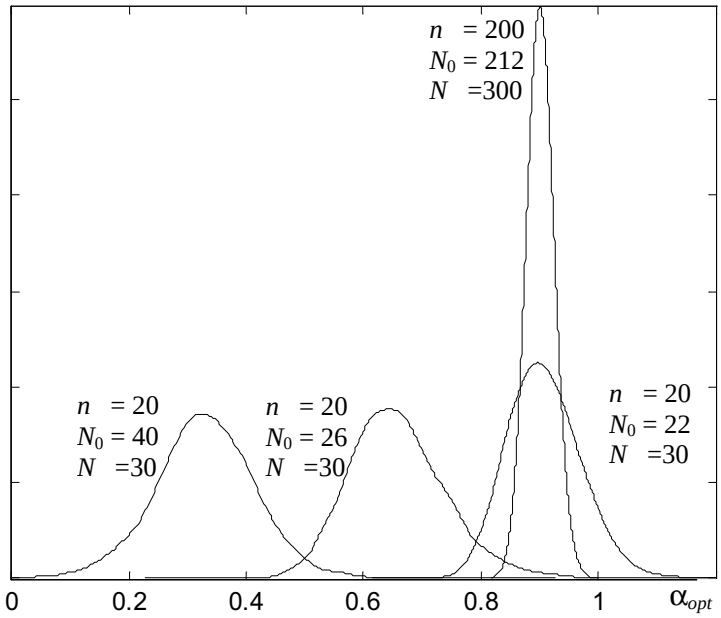

Fig. 4.11. Effect of the accuracy of the weights initialisation (the imaginary training-set size N_0) and dimensionality, n, on the distribution density function of the optimal weighting factor α_{opt}.

This peculiarity of the adaptive network training is one of the key items that help to integrate the statistical and neural net approaches.

4.6 Tools to Control Complexity

In SLP and MLP training, one can obtain many different classification rules. The overtraining and scissors effects advocate that in the finite training-set size situation, it is important to choose a classifier of proper complexity. In this book, we already have met several factors that affect the network's training result. The objective of this section is to enumerate and explain all factors that influence the training result and can be used for an intelligent control of the classifier's complexity. We categorise the factors associated with: (1) the cost function and (2) the optimisation technique used to find the minimum of the cost function.

The most popular cost function used in feedforward ANN design is the sum of squares defined in Section 4.1. Instead of the quadratic loss, one can use the

Kulback–Leibler distance (see e.g. Haykin, 1999). We concentrate here on the quadratic loss (4.2) only. The parameters that determine the cost (4.2) are:

- the character of the activation function $f(net)$;
- the target values $t_j^{(i)}$;
- the weights $\boldsymbol{V}$ themselves (most important are their magnitudes);
- the input patterns $\boldsymbol{X}_j^{(i)}$.

We consider the effects of each of these factors on peculiarities of the cost function and on the type of the classification rule obtained. Note, all these factors influence each other and their optimal values are mutually dependent.

The character of the activation function (a linear function, soft-limiting, the threshold function) introduces new possibilities into SLP training. In dependence on the target values and the magnitude of the weights, we have different types of association between the weighted sums $G_j^{(i)} = \boldsymbol{V}^T\boldsymbol{X}_j^{(i)} + v_0$ and the contribution terms $(t_j^{(i)} - f(G_j^{(i)}))^2$ in the sum $cost_t$. Small weights make the activation function linear, large weights together with the boundary values of the targets (+1 and -1 for tanh(G) activation function) affect the term $(t_j^{(i)} - f(G_j^{(i)}))^2$, moving it towards the threshold function and allowing it to minimise the empirical probability of misclassification.

4.6.1 The Number of Iterations

In non-linear SLP training, with an increase in the number of iterations one can obtain seven statistical classifiers of differing complexity (EDC, RDA, standard Fisher LDF, Pseudo-Fisher LDF, robust DA, minimum empirical error classifier, and the maximum margin (support vector) classifier). Without doubt, the number of iterations is the most important factor which determines the type of classification rule obtained in MLP training.

4.6.2 The Weight Decay Term

The addition of the "weight decay" term, $\lambda\boldsymbol{V}^T\boldsymbol{V}$, to the standard cost function reduces the magnitudes of the weights. Nowadays, it is the most popular technique utilised to control the complexity of the network. For very small weights, the activation function acts as a linear one. Thus, for the linear and non-linear SLP, addition of the regularisation term is equivalent to using a "ridge" estimate of a covariance matrix in a linear regularised discriminant analysis. Suppose we add the "weight decay" term $\lambda\boldsymbol{V}^T\boldsymbol{V}$, use the targets $t_1 = 1$ and $t_2 = -1$, and instead of the tanh activation function, use the linear one, i.e. $f(net) = net$. Then, equating the derivatives (4.4) and (4.5) to zero and solving the equations, we can show that, for the centred data ($\hat{\boldsymbol{M}} = 0$), $\boldsymbol{V} = (\mathrm{I}\lambda + \hat{\boldsymbol{\Sigma}})^{-1}\Delta\,\hat{\boldsymbol{M}}\,k_{\mathrm{WD}}$ and $v_{0(t)} = 0$ (k_{WD} is a scalar which does not depend on $\boldsymbol{X}$, the vector to be classified). Thus, the "weight decay" term leads to regularised discriminant analysis.

An alternative to the weight decay term is the term $+\lambda(\boldsymbol{V}^T\boldsymbol{V} - c^2)^2$. Here, a positive parameter c^2 controls the magnitude of the weights and acts as the traditional regulariser.

4.6.3 The Antiregularisation Technique

After starting from small or even zero values, the weights of the perceptron increase gradually. The non-linear character of the activation function assists in obtaining the complex classifiers (the robust DA, the robust regularized DA, the minimum empirical error and the maximum margin classifiers). The magnitudes of the weights, however, depend on the separability of the training sets (the empirical classification error). If the training sets of opposite pattern classes overlap, the weights cannot be increased very much. Consequently, small weights can prevent us from obtaining one of the complex classifiers.

When the pattern classes are highly intersecting, in order to be able to control the type of classifier obtained at the end, we can utilise the complementary term added to the traditional cost function. We can subtract the weight decay term instead of adding it. Then we obtain large weights and begin to minimise the empirical classification error. This technique is called *antiregularisation*. It was demonstrated that for non-Gaussian pattern classes characterised by a number of outliers, the antiregularisation can reduce the generalisation error radically. Often the traditional weight decay term destabilises the training process. Therefore, the term $+\lambda(\boldsymbol{V}^T\boldsymbol{V} - c^2)^2$, used to control the magnitudes of the output layer weights is preferred. In a modification of this technique, c^2 increases with t, the number of iterations.

Example 10. In Section 4.2.1 we used the antiregularisation term in order to obtain a non-linear decision boundary in the five-variate polynomial feature space (Figure 4.4). In Figure 4.12, we illustrate this technique with a two-variate example where we have Gaussian N($\boldsymbol{M}_i$, $\boldsymbol{\Sigma}$) data with highly correlated features. The data is contaminated by 10% noise N($\mathbf{0}$, $\mathbf{N}$):

$$\boldsymbol{M}_1 = -\boldsymbol{M}_2 = \begin{bmatrix} +0.030 \\ -0.023 \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} 0.040 & 0.0196 \\ 0.0196 & 0.01 \end{bmatrix}, \quad \mathbf{N} = \begin{bmatrix} 1.0 & -0.7 \\ -0.7 & 1.0 \end{bmatrix}.$$

For training, 100 bi-variate vectors from each class were used (Figure 4.12*a*). For testing, we used 1000+1000 vectors. We'd like to draw attention to the fact that, in bi-variate data **A2**, we have a comparatively large number of training vectors ($N_1 = N_2 = 100$), and the signal and noise components have opposite correlations. The variance of noise is much larger than that of the signal. In traditional training (conditions E1 – E4, $\eta = 0.5$), we did not have any success (decision boundary 1 in Figure 4.12*a* and learning curve 1 in Figure 4.12*b*). Adding the antiregularisation term $0.02 \times (\boldsymbol{V}^T\boldsymbol{V} - 25)^2$ to the cost function does not influence the learning curve at the beginning but, later on, when the weights increase substantially, the decision boundary begins approaching the optimal boundary (2 in Figure 4.12*a*) with 12% generalisation error (see curve 2 in Figure 4.12*b*).

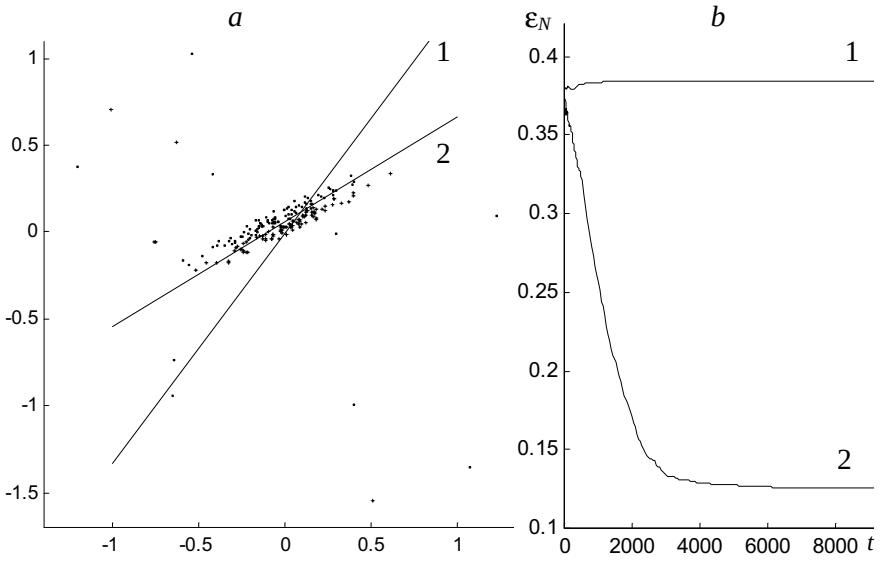

Fig. 4.12. Effect of the antiregularisation technique in SLP training: 1 – standard BP training, 2 – with antiregularisation.

4.6.4 Noise Injection

Another popular method for stabilising solutions while training artificial neural net classifiers is noise injection. Noise can be added to inputs of the network, to outputs, to the targets and to the weights of the network.

4.6.4.1 Noise Injection to Inputs

In this approach, we add a zero mean and small covariance noise vectors $\mathbf{n}_\alpha$ to the training vector $\boldsymbol{X}_j^{(i)}$ at each training iteration:

$$\boldsymbol{X}_{j\beta}^{(i)} = \boldsymbol{X}_j^{(i)} + \mathbf{n}_{j\beta}^{(i)}, \quad (\beta = 1, 2, ..., t_{max}), \tag{4.36}$$

where t_{max} is the number of training sweeps (presentations of all vectors in the training-set).

This technique is also called jittering of the data. While injecting noise $m = t_{max}$ times to each training vector we artificially increase the number of training observations m times.

For the linear classification, we can find a link between noise injection and regularised disriminant analysis. Let the mean of noise vector $\mathbf{n}_{j\beta}^{(i)}$ be zero and the

covariance matrix be $\lambda \times \mathbf{I}$. Then the covariance matrix of random vectors $\boldsymbol{X}_{j\beta}^{(i)}$ (β =1, 2, …, m) with noise added will be $\boldsymbol{\Sigma}_i + \lambda \times \mathbf{I}$, where $\boldsymbol{\Sigma}_i$ is the true covariance matrix of $\boldsymbol{X}_j^{(i)}$. Consider now the sample covariance matrix of the jittered training-set composed of mN_i vectors $\boldsymbol{X}_{j\beta}^{(i)}$ (j=1, 2, …, N_i β = 1, 2, ..., m). When $m \to \infty$, the new sample estimate of the covariance matrix tends to

$$\hat{\boldsymbol{\Sigma}}_N^{(i)} = \frac{1}{N_i - 1} \sum_{j=1}^{N_i} (\boldsymbol{X}_j^{(i)} - \hat{\boldsymbol{M}}_j^{(i)})(\boldsymbol{X}_j^{(i)} - \hat{\boldsymbol{M}}_j^{(i)})^T + \lambda \times \mathbf{I}. \tag{4.37}$$

The covariance matrix (4.37) coincides with the ridge (regularised) estimate (2.37) used in regularised discriminant analysis. In Section 4.1, it was shown that the two-class linear classifier derived by the mean squared error criterion (4.2) and the weight decay regularisation term using an equal number of sample patterns from each class ($N_2 = N_1$) is equivalent to RDA. Therefore when $m \to \infty$, the noise injection technique also approximates the regularisation approach using the weight decay.

In the limit when $m \to \infty$, jittering the training data (4.36) with spherical Gaussian noise is similar to the nonparametric Parzen window probability density estimate (2.48) with $\boldsymbol{D}$=**I**.

Example 11. In Figure 4.13*a* we present a histogram obtained from seven one-dimensional randomly-selected data points, where to each of the seven observations we added m = 80 independent Gaussian N(0, 0.05) random numbers. 100 bins were used to draw the histogram. In Figure 4.13*b* we present the histogram where we added m = 20,000 random numbers. The histogram with m = 20,000 is similar to the Parzen window density estimate obtained with the Gaussian kernel and a smoothing parameter λ = 0.05 (Figure 4.13*c*).

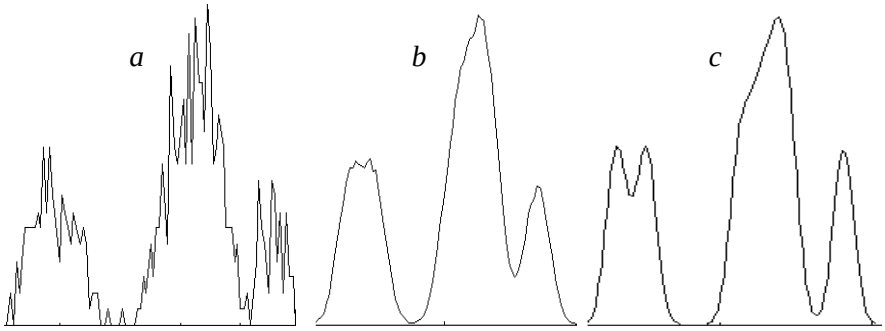

Fig. 4.13. Histograms obtained where, to each of seven one-dimensional observations, *a* m = 80 or *b* m = 20000 independent Gaussian N(0, 0.05) random numbers have been added; *c* the Parzen window density estimate.

One needs a great number of noise injections, *m*, in order to approach the Parzen window density estimate. In the multivariate case, the number of noise injections has to be much higher. Therefore, in very high-dimensional cases, the spherical noise injection can become ineffective.

> Adding a noise essentially amounts to introducing new information which states that the space between the training vectors is not empty.

It is very important in MLP design, since the network (more precisely the cost function to be minimised) does not know that the pattern space between the training observation vectors is not empty. A noise injected to inputs introduces this information. In practice, it is important to choose the correct value of λ, the noise variance. Similarly to regularised discriminant analysis and smoothing in the Parzen window classifiers, an optimal value of λ (the regularisation parameter or the noise variance) depends upon the number of training vectors and the complexity of the problem.

4.6.4.2 Noise Injection into the Weights and into the Outputs of the Network

Adding noise to the weights is a common practice to tackle local extrema problems in optimisation. In addition to fighting the local minima problems, noise injection into the weights increases the overlap of the training sets. Consequently, noise injection reduces the magnitudes of the weights and acts as a tool to control the complexity of the classification rule. In the MLP classifier design, noise injection into the weights of the hidden layer acts as noise injection into the inputs of the following layer and acts as the regulariser of the output layer neurons.

During the training process, noise can be also added to the outputs of the networks (or the targets). Such noise injection also reduces the magnitudes of the weights and influences the complexity of the classification rule. In all cases, an optimal value of a noise variance should be selected. Typically in situations with unknown data, the variance is determined by the cross validation technique.

A drawback of training with noise injection is that it usually requires one to use a small learning rate constant and many sample presentations, *m*, over the noise. In high-dimensional space, this problem is particularly visible.

4.6.4.3 "Coloured" Noise Injection into Inputs

As a rule, one injects into network inputs a "white" noise – spherical uniform or Gaussian N(0, $\mathbf{I}\lambda$) random vectors. In high-dimensional, real-world pattern classification problems, data is situated in non-linear subspaces of much lower dimensionality than the formal number of the features. This means that for large λ spherical noise injection can distort data configuration. In *k-NN* directed noise injection, *k-NN* clustering should be used to determine the local intrinsic data dimension; and the added noise should be limited to this intrinsic dimension. Then we have minor data distortion and, in comparison with spherical noise injection, we obtain a gain. One can add noise to the training vector $\boldsymbol{X}_j^{(i)}$ in the directions of *k* of its nearest neighbours.

An alternative way is to use k nearest neighbours to calculate a singular sample covariance matrix, $\hat{\Sigma}_{ij}^{k}$, around $\boldsymbol{X}_{j}^{(i)}$, and then to add Gaussian N($\mathbf{0}$, $\lambda \hat{\Sigma}_{ij}^{k}$) noise. Thus, instead of white noise we are adding "coloured" noise. The optimal number of nearest neighbours k, as well as a scale parameter, λ, should be determined in an experimental way.

4.6.5 Control of Target Values

One more factor which always affects the magnitudes of the weights is the *target values*. Let us analyse the classification problem with two pattern classes using SLP trained by the sum of squares cost function (4.2) and tanh activation function (1.8).

Suppose targets t_1 and t_2 are close to each other, e.g. t_1 = 0.5 and t_2= -0.5. Then *after* minimisation of the cost function, we obtain small values of the sums $G_{ij} = \sum_{\alpha=1}^{n} v_\alpha x_{\alpha j}^{(i)} + v_0$ and, consequently, small weights. Thus, the weighted sum G_{ij} varies in a small interval around zero. Essentially, the activation function, $o(G)$, is acting as a linear function. Hence, with proximal targets we will not obtain the robust classification rule, the minimum empirical error and maximal margin classifiers.

When $t_1 \to 1$ and $t_2 \to$ -1, the weights will increase. Then the trained SLP classifier will begin to ignore training vectors distant from the decision hyperplane $\sum_{\alpha=1}^{n} v_\alpha x_{\alpha j}^{(i)} + v_0 = 0$. Thus, the target values can be utilised as a tool to control the "robustness" of the classification rule. To obtain the minimum empirical error and the maximal margin classifier, we need to use target values that are very close to the boundary activation function values (1 and -1 for activation function (1.8)). *Values outside the interval* (-1, 1) assist in fast growth of the magnitudes of the weights and speed up the training process. However, they also increase the probability of being trapped at a local minimum (the local trench).

4.6.6 The Learning Step

The learning step, η, affects the speed of the iterative training process. A small learning step slows down the training and forces the weights to remain small for a long time. Thus, while training the non-linear SLP, a small learning step assists in obtaining a more complete sequence of the regularised classifiers and the standard linear discriminant function with the conventional or with the pseudo-inverse covariance matrix.

A large learning step speeds up the growth of the weights. However, too large learning step can stop the training process after just the first iteration. When we use a very large learning step, then after the first iteration we can obtain enormous weights. Consequently, the gradient of the cost function can become very small and the training algorithm can stop. If we fulfil the conditions E1 – E4,

enumerated in Section 4.1, we get EDC after the first iteration and do not move further towards other classification rules (see e.g. curve 3 in Figure 4.3). Large and intermediate learning step values can stop the training process soon after RDA is obtained and do not allow one to obtain RDA with a small regularisation parameter. A large learning step leads immediately to large weights. Thus, it increases the possibility of being trapped at local minimum.

Furthermore, if a constant learning step is utilised, then with an increase in the magnitude of the weights, the training process slows down and stops. One can say that the *effective value* of η decreases. In this case, the fixed value of the learning step can prevent the algorithm from getting to the minimum empirical error or the maximum margin classifier. One solution to overcome this difficulty is to use a *variable learning step* η, where after a certain number of iterations η is either increased or reduced. In back propagation training, in order to obtain the maximal margin classifier, it was suggested that the learning step η should increase exponentially, e.g. $\eta = \eta_0 (1+\varepsilon)^t$, where t is the iteration number, and ε is a small positive constant chosen by trial and error. Typically, one can use larger ε values when one has small or even no error while classifying the training-set. A larger learning step can prevent us from obtaining the standard statistical linear classifiers. It can allow us to obtain the robust regularised discriminant function. The degree of regularisation and the robustness can be controlled by the learning step value. Therefore, one should choose ε value carefully.

Example 12. In Figure 4.14, we present four pairs of learning curves: the generalisation error as function of the number of iterations for four η values.

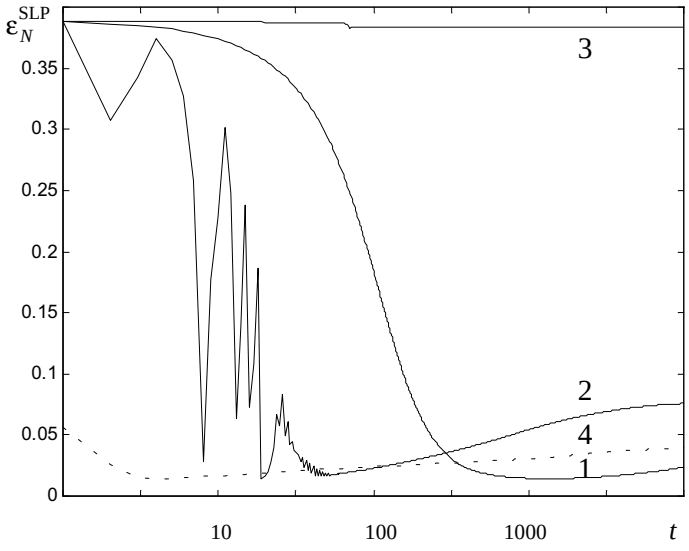

Fig. 4.14. Generalisation error as a function of the number of iterations t for different values of the learning step η: 1 – η = 0.001, 2 – η = 0.1, 3 – η = 10 (targets 0.9 & 0.1) ; 4 – $\eta = 1.0 \times 1.005^t$ (after whitening data transformation; targets 1 & 0).

Here we used two 20-variate GCCM pattern classes with correlated features (ρ = 0.9), the Mahalanobis distance δ = 4.65 (the Bayes error ε_B = 0.01), targets t_1 = 0.9, t_2 = 0.1, (for graphs 1, 2, 3) and one training-set composed of 30 + 30 vectors. The straight line 3 shows that for η = 10, the adaptation process after the first few iterations practically stops. We have very slow (at the beginning) training when the learning step is too small (curve 1 for η = 0.001). A whitening data transformation (see Section 5.4.3) together with the exponential increase in the learning step $\eta = 1.005^t$, results in a fast training and the smallest generalisation error, the generalisation error 1.4% (curve 4).

Thus, the value of the learning step is an extremely important factor that affects the result of the training process: the minimal generalisation error and the optimal number of iterations.

4.6.7 Optimal Values of the Training Parameters

In linear single layer perceptron training, the regularisation techniques based on weight decay, spherical noise injection, and early stopping can be equivalent to classical linear regularised discriminant analysis. In all four complexity control methods, we need to choose optimal values of λ or the optimal number of iterations. It was shown theoretically that in RDA, the optimal value of the regularisation parameter λ depends on N, the training-set size: λ_{opt} decreases when N increases (see Section 3.4.9). Bellow we present an illustration of equivalence of these complexity control techniques.

Example 13. In Figure 4.15, we present the generalisation error versus λ, the conventional regularisation parameter in RDA (*a*), the regularisation parameter in the weight decay procedure (*b*), and the noise variance in the noise injection regularisation (*c*). In this experiment, we used artificial eight-variate GCCM data with strongly dependent features. It is important to note that the optimal values of the complexity control parameters under discussion significantly depend on the particular randomly chosen training-set. Therefore, the experiments were repeated 20 times with different randomly chosen training sets. In Figure 4.15, we have averaged the values.

For each value of N, the number of training examples, all three families of the graphs approximately exhibit the same generalisation error at minima points and explain, by example, the equivalence of all three regularisation techniques. This experiment confirms the theoretical conclusion: the optimal values of the regularisation parameters decrease when the number of training examples increase.

In SLP training, the number of iterations, the learning step, the targets, the noise, the weight decay term, and the non-linearity of the activation function are all acting together. The influence of one regularisation factor is diminished by the other ones. This circumstance causes additional difficulties in determining the optimal values of the complexity control parameters.

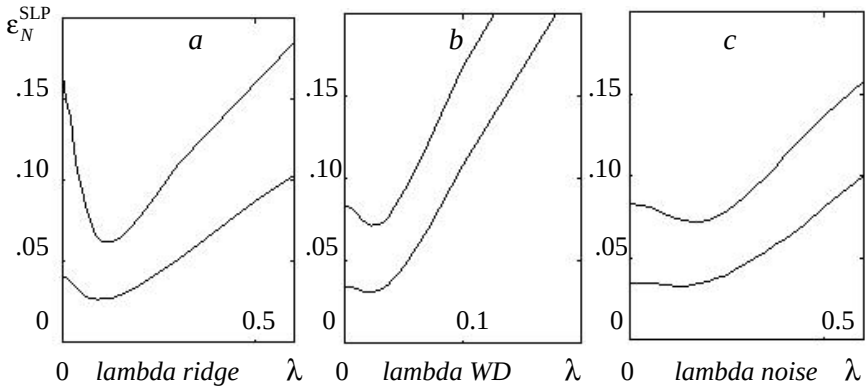

Fig. 4.15. Three types of regularisation of the SLP classifiers for two sizes of the learning sets ($N_1 = N_2 = 5$ and 10): *a* – the regularised discriminant analysis with the ridge estimate of the covariance matrix, *b* - weight decay, *c* – noise injection to inputs.

The targets, the learning step (to some extent even the number of iterations) do not act in a clearly visible way. Thus, they can be considered as *automatic regularisers*. In neural network training, however, their effects cannot be ignored.

Considerations presented in this section are valid for the gradient type search techniques used to find the weights of the perceptrons. Utilisation of more sophisticated optimisation techniques such as a second order Newton method or its modifications can speed up the training process. However, at the same time it introduces definite shortcomings. For example, in the case of the linear pereceptron, the second order (Newton) method, in principle, allows us to minimise the cost function in just one step, however, it prevents us from obtaining the regularised discriminant analysis. In this sense, the back propagation algorithm is more desirable. Nevertheless, this shortcoming of the Newton method can be diminished by introducing the regularisation term.

4.6.8 Learning Step in the Hidden Layer of MLP

In comparison with SLP, in MLP training several additional aspects arise. The learning step is the first important *secret factor* that affects the complexity. In the gradient descent optimisation procedure, a fixed value of the learning step η causes a random fluctuation of the parameters to be optimised (in our case, the components of the weight vector). Asymptotically, for fixed η, and large *t*, the number of iterations, the weight vector $\boldsymbol{V}_{(t)}$ oscillates around $\hat{\boldsymbol{V}}_N$, an optimal weight vector for this particular training-set. In his classic paper, Amari (1967) has shown that, asymptotically, for large *t*, the weight $\boldsymbol{V}_{(t)}$ is random Gaussian $N(\hat{\boldsymbol{V}}_N, \eta\boldsymbol{\Sigma}_{\boldsymbol{V_t}})$, where $\boldsymbol{\Sigma}_{\boldsymbol{V_t}}$ is a covariance matrix that depends on the data, a configuration of the network and peculiarities of the cost function near the optimum. In optimisation, in order to find the minimum exactly, the parameter η should converge to zero with a certain speed.

Random fluctuation of the hidden units' weight vector $\boldsymbol{V}_{(t)}$ suggests that the outputs of the hidden layer neurones are random variables too. Thus, we have a *chaos* (a process noise) inside the feedforward neural network. Random outputs serve as random inputs to the single-layer perceptrons of the output layer. Consequently, in the output layer, we have a type of data jittering. The noise variance depends on the value of the learning step η_h used to adapt the hidden units. Note, a noise injected into the inputs of the output layer is coloured.

Hence, in MLP training, the learning step η_h in the hidden layer, a traditional noise injection, the weight decay term, as well as the number of iterations are producing similar effects. At the same time, we need to remember that if the hidden units' weights become large, the non-linear character of the activation function flattens the minima of the cost function, reduces the gradient and diminishes the variance of the noise injected to the output layer. Hence, we have an automatic reduction of the process noise variance.

Example 14. The effect of regularisation of the multilayer perceptron performed by three different, but at the same time similar, techniques: weight decay, jittering of the training data by a spherical Gaussian noise and jittering the inputs of the output layer controlled by η_h, the learning step of the hidden neurones (*eta hidden*), is illustrated in Figure 4.16. The MLP with eight inputs, eight hidden neurones and one output was used. In this experiment we used the artificial non-Gaussian data, **SF8**, already discussed in Section 4.4.2.1. The training-set size was N = 140 (70 observation vectors from each class) and the test set Nt = 1000 = 500 + 500. Average values of the generalisation error estimated from seven independent randomly generated training sets are presented in the figure.

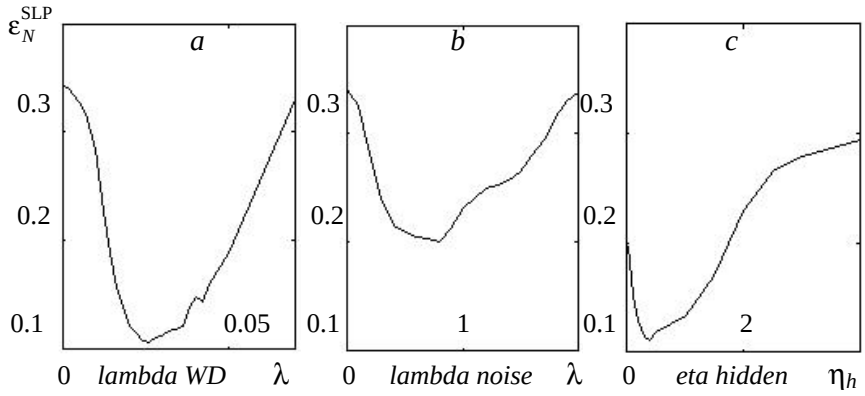

Fig. 4.16. Three types of regularisation of the multilayer perceptron: *a* – weight decay, *b* – uniform noise injection into the inputs, *c* – learning step of the hidden layer neurones.

This experiment confirms the theoretical conclusion that the learning step η_h used to train the hidden layer neurones has a similar effect to that of the weight decay and noise injection. Thus, the parameter η_h can be used to control the classifier's complexity. At the same time, the learning step is an obligatory parameter in BP

training. Hence, the learning step η_h acts in an inexplicit way and affects value of other regularisation tools like the weight decay parameter, the variance of noise, the target values, etc.

4.6.9 Sigmoid Scaling

The SLP classifier performs classification of an unknown vector, $\boldsymbol{X}$, according to a sign of the discriminant function $g(\boldsymbol{X}) = v_1 x_1, v_2 x_2, ..., v_n x_n + v_0$. Multiplication of all weights, $v_1, v_2, ..., v_n, v_0$, by a positive scalar α does not change the decision boundary. In MLP classifiers, however, the effect of the multiplication of the hidden layer weights has more important consequences.

A proportional increase in the magnitude of the hidden units' weights creates sharp angles in the edges of the decision boundary. A proportional decrease in the magnitudes of all hidden layer weights smoothes the sharp angles of the decision boundary and changes the complexity of the MLP classifier. Thus, the control of the magnitudes of the hidden layer weights is one of the possible techniques that could be utilised to determine the classifier's complexity. We have illustrated this phenomenon by the example already discussed in Section 1.2 (Figure 1.5).

Example 15. In Figure 4.17*a* we have 15 + 15 bi-variate training vectors from two artificially generated pattern classes where vectors from the first class are inside the second one, and there is a ring-shaped margin between the pattern classes.

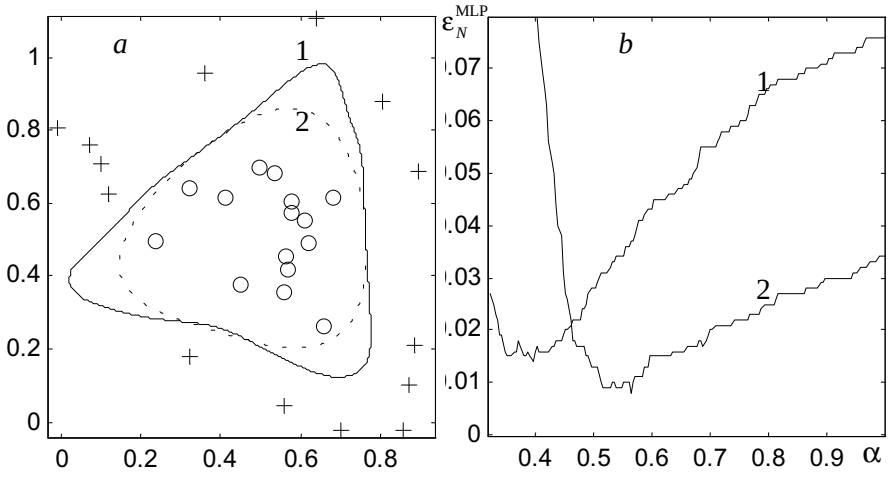

Fig. 4.17. *a* The training vectors from two pattern classes and the decision boundaries: 1– decision boundary of overtrained MLP (after 10,000 training sweeps), 2 – the smoothed decision boundary of the optimally stopped MLP; *b* The generalisation error as a function of the scaling parameter α: 1 – scaling of the weights of the overtrained perceptron), 2 – scaling of the optimally stopped perceptron.

After training MLP with 10 hidden units for 10,000 iterations in batch-mode, we obtained the non-linear decision boundary 1 with generalisation error, 7.6% (for

estimation, we used 500 + 500 test vectors). In back-propagation training, we observed a significant overtraining: optimal early stopping resulted in a generalisation error two times smaller (3.4%).

The proportional reduction of all hidden layer weights by a factor $\alpha = 0.35$ smoothed the decision boundary and reduced the generalisation error to 1.5% (see curve 1 in Figure 4.17*b*). In this experiment, a notably better smooth decision boundary (curve 2 in Figure 4.17*a*) was obtained after smoothing the decision boundary of the optimally stopped MLP – the generalisation error was reduced from 3.4% to 0.09% (curve 2 in Figure 4.17*b* with a minimum at $\alpha_{opt} = 0.53$). Thus, in order to reduce the generalisation error we had to optimise both the number of iterations and the weights scaling parameter α.

As a possible generalisation of the scaling method, the magnitudes of the weights can be controlled individually for each hidden unit. An alternative to the scaling of the hidden layer weights by the factor α is to add the regularisation term, $\lambda(\boldsymbol{V}_{hidden}^{T}\boldsymbol{V}_{hidden} - h^2)^2$, to the cost function. Here, the parameter h controls the magnitudes of the hidden layer weights $\boldsymbol{V}_{hidden}$. A success of the sigmoid scaling technique is caused of the fact that, in MLP training, the large weights of the output layer reduce magnitudes of the error transferred back to adapt the hidden units' weights.

4.7 The Co-Operation of the Neural Networks

One of the principal methods to control the complexity of the ANN is to choose a proper architecture of the network. The architecture and the complexity of the network should be chosen in accordance with the training-set size: for short learning sequences, one should use simple structured networks and for long sequences, one can utilise complex ones.

There are three ways to change the network architecture. One is to *prune* the network, i.e. sequential rejection of the features, the weights or the separate neurones of the network. Another possibility is *incremental learning* where one designs a simple network at first and then gradually adds new neurones, one after the another. The third way is *evolutionary training*, where the features, weights or neurones are added and removed sequentially. These three approaches represent the backward, forward and mixed feature selection procedures to be discussed in Chapter 6.

One more popular approach to selecting the architecture is to design several simple networks ("local experts") and then to join their individual solutions. To achieve this decision-making strategy, one has to design a final, "boss" (also known as "combining" or "gating" rule) decision-making algorithm.

4.7.1 The Boss Decision Rule

For complex data configurations, it often happens that in spite of a large number of perceptron training experiments with various architectures, initialisations and training algorithm parameters, the researcher does not succeed in minimising the

cost function satisfactorily and thus does not obtain sufficiently good classification performance. Typically, in spite of the fact that each time the training procedures stop at different points of the multivariate weights space, the minimal cost function values are close to each other. The weights obtained in different initialisations of the network, however, are different.

Often, it becomes visible that one preceptron performs well in one *region of the multivariate feature space*, while another one is a good expert in another region. Some researchers suppose that the local minima and the flat multidimensional areas of the ANN cost function are "windmills" that are impossible to fight, and suggest, instead, designing a complex well-tuned cooperative network in the high-dimensional feature space. Thus, in spite of fighting the "windmills" one can try to make use of them.

An alternative to the direct building of complex networks is to make several "small" networks and then to join their solutions. The small networks are called local experts, local classifiers, or local neural networks, and the approach of synthesising the complex ANN from simple ones is called a co-operation of neural networks, ensemble of local experts, modular neural networks, etc. We show that the statistical methods described in this book can help to solve the network synthesis problem in a reasonable way.

A starting point of the co-operation approach is an assumption that different members of the co-operative network – the local experts – perform well in different regions. Then the boss – the output network – should be created and use solutions of all the local experts. On the basis of this information, the boss ought to perform the final decision.

Voting. It is the simplest technique used to make the decision. The voting can perform well if the decisions of the experts are statistically independent and all experts are equally qualified. More often, separate experts make very similar or identical decisions. Therefore, their decisions are statistically dependent and one needs more sophisticated methods to make the final classification. Moreover, different experts have diverse qualifications. Then, instead of the ordinary voting, a *weighted voting* can be utilised. The number of methods for designing the linear classifier can be put to use in determining the weights for each expert.

Nonparametric classification methods. In more complex situations, we can apply the local decision rules discussed in Chapter 2. Suppose we have r local experts, and let the j-th expert perform classification of an unknown vector $\boldsymbol{X}$ into one of m_j classes. We suppose various experts can be authorities in classifying into the different number of classes. This is a generalisation of a standard situation where $m_j = L$, the number of the classes. The total number of possible solutions (bins, cells) made by the r experts is $m = \prod_{j=1}^{r} m_j$. Denote a decision made by the j-th expert as o_j and that made by a set of experts by the vector $\boldsymbol{O} = (o_1, o_2, \ldots, o_r)^T$. Each vector $\boldsymbol{O}$ can assume only one bin, b_s, from the m bins: $b_1, b_2, \ldots, b_{m-1}, b_m$.

To design the final, boss classifier, we assume $\boldsymbol{O} = (o_1, o_2, \dots, o_r)^T$ to be discrete valued random vector. Then the conditional distribution of the i–th class vector $\boldsymbol{O}$ taking one bin from m is characterised by m probabilities

$$P_1^{(i)}, P_2^{(i)}, ..., P_{m-1}^{(i)}, P_m^{(i)}, \quad \text{with} \qquad \sum_{j=1}^{m} P_j^{(i)} = 1, \quad (i = 1, 2, \dots, L).$$

Let P_i be the prior probability of the pattern class ω_i. Then the optimal Bayes rule should make the final classification according to decisions of r local experts:

to allocate vector $\boldsymbol{O}$, falling into the s-th bin, according to the maximum of products

$$P_1 P_S^{(1)}, \quad P_2 P_S^{(2)}, ..., \quad P_L P_S^{(L)}.$$

This is the *multinomial classifier* for categorical data discussed in Section 2.9.1. To design this classifier, we have to know the probabilities $P_1^{(1)}, P_2^{(1)}, \dots, P_m^{(L)}$. The maximum likelihood and Bayes approaches to estimate these probabilities from the training data can be used. E.g. in the maximum likelihood approach, we use $\hat{P}_j^{(i)} = n_{ij} / N_i$, where n_{ij} are a number of training examples from the i-th class with j-th bin.

Example 16. We demonstrate the statistical decision functions approach to designing the boss rule by considering a problem of classifying the lung-sounds data by three local experts. Each local expert uses a different part of the data (early expiration, middle expiration or late expiration) and performs classification into one of three pattern classes. Thus, $m_j = 3$, and $m = 3 \times 3 \times 3 = 27$. To make the decision, each local expert (MLP with six inputs, seven hidden units and three outputs) used six cepstral coefficients calculated from a certain segment of the acoustic lung signal. In the first class, we have 180 vectors, in the second class 190 vectors and in the third one 200 vectors.

The decision-making scheme of the boss classifier is illustrated by Table 4.4. In rows 2, 3, and 4, we have all possible answers of the three experts. In rows 5, 6, and 7 (# of ω_1, # of ω_2, and # of ω_3), we present n_{ij}, the numbers of training examples from each of three classes within each of 27 bins.

In row 8 (Multinom), we have decisions of the Bayes (the multinomial classifier) that performs judgements according to the maximum of n_{1j}, n_{2j}, and n_{3j}. In the row 9, we present the decisions of the voting algorithm. In dubious cases, we give preference to the decision of the first expert (or the second one). Decisions made by voting and by the multinomial classifier differ in 11 bins (printed in bold) out of the 27. For this data set, the empirical error rates, nevertheless, are similar: 25.6% for the multinomial classifier and 30.5% for the voting rule.

Table 4.4. Classification by three local experts.

N.of bin	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Expert1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
Expert2	1	1	1	2	2	2	3	3	3	1	1	1	2	2	2	3	3	3	1	1	1	2	2	2	3	3	3
Expert3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
# of ω_1	76	5	8	14	3	0	3	2	3	7	1	0	7	8	3	1	0	2	7	3	1	5	3	2	5	2	9
# of ω_2	9	7	5	7	16	1	2	1	2	3	3	3	10	66	9	2	1	3	1	1	5	4	13	7	1	4	4
# of ω_3	4	4	0	5	1	1	8	7	11	7	1	0	2	0	1	3	3	15	0	1	1	0	2	6	8	9	100
Multinom	1	2	1	1	2	2	3	3	3	1	2	2	2	2	3	3	3	3	1	1	2	1	2	2	3	3	3
Voting	1	**1**	1	1	2	**1**	**1**	**1**	3	**2**	2	2	2	2	2	**2**	**2**	3	1	**3**	**3**	**3**	2	**3**	3	3	3

4.7.2 Small Sample Problems and Regularisation

When the number of the local experts and the number of the classes are large, the number of bins *m* becomes enormous. Hence, utilisation of the multinomial classifier becomes impractical. In such cases, one ought to simplify the decision rule. For this purpose, the decision-tree, the nonparametric Parzen window, or the *k-NN* classifiers discussed in Chapter 2 can be put to use. Recall that in the small training-set case, simpler classification rules can perform better than the complex ones. Consequently, one needs to select the complexity of the boss rule properly. This entails selecting the optimal smoothing parameters of the PW classifier, the optimal number of neighbours, the appropriate structure and the complexity of the decision tree.

Strictly speaking, the training data used to design separate experts cannot be used to estimate the unknown parameters of the boss decision rule. Usage of the training-set to estimate parameters of the boss rule leads to biased estimates of these parameters and worsens the decision-making algorithm (in such cases, each expert network boasts itself and deceives the boss). In principle, a *separate independent validation data* set should be used. Often it is difficult to have an independent validation set. However, in spite of this warning, researchers often ignore the bias problem and decide to use the training-set data once more.

In order to reduce the bias, one can perform regularisation of the boss decision making algorithm. One of possible ways to regularise the boss classifier is through *noise injection*. In the above medical diagnosis problem with the lung data, we have jittered the six cepstral coefficients by adding zero-mean and variance λ noise. In this way we increased the number of vectors 20 times. Later we classified this semi-artificial validation set by means of three local experts and estimated the probabilities $P_1^{(1)}$, $P_2^{(1)}$, ..., $P_m^{(L)}$. Note, the jittered signal was not used to train the expert networks. It was used only to estimate the probabilities $P_1^{(1)}$, $P_2^{(1)}$, ..., $P_m^{(L)}$. This way of injecting noise has led to coloured noise injection into the outputs of the local experts. In the previous Section, we advocated that jittering of the data

carries supplementary information that while recording the data, the measurements were performed with some error.

The noise variance, λ, plays the role of the regularisation parameter. As in other regularisation problems, the curve generalisation error versus the regularisation parameter λ has a minimum. It is important to choose the proper value of λ. This value should be selected in accordance with the training-set size. In the experiment considered, after proper selection of the noise variance, we have succeeded in reducing the generalisation error substantially.

If as the boss rule, we use the decision tree or the Parzen window with special kernels, the complexity of the decision rules should be controlled by the number of the decision tree's final leaves or the smoothing parameter of the PW classifier. Techniques to select the classifier of optimal complexity will be discussed in Chapter 6.

4.8 Bibliographical and Historical Remarks

The mathematical model of the neurone was first described in McCulloch and Pitts (1943) and the supervised learning algorithm of the perceptron was suggested in Rosenblat (1958). Widrow and Hoff (1960) introduced the least mean square cost function, the training algorithm and the linear adaptive element called an adaline. Cybenko (1989) and Hornik et al. (1989) demonstrated that the multiplayer perceptrons are universal approximators. Koford and Groner (1966) were first to show that minimisation of the sum of squares cost in the linear SLP leads to the standard linear Fisher DF. Raudys (1995*ab*, 1996, 1998*b*) demonstrated that in BP training, six other statistical classifiers can be obtained. Broomhead and Lowe (1988) first introduced the RBF concept into ANN design and the idea of learning vector quantisation belongs to Kohonen (1986). The analysis of the effect of eigenvalues on training speed discussed in the Section 4.3.3 belongs to Le Cun (1987).

Section 4.4 on relations between the training-set size and generalisation of SLP is from Raudys (1998*c*) and Raudys (1995*a*). Comparison of the standard adaline training algorithm with two other modifications (relaxation and the fixed increment algorithms) was carried out by Smith (1972). He concluded that these two algorithms are more sensitive to the training-set size. The overtraining effect was noticed and described by a number of authors (see e.g. Hertz, Krogh and Palmer (1991) and Bishop (1995) and references therein). Its explanation by extra two factors is new: 1) the initial weight values of the perceptron and 2) the constant changing of the cost function in the BP training process. Equation (4.33) for α_{opt} was derived in Raudys and Amari (1998). In the analysis of the initialisation accuracy, a more general result than (4.27) with α_{opt} determined by (4.33) is known. Let $\hat{\boldsymbol{V}}_0$ and $\hat{\boldsymbol{V}}_1$ be estimators derived from these two sets. Then

$$\hat{\boldsymbol{V}}_{opt} = \mathbf{G}^{-1} (N_0 \mathbf{G}_0 \hat{\boldsymbol{V}}_0 + N_1 \mathbf{G}_1 \hat{\boldsymbol{V}}_1), \tag{4.38}$$

where $\mathbf{G}_0 = \mathbf{G}(\hat{\boldsymbol{V}}_0)$, $\mathbf{G}_1 = \mathbf{G}(\hat{\boldsymbol{V}}_1)$ and $\mathbf{G} = N_0\mathbf{G}_0 + N_1\mathbf{G}_1$.

For large training-set sizes N_0, N_1, and fixed dimensionality n, we may put the matrix $\mathbf{G}_0 = \mathbf{G}$, and we have (4.33). This is the first-order asymptotic theory, and we can show the second-order result by using the ancillary statistics (Amari, 1987).

Tools to control the perceptron complexity are reviewed according to Raudys (2000*d*). It has been known for a long time that a weight decay and noise injection can improve generalisation (Plaut *et al*. 1986; Hinton, 1989). However, antiregularisation technique has been proposed only recently (Raudys, 1995*b*, 1998*b*). The coloured noise injection has been suggested by Duin (1993) and considered later by Skurichina, Raudys and Duin (2000). More details and bibliographical references on the means to control complexity can be found in Reed (1993), Reed *et al*. (1995), Bishop (1995), An (1996), Mao and Jain (1993), Jain, Mao and Mohiuddin (1996). Effective capacity study by simulation reported in Section 4.4.2.3 belongs to Kraaijeveld and Duin (1994) and remarks on a different effect of accuracy of the hidden and output layer weights to an increase in generalisation error to Raudys (1994, 1995*a*). Considerations on co-operation of neural networks as well as suggestion to regularise it in small sample situations are from Güler *et al*. (1996). More information about the co-operation of neural networks, multi-modular ANN, mixture of local experts and committee machines can be found in Mazurov (1971), Rastrigin and Erenstein (1981), Fogelman-Soulie, Vinnet and Lamy (1993), Chapter 7 of Haykin (1999) and Tax *et al*. (2000) and references therein.