Chapter 2

Learning with neural networks

Functions can be approximated by feed-forward neural networks. The accuracy of such an approximation depends on the number of hidden units (which can be seen as the basis functions that are used to approximate the function) and the algorithm used for adjusting the weights in the neural network, based on the available learning samples. For both the number of hidden units and the values of the weights exist optimal values for each particular problem in order to reach some desired accuracy in the approximation. In this chapter we will discuss a method for finding the optimal number of hidden units for a feed-forward neural network, and compare various parameter estimation methods which determine the weights in the network such as to minimise an error function.

Depending on the structure of the neural network, for each problem and network architecture there exists a minimal **representation error**: this is the error that is obtained with optimal weight values for that particular network and approximated function. Finding optimal weight values means minimisation of an error function which expresses the error in approximation by the neural network of a set of learning samples $(\mathbf{x}, \mathbf{y})$. This will lead to a low **optimisation error**. However, it is important that the approximation represent the underlying function well, rather than the separate learning samples: a low **generalisation error** is what we are actually looking for. The generalisation error is measured by testing the approximation over a separate set of test samples which are obtained from the same process as the learning samples. When the optimisation and generalisation errors lie far apart, **overfitting** is usually the case: the neural network learns to represent the learning samples very well, but does not represent the underlying process from which the samples are generated.

It is generally assumed that the noise present in the learning samples is Gaussian. From this assumption the error function is taken to be the L_2 norm of $\|\mathbf{y} - \mathcal{N}(\mathbf{x})\|$ summed over all learning (or test) samples. This error function is known as the **summed squared error** or **SSQ**. Several methods exist which estimate parameters to optimise this (or another) error function; this is a problem that is studied by numerical analysis (Stoer & Bulirsch, 1980). In this case, we are interested which is the best method for the type of optimisation problem at hand. Traditionally, feed-forward type neural networks are trained by an algorithm known as **error back-propagation** which is a simplified variant

(*) Part of this text was previously published in *Neural Networks* (van der Smagt, 1994a).

of the steepest descent method and was, after the initial discovery by Paul Werbos in 1974 (Werbos, 1974), rediscovered by various researchers in 1985 (Cun, 1985; Parker, 1985; Rumelhart, Hinton, & Williams, 1986). With this algorithm, the gradient of the error function with respect to the parameters (i.e., the weights) is computed and greedily minimised. This method is very, very slow and inefficient. It is a well-known fact that methods which do not only use the local gradient but also the Hessian (matrix of second derivatives) of the error function tend to be much faster. Recently, such methods have also been applied to feed-forward network learning (Battiti, 1992; Kinsella, 1992; Barnard, 1992). In this chapter, we will discuss and compare various methods using second order information of the error function.

Finding a minimal set of hidden units to solve a particular problem with a certain accuracy is important in order to minimise the computing cost. A smaller number of hidden units leads to fewer parameters (i.e., weights) in the system, and hence the parameter estimation method will be cheaper. Furthermore, there will be a smaller chance that the neural network is *overfitting* the data, i.e., that it represents the learning samples with great accuracy, but not the underlying problem. Few results are known. Barron (Barron, 1991) discusses a theoretical upper bound for the approximation of a feed-forward neural network depending on the number of hidden units and learning samples that it is trained with. Using a different method, Vyšniauskas *et al.* (Vyšniauskas, Groen, & Kröse, 1993) find an estimate of the required number of hidden units and learning samples for an averaged desired approximation error. We will discuss the latter method in section 2.2.

When applied to a control problem such as hand-eye coordination, learning samples are continuously generated. The error function that has to be minimised, however, is evaluated with respect to *all known* learning samples; an evaluation (and hence error minimisation) for each new sample separately is not feasible for most methods discussed. Instead, learning samples must be collected and the SSQ is minimised over this set of learning samples. Minimisation of the error function over a set of learning samples is known as **batch learning**. Adaptation of the network weights for each sample separately, which is possible for error back-propagation when it minimises the SSQ only for one sample at a time (i.e., the squared error instead of *summed* squared error), is known as **incremental learning**.

Approximation of an unknown function

We consider a function $\mathcal{F} : \Re^m \to \Re^n$ that is to be approximated. The function $\mathcal{F}$ is not analytically known, but samples $S^N = \{s^1, s^2, \ldots, s^N\}$ with $s^p = (\mathbf{x}^p, \mathbf{y}^p)$ are generated by a process which is described by $\mathcal{F}$, i.e., $\mathcal{F}(\mathbf{x}^p) = \mathbf{y}^p$. From the available samples we want to build a 'smooth' approximation of $\mathcal{F}$.

In order to approximate $\mathcal{F}$ we regard a feed-forward neural network $\mathcal{N}_S^w : \Re^m \to \Re^n$. $\mathcal{N}_S^w$ consists of *basis functions* $\phi_j : \Re \to \Re$, $(1 \le j \le \kappa)$ which are represented by hidden units in the network, and weighted connections between the hidden and output units. The network is parametrised by the weights $\boldsymbol{w} = (w_{1ij}, w_{2kl})$. The w_{1ij} are the parameters of the basis functions, where $1 \le i \le \kappa$ and $1 \le j \le m$. The connections between the hidden and output units have associated weights w_{2lk}, with $1 \le l \le n$ and $1 \le k \le \kappa$. The total number of parameters in $\boldsymbol{w}$ is indicated by ν. The ith output element of $\mathcal{N}$ is calculated by

$$\mathcal{N}_S^w(\mathbf{x})_i = \sum_{j=1}^{\kappa} w_{2ij}\phi_j\left(\boldsymbol{w}_{1j}, \mathbf{x}\right) + w_{2,i,\kappa+1}, \qquad 1 \le i \le n, \tag{2.1}$$

with one layer of hidden units and linear output units. The structure of the network is depicted in figure 2.1.

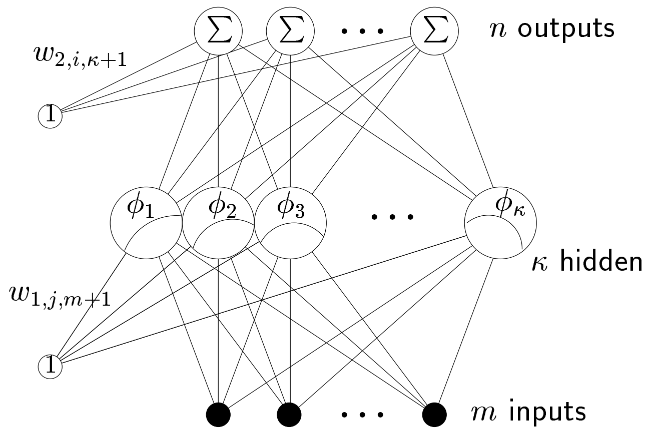

Figure 2.1: The structure of a feed-forward neural network.

Without loss of generality we assume $\mathcal{N}$ to have a fixed m and n. The approximation $\mathcal{N}_S^w$ then depends on the number of hidden units κ and learning samples S, as well as the learning algorithm which determines the parameters $\boldsymbol{w}$ from S and the architecture of $\mathcal{N}$.

After discussing which basis functions are best used in section 2.1, two matters will be investigated. First, section 2.2 discusses a method to determine the optimal number of learning samples N and hidden units κ. Then, section 2.3 reviews various parameter estimation methods to determine optimal parameters $\boldsymbol{w}$. In section 2.4 the reviewed methods are compared for two standard problems. A discussion is given in section 2.5.

2.1 Choosing the basis functions

Choosing a good set of basis functions ϕ_i reduces the burden of finding optimal values of parameters $\boldsymbol{w}_i$. In particular, when the SSQ error function is quadratic in the parameters $\boldsymbol{w}$, the optimal solution can be analytically found[(*)].

Such quadratic dependency can be obtained by intelligently choosing the basis functions. For instance, when the approximated function is known to be a polynomial, an obvious choice is to use basis functions $\{1, x, x^2, \ldots\}$ (although orthogonal polynomials are preferred; see appendix A). The complexity of the parameter estimation method is dependent of the interference of the basis functions used in the approximation.

[(*)] Roundoff errors might require that the optimal solution be computed twice or thrice, instead of only once which would theoretically suffice to find the $\boldsymbol{w}^0$.

2.1.1 Types of functions

In order to determine this interference, we distinguish the following properties that basis functions have:

1. parametrised or non-parametrised;

2. local or global;

3. orthogonal or non-orthogonal;

and discuss those properties in turn.

Parametrised. With non-parametrised basis functions we mean that the input of these basis functions is independent of the parameters in the network, i.e., the network computes the following function:

$$\mathcal{N}_S^w(\mathbf{x})_i = \sum_{j=1}^{\kappa} w_{2ij}\phi_j(\mathbf{x}), \qquad 1 \le i \le n \tag{2.2}$$

(for notational simplicity we ignore the constant term $w_{2i,\kappa+1}$). This can be regarded as a feed-forward network where the w_{1ij} have fixed values. An example of such a method is the Fourier transform, where the basis functions are cosine and sine and the w_{1ij} are fixed at 1, 2, etc.; Taylor expansions also fall into this category. Only the w_{2ij} have to be estimated. Since the w_{1ij} are fixed, the neural network becomes linear in its parameters $\mathbf{w}$; the optimal parameters can be determined by minimisation of the error functions

$$E_i = \sum_{(x,y)\in S} \left\| y_i - \sum_{j=1}^{\kappa} w_{2ij}\phi_j(\mathbf{x}) \right\|, \qquad 1 \le i \le n, \tag{2.3}$$

in parameters w_{2ij}. When $N > \kappa$, (2.3) can be solved by minimising all of its terms independent of the others. Thus a minimisation of (2.3) can be analytically found.

The disadvantage of this approach is that, in general, the basis functions have to be tailored to each particular problem. More flexible approaches are only possible in the approximation of one-dimensional functions, or two dimensions when the samples are arranged on a grid (e.g., visual data).

Locality. Secondly, global basis functions are more interfering than local ones. The most local basis function is a delta function:

$$\phi(\mathbf{w}_{2j}, \mathbf{x}) = \delta(\mathbf{w}_{2j} - \mathbf{x}) := \begin{cases} 1 & \text{if } \mathbf{w}_{2j} = \mathbf{x}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

In effect, a table lookup method is implemented: each learning sample can be simply stored in a table such that learning is a trivial task. However, generalisation is a problem that is not solved by this method: we know only the values at the stored points; interpolation has to be added separately. A simple method to predict the output of a sample whose

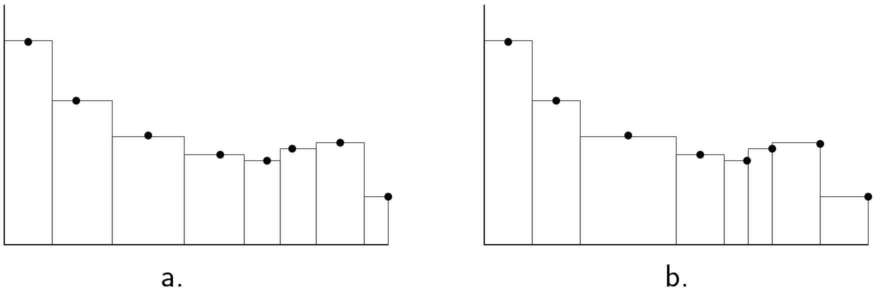

Figure 2.2: Approximating a smooth function with delta-functions.
In a., for 'interpolation' the value of the nearest neighbouring sample (marked by black dots) is chosen. In b., each sample is accommodated by repeatedly splitting the input space in half.

input value is not stored is to use the value of its nearest neighbour, or to split the input space into fixed disjoint kernels (see figure 2.2). For neural implementations of this and similar techniques see, e.g., (Miller III, 1989; van der Smagt, Jansen, & Groen, 1992a).

Other types of local basis functions used in neural network approximators are radial basis functions (Moody & Darken, 1988).

Orthogonality. A final method of creating non-interfering basis functions is by making them orthogonal. Orthogonality of two functions $\phi_i(\mathbf{x})$ and $\phi_j(\mathbf{x})$ is defined as

$$\int_{x_a}^{x_b} \phi_i(\mathbf{x})\phi_j(\mathbf{x})\,d\mathbf{x} = c\delta_{ij} \tag{2.5}$$

over some interval $[\mathbf{x}_a, \mathbf{x}_b]$, where δ_{ij} is the Kronecker delta which is 1 if and only if $i = j$, and c is a constant. When $c = 1$, the functions are called *orthonormal*.

Since we have finite set of samples S, the orthonormality of (2.5) is interpreted only in terms of S, i.e., it is required that

$$\sum_{(x,y)\in S} \phi_i(\mathbf{x})\phi_j(\mathbf{x}) = \delta_{ij}. \tag{2.6}$$

The advantage of using orthogonal basis functions is that basis functions can be added or removed one at a time without having to update the parameters of the previously placed basis functions. Thus the approximation can be incrementally improved at low computational cost (see appendix A).

An example of orthogonal basis functions has already been given: the delta functions and Fourier transform described above. Otherwise, typical orthogonal functions are wavelets or orthogonal polynomials (which can be obtained by use of a three-term recursive generator (Stoer & Bulirsch, 1980); Forsythe (Forsythe, 1957) describes an elegant implementation of this generator. With this method, the orthogonal polynomials are constructed incrementally (i.e., updated for each new sample), such that the system can accommodate new learning samples in $O(\kappa)$ time.) Practice shows, however, that polynomial basis functions are rather *stiff* (i.e., not flexible) for fitting data; especially when polynomials of high order are used the fitted curve tends to oscillate.

For functions $\mathcal{F}$ of more than one variable, constructing orthonormal interpolating basis functions becomes cumbersome and even prohibitive when the data samples are not arranged on a grid. For problems of high dimensionality, orthonormal basis functions are impractical in use.

Which basis functions do we choose?

In our approach we require optimal flexibility and applicability. Furthermore, a good generalisation is advantageous. Therefore, the basis functions that we choose are non-orthonormal, global, parametrised basis functions which are quantitatively independent of the available data samples. In this case, both the w_{1ij} and w_{2ij} must be estimated. Again, (2.1) is the representation of the neural approximator, and the error functions that must be minimised are

$$E_i = \sum_{(x,y)\in S} \Big\| y_i - \sum_{j=1}^{\kappa} w_{2ij}\phi_j\Big(\sum_{k=1}^{m} w_{1jk}x_k + w_{1,j,m+1}\Big) + w_{2,i,\kappa+1}\Big\|, \qquad 1 \le i \le n. \tag{2.7}$$

Since no model of $\mathcal{F}$ is assumed, the basis functions are chosen on basis of their 'flexibility' and cost. A common choice is the sigmoid function $\sigma(\cdot)$.

2.2 The minimal approximation error

In this section we discuss an error model for function approximation with feed-forward neural network presented in (Vyšniauskas, Groen, & Kröse, 1992; Vyšniauskas *et al.*, 1993).

When approximating $\mathcal{F}$ with $\mathcal{N}$, there will remain an error in the approximation which we will call the *approximation error* and indicate by $E_a(\mathbf{w})$. We define E_a as

$$E_a(\mathbf{w}) = \int_x \|\mathcal{F}(\mathbf{x}) - \mathcal{N}_S^w(\mathbf{x})\|\, d\mathbf{x}. \tag{2.8}$$

The approximation error can be separated into three types of error: representation error, generalisation error, and optimisation error.

Representation error. Let us first consider the case when the full set of learning samples S^∞ is available. Also, assume that, given S^∞, we can find the optimal $\mathbf{w}^0$. On a digital computer this can theoretically be obtained via a brute-force search through the whole (finite but very large) parameter space. This will render the optimal network $\mathcal{N}_\infty^{w^0}$, such that

$$E_r(\mathbf{w}^0) = \int_x \left\|\mathcal{F}(\mathbf{x}) - \mathcal{N}_\infty^{w^0}(\mathbf{x})\right\| d\mathbf{x} \tag{2.9}$$

is minimal and only depends on the architecture (c.q., number of hidden units κ) of $\mathcal{N}$. $E_r(\mathbf{w}^0)$ is called the *representation error*. This error goes to zero as the number of hidden units goes to infinity (Hornik *et al.*, 1989):

$$\lim_{\kappa\to\infty} E_r(\mathbf{w}^0) = 0. \tag{2.10}$$

Generalisation error. In real-world applications, only a finite (i.e., small) number of learning samples is available or can be used at the same time. The values of $\mathcal{F}$ for which no samples are available must be interpolated. Thus a generalisation error occurs, consisting of the errors for those pairs $(\mathbf{x}, \mathbf{y}) \notin S$. The generalisation error is defined as:

$$E_g(\mathbf{w}^0|S^N) = \int_x \left\| \mathcal{N}_\infty^{w^0}(\mathbf{x}) - \mathcal{N}_{S^N}^{w^0}(\mathbf{x}) \right\| d\mathbf{x} \tag{2.11}$$

where $\mathcal{N}_S^w$ is short for $\mathcal{N}_{S^N}^{w|S^N}$, and the notation $\mathbf{w}^0|S^N$ indicates the optimal weight matrix for the given samples in S^N. This error goes to zero as the size of the learning set increases:

$$\lim_{N\to\infty} E_g(\mathbf{w}^0|S^N) = 0. \tag{2.12}$$

Optimisation error. Finally, the optimisation error is defined as

$$E_o(\mathbf{w}|S) = \int_x \left\| \mathcal{N}_S^{w^0}(\mathbf{x}) - \mathcal{N}_S^{w}(\mathbf{x}) \right\| d\mathbf{x} \tag{2.13}$$

which is the error obtained due to the fact that the optimisation procedure is not exact but heuristic. We can substitute this in (2.8) to obtain

$$\begin{aligned}
E_a(\mathbf{w}) &= \int_x \left\| \mathcal{F}(\mathbf{x}) - \mathcal{N}_S^{w^0}(\mathbf{x}) \right\| d\mathbf{x} \\
&= \int_x \left\| \mathcal{F}(\mathbf{x}) - \mathcal{N}_\infty^{w^0}(\mathbf{x}) + \mathcal{N}_\infty^{w^0}(\mathbf{x}) - \mathcal{N}_S^{w^0}(\mathbf{x}) + \mathcal{N}_S^{w^0}(\mathbf{x}) - \mathcal{N}_S^{w}(\mathbf{x}) \right\| d\mathbf{x} \\
&\le \int_x \left\| \mathcal{F}(\mathbf{x}) - \mathcal{N}_\infty^{w^0}(\mathbf{x}) \right\| + \left\| \mathcal{N}_\infty^{w^0}(\mathbf{x}) - \mathcal{N}_S^{w^0}(\mathbf{x}) \right\| + \left\| \mathcal{N}_S^{w^0}(\mathbf{x}) - \mathcal{N}_S^{w}(\mathbf{x}) \right\| d\mathbf{x} \\
&\le \int_x \left\| \mathcal{F}(\mathbf{x}) - \mathcal{N}_\infty^{w^0}(\mathbf{x}) \right\| d\mathbf{x} + \int_x \left\| \mathcal{N}_\infty^{w^0}(\mathbf{x}) - \mathcal{N}_S^{w^0}(\mathbf{x}) \right\| d\mathbf{x} \\
&\quad + \int_x \left\| \mathcal{N}_S^{w^0}(\mathbf{x}) - \mathcal{N}_S^{w}(\mathbf{x}) \right\| d\mathbf{x} \\
&= E_r(\mathbf{w}^0) + E_g(\mathbf{w}^0|S) + E_o(\mathbf{w}|S).
\end{aligned}$$

Thus we conclude

$$E_a(\mathbf{w}|S) \le E_r(\mathbf{w}^0) + E_g(\mathbf{w}^0|S) + E_o(\mathbf{w}|S). \tag{2.14}$$

We can calculate the average optimisation, approximation, and generalisation error as a function of κ and N:

$$\begin{aligned}
\epsilon_a(N,\kappa) &= \langle E_a(\mathbf{w}|S^N) \rangle_{i,S^N}, \\
\epsilon_r(\kappa) &= \langle E_r(\mathbf{w}^0|S^N) \rangle_{i,S^N}, \\
\epsilon_g(N,\kappa) &= \langle E_g(\mathbf{w}^0|S^N) \rangle_{i,S^N}, \\
\epsilon_o(N,\kappa) &= \langle E_o(\mathbf{w}|S^N) \rangle_{i,S^N},
\end{aligned}$$

where $\langle\cdot\rangle_{i,S^N}$ means an average over random initial network weights and sample set S of size N. The weights $\mathbf{w}$ are those obtained after optimisation with the chosen learning algorithm.

From (2.14) it is derived that

$$\epsilon_a \le \epsilon_g + \epsilon_r + \epsilon_o. \tag{2.15}$$

2.2.1 Model of the approximation error

From the following properties:

$$\begin{aligned}
\epsilon_a(\infty, \kappa) &= \epsilon_r(\kappa), \\
\epsilon_a(N, \infty) &= \epsilon_g(N, \infty), \\
\lim_{\kappa \to \infty} \epsilon_a(\infty, \kappa) &= 0, \\
\epsilon_a(0, \kappa) &= \infty, \\
\epsilon_a(N, 0) &= \infty
\end{aligned}$$

a model for the ϵ_a must be established. This model can then be used to determine the optimal κ and N in order to obtain a desired approximation error ϵ^0. Optimality is defined in terms of computational complexity of the learning algorithm. The computational complexity r can be estimated as (Vyšniauskas *et al.*, 1993)

$$r \sim N\kappa^k \tag{2.16}$$

where k is the order of the learning algorithm; $k = 1$ for the conjugate gradient algorithm (discussed below) or error back-propagation; $k = 2$ for Hessian-based algorithms (see below). The solution is found by simultaneous satisfaction of minimum computational complexity and reaching the desired approximation error:

$$\frac{\partial r}{\partial \kappa} - \frac{\partial r}{\partial N} \frac{\partial \epsilon_a / \partial \kappa}{\partial \epsilon_a / \partial N} = 0, \tag{2.17a}$$

$$\epsilon_a(N, \kappa) = \epsilon^0. \tag{2.17b}$$

The intersection of the dashed and solid curves shown in figure 2.3 present the desired

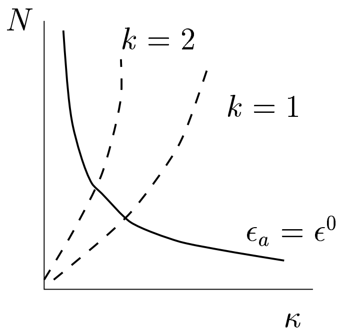

Figure 2.3: The solution of minimal resources of computation is determined by the intersection of the isoline $\epsilon_a = \epsilon^0$ (solid line) and the curve of minimal resources (dashed lines). (From (Vyšniauskas *et al.*, 1993). Reprinted with permission.)

solution.

We are interested in the asymptotic behaviour of $\epsilon_a(N, \kappa)$ for $\kappa \gg 1$ and $N \gg 1$. According to Vyšniauskas *et al.*, ϵ_a can be approximated by

$$\epsilon_a^2(N, \kappa) = \sum_p \sum_{i+j=p} \gamma_{ij} (N^{-1} - N_*^{-1})^i (\kappa^{-1} - \kappa_*^{-1})^j \approx \sum_p \sum_{i+j=p} \frac{\gamma_{ij}}{N^i \kappa^j} \tag{2.18}$$

where $i, j, p \in \aleph$ and γ_{ij} are the parameters we are looking for. These parameters can be found by minimising the difference between the model of the error (2.18) and the measured error in approximation given by the feed-forward neural network, as a function of κ and N. This latter error can be found by evaluating the feed-forward network error over a number of test samples which are not used as learning samples.

This method will be used to determine the optimal network size in chapter 4. In particular, the parameters are found by comparing (2.18) with E_a for various values of N and κ. Note that $\mathcal{F}$ is not a known function, but instead only learning samples from $\mathcal{F}$ are available. Therefore the integral (2.8) is replaced by the sum

$$\sum_{(x,y)\in S} \|\mathbf{y} - \mathcal{N}_S^w(\mathbf{x})\| .$$

Barron (Barron, 1991) uses a different approach to attack the same problem, resulting in a different approximation of $\epsilon_a(N, \kappa)$. For the problems that are tackled in this thesis, the results of Barron's and Vyšniauskas' methods are very close for large N (Vyšniauskas *et al.*, 1993); we chose to follow the latter approach.

2.3 Parameter estimation

Having determined a set of basis functions $\phi_i(\cdot)$ and the optimal number of learning samples and hidden units, an optimal set of parameters $\mathbf{w}$ has to be found so as to minimise the summed squared error

$$E(\mathbf{w}) = \sum_{i=1}^{n} \sum_{(x,y)\in S} \frac{1}{2} \left[y_i - \sum_{j=1}^{\kappa} w_{2ij}\phi_j \left(\sum_{k=1}^{m} w_{1jk}x_k + w_{1,j,m+1} \right) + w_{2,i,\kappa+1} \right]^2 . \tag{2.19}$$

This learning process can be depicted as a walk through a very high-dimensional weight space (see figure 2.4). Even though an optimal weight vector (point A) exists for a given network structure, the limited learning set will render a different optimal solution (point B) within the set W^0 of optimal solutions. Due to imperfect minimisation a sub-optimal solution (point C) is reached. Note that, due to the possibility of permutations in the weights and hidden units (Chen & Hecht-Nielsen, 1989), the solutions for $\mathbf{w}^0$ are not unique, but the weight space consists of a number of identical cones, each containing a global optimum; this may help in finding an optimal solution.

Iterative minimisation methods are based on the following principle: given are a function $E(\mathbf{w})$ which is to be minimised, and an initial state $\mathbf{w}_0$. Now for each iteration, find a minimisation *direction* $\mathbf{u}_i$ and a *stepsize* α_i, and update $\mathbf{w}$ as

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i \mathbf{u}_i. \tag{2.20}$$

The task of minimisation is finding optimal values for α_i and $\mathbf{u}_i$ when only local information of the function is available. Repeated application of (2.20), each time with an 'optimal' $\mathbf{u}_i$ and α_i, will bring E to a minimum. This minimum can be either local or global; there is no guarantee whatsoever about what kind of minimum is encountered. If

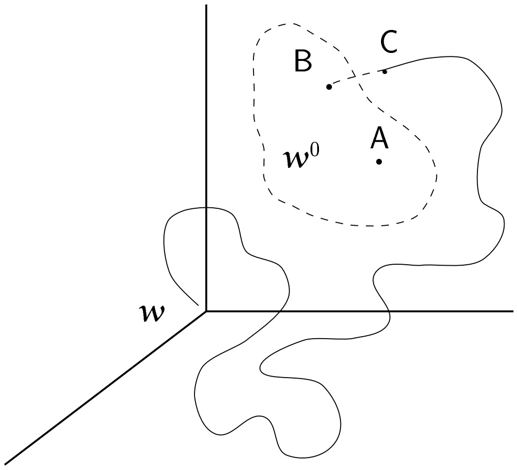

Figure 2.4: Learning of a feed-forward neural network as a trajectory in weight space. The figure shows an fictive three-dimensional weight space. Point A represents the best approximation w^0, point B a single realisation of the optimal solution for a given learning set, and point C is the actual solution. The subset of optimal solutions is given by the dashed line.

the error landscape is 'complicated,' i.e., has many local minima, a better minimum may be sought by starting at another initial point, or perturbing the system away from the just found minimum in hope for a better one (Press, Flannery, Teukolsky, & Vetterling, 1986; Aarts & Korst, 1989).

Line minimisation. When the minimisation direction is available, the problem is to decide *how far* to go along this direction before a new direction is chosen. Usually, evaluations of the function and its derivatives are used to locate some minimum, global or local, and stop there. There are methods available (e.g., (Bromberg & Chang, 1992)) which locate the global minimum, but these methods require several tens to hundreds of function evaluations and are therefore too expensive. Also, it can be argued that perfect line minimisation may lead to getting stuck in local minima of the overall error landscape.

One often used method, presented in (Press *et al.*, 1986), is modelled after Brent: Given there are three values $x_a < x_b < x_c$ such that the function at x_b is the lowest (i.e., a minimum is bracketed by x_a and x_c). The sign of the derivative at x_b indicates whether a minimum is located in $[x_a, x_b]$ or in $[x_b, x_c]$. This section is then linearly interpolated from its endpoints, and the procedure is repeated. This method is used in the simulations presented below.

2.3.1 First order methods

The learning error (2.19) can be written as a Taylor expansion around $\boldsymbol{w}_0$,

$$E(\boldsymbol{w}) = E(\boldsymbol{w}_0) + \sum_{l,i,j} \left. \frac{\partial E}{\partial w_{lij}} \right|_{w_0} w_{lij}$$

$$+\frac{1}{2}\sum_{l,i,j,l',i',j'} \left.\frac{\partial^2 E}{\partial w_{lij}\partial w_{l'i'j'}}\right|_{w_0} w_{lij}w_{l'i'j'} + \cdots. \tag{2.21}$$

In first-order methods, all but the first two terms of this Taylor expansion are ignored. These methods, where the local gradient alone determines the minimisation direction $\mathbf{u}$, are known as *steepest descent* or *gradient descent*, in feed-forward neural network training known as *error back-propagation*.

Steepest descent works as follows. When the system is in a 'state' $\boldsymbol{w}_i$, the gradient $\mathbf{g}_i = \partial E/\partial \boldsymbol{w}_i$ is computed and a minimisation step in the direction $\mathbf{u}_i = -\mathbf{g}_i$ is performed using the weight update rule (2.20). In normal steepest descent minimisation, the stepsize α is obtained by a line minimisation in the direction of $\mathbf{u}_i$. The value of α is thus computed that the gradient $\mathbf{g}_{i+1}$ at $\boldsymbol{w}_{i+1}$ is perpendicular to $\mathbf{u}_i$, in order to be minimal along $\mathbf{u}_i$.

In standard back-propagation, the line minimisation is usually replaced by a fixed step size α. A reason for this is cost: one function evaluation for a feed-forward network means forward propagation of all the learning patterns. Therefore, performing a line minimisation, which usually requires at least three to five function evaluations, is costly and may not improve the algorithm (see section 2.4).

The back-propagation search direction is usually augmented with a *momentum term* (Rumelhart *et al.*, 1986):

$$\mathbf{u}_{i+1} = -\mathbf{g}_{i+1} + \beta_i \mathbf{u}_i. \tag{2.22}$$

This extra term is generally interpreted as to avoiding oscillations. It will be shown below that adding the momentum term is wise when the values α_i and β_i are well-chosen; one method which chooses these parameters is known as *conjugate gradient*.

The inefficiency of steepest descent is due to the fact that the minimisation directions and step sizes are chosen poorly: unless the first step is chosen such that it leads directly to the minimum, steepest descent will zig-zag with many small steps.

2.3.2 Second order methods

When using not only the first but also the second term from the Taylor expansion (2.21), the error (2.19) can be written as

$$E(\boldsymbol{w}) \approx \tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}_0) - \mathbf{b}^T\boldsymbol{w} + \tfrac{1}{2}\boldsymbol{w}^T B\boldsymbol{w} \tag{2.23}$$

where $\boldsymbol{w}$ is expressed relative to $\boldsymbol{w}_0$ and

$$\mathbf{b} \equiv -\left.\nabla E\right|_{w_0}, \quad \text{and} \quad [B]_{ij} = \left.\frac{\partial^2 E}{\partial w_i \partial w_j}\right|_{w_0}. \tag{2.24}$$

B, the matrix of second derivatives, is called the *Hessian* of E at $\boldsymbol{w}_0$. The gradient of $\tilde{E}$ at $\boldsymbol{w}_i$ can be found to be

$$\mathbf{g}_i = \nabla\tilde{E}(\boldsymbol{w}_i) = B\boldsymbol{w}_i - \mathbf{b}. \tag{2.25}$$

Minima are located where the $\mathbf{g}_i$ is 0, i.e.,

$$B\boldsymbol{w}_i - \mathbf{b} = 0. \tag{2.26}$$

The weight vector $\mathbf{w}_i$ for which this holds is called *optimal*, for which we write $\mathbf{w}^0$, such that

$$\mathbf{w}^0 = B^{-1}\mathbf{b}. \tag{2.27}$$

Thus, knowledge of the Hessian and gradient of E reduce the minimisation to matrix inversion, if $E - \tilde{E}$ is small. However, amongst other problems (Battiti, 1992) calculation of B is computationally prohibitive. For reasons of practical feasibility we will therefore have to revert to approximating methods.

Quasi-Newton methods.

The aim of quasi-Newton methods, such as the BFGS ('Broyden-Fletcher-Goldfarb-Shanno') and DFP ('Davidon-Fletcher-Powell') methods is to iteratively compute matrices M_i such that

$$\lim_{i\to\infty} M_i = B^{-1}. \tag{2.28}$$

The term *quasi-Newton* applies if

$$M_{i+1}(\mathbf{g}_{i+1} - \mathbf{g}_i) = \mathbf{w}_{i+1} - \mathbf{w}_i \tag{2.29}$$

is satisfied (this is trivially true for $M = B^{-1}$, since $\mathbf{g}_k = B\mathbf{w}_k - \mathbf{b}$). The resulting M_i can then be used to find

$$\mathbf{w}_{i+1} = M_i\mathbf{b} \tag{2.30}$$

until a minimum is reached. It can be shown that the DFP update formula for M_i,

$$\begin{aligned} M_{i+1} = M_i &+ \frac{(\mathbf{w}_{i+1} - \mathbf{w}_i) \times (\mathbf{w}_{i+1} - \mathbf{w}_i)}{(\mathbf{w}_{i+1} - \mathbf{w}_i)^T(\mathbf{g}_{i+1} - \mathbf{g}_i)} \\ &- \frac{[M_i(\mathbf{g}_{i+1} - \mathbf{g}_i)] \times [M_i(\mathbf{g}_{i+1} - \mathbf{g}_i)]}{(\mathbf{g}_{i+1} - \mathbf{g}_i)^T M_i (\mathbf{g}_{i+1} - \mathbf{g}_i)} \end{aligned} \tag{2.31}$$

satisfies (2.29); it can be shown to converge to B^{-1} (Polak, 1971). A variant, with a slightly different update for M_{i+1}, is the BFGS update.

A disadvantage of these methods, which can be especially cumbersome for neural network training due to the number of elements ν in $\mathbf{w}$, is that storage of the M_i is quadratic in the number of weights of the network.

Conjugate gradient methods

An alternative second-order minimisation technique is *conjugate gradient optimisation* (Stoer & Bulirsch, 1980; Press *et al.*, 1986; Polak, 1971; Powell, 1977)[*]. The direction of minimisation is always chosen such that the minimisation steps in all previous directions are not spoiled. That is to say, when the direction $\mathbf{u}_i$ is chosen and a line minimisation is performed in this direction, leading to a point $\mathbf{w}_{i+1}$, then the gradient

[*]Despite this long list of references, the single best explanation of the conjugate gradient optimisation technique is given in (Shewchuk, 1994).

$\mathbf{g}_{i+1}$ at $\mathbf{w}_{i+1}$ must be perpendicular to $\mathbf{g}_i$, $\mathbf{g}_{i-1}$, ..., $\mathbf{g}_0$. The initial minimisation direction may be chosen randomly, but is usually set to $\mathbf{g}_0$.

When such directions $\mathbf{u}_i$ can be found and the approximation of E by its second order Taylor expansion is exact, the minimum will be located in ν steps, where ν is the number of free parameters of the system.

Suppose the initial direction of minimisation, which is started at $\mathbf{w}_0$, is $\mathbf{v}_0$. A line minimisation in the direction of $\mathbf{v}_0$ results in a gradient at $\mathbf{w}_1$ perpendicular to $\mathbf{v}_0$. In general,

$$\mathbf{v}_i^T \mathbf{g}_{i+1} = 0. \tag{2.32}$$

Since we do not want to spoil this minimisation step in subsequent minimisations, the gradients of subsequent points of minimisation must also be perpendicular to $\mathbf{v}_i$:

$$\mathbf{v}_i^T \mathbf{g}_{i+2} = 0. \tag{2.33}$$

Therefore, with (2.32) and (2.33),

$$\mathbf{v}_i^T (\mathbf{g}_{i+2} - \mathbf{g}_{i+1}) = 0. \tag{2.34}$$

Now, $\mathbf{g}_{i+2} - \mathbf{g}_{i+1}$ is the change in the gradient as we move from $\mathbf{w}_{i+1}$ to $\mathbf{w}_{i+2}$. As was found in (2.25), the gradient of $\tilde{E}$ at $\mathbf{w}_i$ equals $\mathbf{g}_i = B\mathbf{w}_i - \mathbf{b}$, such that

$$0 = \mathbf{v}_i^T (\mathbf{g}_{i+2} - \mathbf{g}_{i+1}) = \mathbf{v}_i^T B(\mathbf{w}_{i+2} - \mathbf{w}_{i+1}) = \mathbf{v}_i^T B\mathbf{v}_{i+1}\alpha_{i+1} \tag{2.35}$$

or

$$\mathbf{v}_i^T B\mathbf{v}_{i+1} = 0. \tag{2.36}$$

When (2.36) holds for two vectors $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ these vectors are said to be *conjugate*.

After, through line minimisation along $\mathbf{v}_i$, a point $\mathbf{w}_{i+1}$ is reached, the next minimisation direction is constructed using

$$\mathbf{v}_{i+1} = \mathbf{g}_{i+1} + \beta_i \mathbf{v}_i. \tag{2.37}$$

Conjugacy of $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ is obtained when

$$\beta_i = \frac{\mathbf{g}_{i+1}^T \mathbf{g}_{i+1}}{\mathbf{g}_i^T \mathbf{g}_i}. \tag{2.38}$$

It is easy to show, if we take

$$\alpha_i = \frac{\mathbf{g}_i^T \mathbf{g}_i}{\mathbf{u}_i^T B\mathbf{u}_i} \tag{2.39}$$

and

$$\mathbf{g}_{i+1} = \mathbf{g}_i - \alpha_i B\mathbf{u}_i, \tag{2.40}$$

that $\mathbf{v}_i^T B\mathbf{v}_{j\neq i} = 0$ ($\mathbf{v}_i$ and $\mathbf{v}_j$ conjugate) and that $\mathbf{g}_i^T \mathbf{g}_{j\neq i} = 0$ ($\mathbf{g}_i$ and $\mathbf{g}_j$ orthogonal). This method is known as *Fletcher-Reeves conjugate gradient.*

It is immediately concluded that (2.37), (2.38), and (2.39) implement the error back-propagation learning rule (2.22) with learning parameter α and momentum β thus chosen that subsequent search directions are conjugate. Therefore, conjugate gradient is a special case of error back-propagation with momentum term, such that second order information is used (the reader is reminded, however, of the difference between the two methods concerning batch and incremental learning).

A closer look at (2.39) shows that α is large when $\det(B)$ is small, i.e., there is a small second-order component in the Taylor approximation of E. In this case, where the function is almost linear, the minimisation direction follows the gradient.

Equations (2.39) and (2.40) both require the Hessian B. This embarrassment, since B is not known, can be solved by line minimisation. Instead of calculating α using (2.39) it can also be obtained via line minimisation, as described above. Furthermore, the gradient obtained at the point where the line-minimum is found is the same gradient as computed with (2.40). Thus the need of the matrix B or its inverse is eliminated.

Improvements of conjugate gradient

Although only ν iterations are needed for a quadratic system with ν degrees of freedom, due to the fact that we are not minimising quadratic systems, as well as a result of roundoff errors, the ν directions have to be followed several times. After each time the algorithm is restarted by setting $\mathbf{v}_{\nu+1} = \mathbf{g}_{\nu+1}$, thus ignoring conjugacy with previous search directions. When does one restart the minimisation? This question requires some thought, and answers to that question have been suggested by Polak and Ribière (Polak, 1971) and Powell (Powell, 1977). The resulting cost of the minimisation procedure is $O(\nu)$ which is significantly better than the linear convergence[*] of steepest descent.

The value of β_i. Equation (2.37) can give erroneous results when the system is in a region where it is not nearly quadratic. Consider the case in figure 2.5. When the search direction $\mathbf{v}_i$ is very large, and a quadratic portion near the minimum is entered where the gradient is small, a spiraling minimisation will ensue. The slow spiraling can be detected as follows. At each new minimisation step, the angle θ_i between $\mathbf{g}_i$ and $\mathbf{v}_i$ (see figure 2.6) equals

$$\cos\theta_i = \frac{\|\mathbf{g}_i\|}{\|\mathbf{v}_i\|}. \tag{2.41}$$

Replacing i by $i+1$ in figure 2.6 results in

$$\beta_i \|\mathbf{v}_i\| = \tan\theta_{i+1} \|\mathbf{g}_{i+1}\|. \tag{2.42}$$

$\|\mathbf{v}_i\|$ Can be eliminated from (2.41) and (2.42):

$$\tan\theta_{i+1} = \frac{1}{\cos\theta_i}\frac{\|\mathbf{g}_{i+1}\|}{\|\mathbf{g}_i\|} > \tan\theta_i \frac{\|\mathbf{g}_{i+1}\|}{\|\mathbf{g}_i\|}. \tag{2.43}$$

Now, if θ_i approaches $\pi/2$, the iteration may take a small step such that $(\mathbf{g}_{i+1} - \mathbf{g}_i)$ is small and the ratio $\|\mathbf{g}_{i+1}\|/\|\mathbf{g}_i\|$ approaches unity. Then, according to (2.43), θ_{i+1} is also

[*] A method is said to converge linearly if the error $E(\mathbf{w}_{i+1}) = cE(\mathbf{w}_i)$ with $c < 1$. Methods which converge with a higher power, i.e., $E(\mathbf{w}_{i+1}) = cE(\mathbf{w}_i)^m$ with $m > 1$ are called *super-linear*.

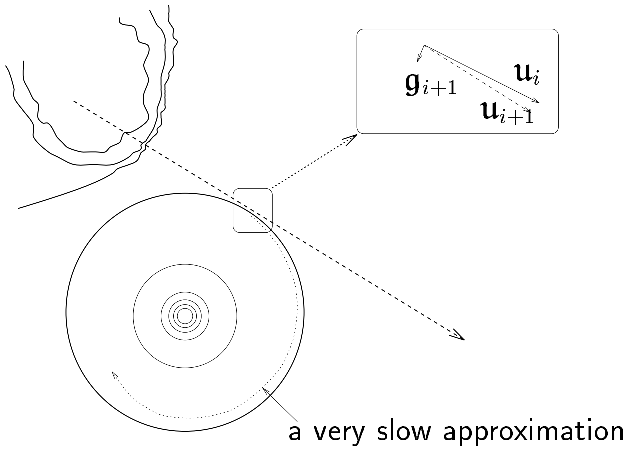

Figure 2.5: Slow decrease with conjugate gradient in non-quadratic systems.
The hills on the left are very steep, resulting in a large search vector $\mathbf{v}_i$. When the quadratic portion is entered the new search direction is constructed from the previous direction and the gradient, resulting in a spiraling minimisation.

near $\pi/2$ and the effect depicted in figure 2.5 is observed. If, however, the expression for β_i in (2.38) is replaced by

$$\beta_i = \frac{\mathbf{g}_{i+1}^T(\mathbf{g}_{i+1} - \mathbf{g}_i)}{\mathbf{g}_i^T \mathbf{g}_i}, \tag{2.44}$$

then

$$\beta_i \leq \frac{\|\mathbf{g}_{i+1}\| \cdot \|\mathbf{g}_{i+1} - \mathbf{g}_i\|}{\|\mathbf{g}_i\|^2} \tag{2.45}$$

or

$$\tan\theta_{i+1} \leq \frac{1}{\cos\theta_i} \frac{\|\mathbf{g}_{i+1} - \mathbf{g}_i\|}{\|\mathbf{g}_i\|} \tag{2.46}$$

such that $\theta_{i+1} \leq \theta_i$ and $\mathbf{v}_{i+1}$ is turned towards the steepest descent direction. Notice that this expression for β_i retains the conjugacy of $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$.

Conjugate gradient minimisation with (2.44) is known as Polak-Ribière conjugate gradient.

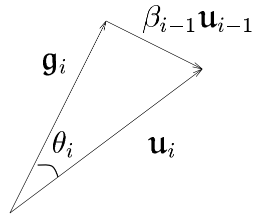

Figure 2.6: Definition of θ_i.

Powell's restart procedures. A second improvement involves the restart. It appears that restarting with $\mathbf{u}_{\nu+1} = \mathbf{g}_{\nu+1}$ is inefficient. Instead, a restarting method which does not abandon the second derivative information is needed. A proposed method is obtained by setting $\mathbf{v}_i$ according to (2.37) when a restart is made, and by extending the definition of $\mathbf{v}_i$ on normal iterations (Beale, 1972).

Let $\mathbf{v}_k$ be an arbitrary downhill restarting direction. Supposing E is quadratic, we are looking for a direction $\mathbf{v}_{i+1}$ which is a linear combination of $\mathbf{v}_k$ and the gradients $\mathbf{g}_k$, $\mathbf{g}_{k+1}$, ..., $\mathbf{g}_{i+1}$ such that $\mathbf{v}_k$, $\mathbf{v}_{k+1}$, ... are mutually conjugate. An expression which suffices these conditions is

$$\mathbf{v}_{i+1} = \mathbf{g}_{i+1} + \beta_i \mathbf{v}_i + \beta'_i \mathbf{v}_k. \tag{2.47}$$

Again, β_i is calculated to make $\mathbf{v}_{i+1}$ conjugate to $\mathbf{v}_i$, while the extra term provides conjugacy to $\mathbf{v}_k$:

$$\beta_i = \frac{\mathbf{g}_{i+1}^T(\mathbf{g}_{i+1} - \mathbf{g}_i)}{\mathbf{v}_i^T(\mathbf{g}_{i+1} - \mathbf{g}_i)}, \qquad \beta'_i = \frac{\mathbf{g}_{i+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{v}_{i+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}. \tag{2.48}$$

To prevent that the resulting direction leads uphill instead of downhill, we require that $\mathbf{v}_i^T \mathbf{g}_i > 0$ for $i > k$. Furthermore, the orthogonality between $\mathbf{g}_{i-1}$ and $\mathbf{g}_i$ must be guaranteed after restart to prevent that the approximations go to a nonzero limit. Powell suggests the following inequality:

$$\mathbf{g}_{i-1}^T \mathbf{g}_i \leq 0.2 \|\mathbf{g}_i\|^2. \tag{2.49}$$

Thirdly, the new search direction must go *sufficiently* downhill. Again following a suggestion by Powell:

$$-1.2\|\mathbf{g}_i\|^2 \leq \mathbf{v}_i^T \mathbf{g}_i \leq -0.8\|\mathbf{g}_i\|^2. \tag{2.50}$$

If (2.49) or (2.50) are not satisfied, we restart with $i = k - 1$.

2.4 Results

Where appropriate, the following methods are compared:

bp : standard error back-propagation with fixed learning rates;

sd : error back-propagation with line minimisation instead of learning rates (i.e., steepest descent minimisation);

dfp : Davidon-Fletcher-Powell quasi-Newton minimisation;

fr : Fletcher-Reeves conjugate gradient minimisation;

cg : conjugate gradient minimisation with Powell restarts.

The number of necessary error function evaluations is counted as the number of iterations. With error back-propagation, this equals the number of search directions traversed. For the other methods, due to line minimisation every search direction is traversed multiple times, and in each instance the function value as well as the derivative is computed.

The methods are compared on the following three problems:

1. classification of the four XOR patterns, to reach an average squared error of less than 0.025 per pattern;

2. approximation of the function $\sin(x)\cos(2x)$ for $0 \leq x \leq 2\pi$, from which 20 samples are uniformly chosen. An average squared error less than 0.025 per pattern must be reached;

3. approximation of the function $\tan(x)$ (with one discontinuity) for $0 \leq x \leq \pi$, from which 20 samples are uniformly chosen. An average squared error less than 0.025 per pattern must be reached.

In all instances, 10,000 trials were run.

2.4.1 Classification: the XOR problem

It has been observed (Fahlman, 1988) that the exclusive or classification problem is not a representative problem, since it does not encourage but rather penalise when the network tries to generalise the learning patterns. However, it is a difficult classification problem, and therefore we use it as a benchmark for the five minimisation methods.

For all learning methods, 10,000 trials were run with a network with two hidden units in one layer. For those runs which reached a global minimum, where a global minimum is considered reached when the summed squared error is less than 0.025 per pattern, the number of iterations needed is counted. Table 2.1 shows how often a global minimum is reached.

First, standard error back-propagation is tested with a learning rate of 0.1 and a momentum of 0.9. In 91.3% of the trials a global minimum is reached. Figure 2.7 (left) shows the number of iterations needed to reach a summed squared error of less than 0.025 per pattern; the average is located at 332. When, each step, a line minimisation

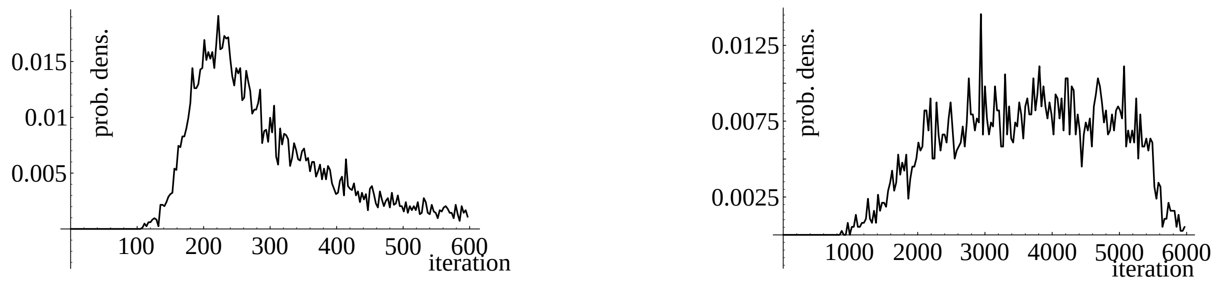

Figure 2.7: Training XOR with **bp** and **sd**.
The figures show the probability density of the number of function evaluations required to reach a global minimum (summed squared error less than 0.025 per pattern) for XOR classification with (left) standard error back-propagation and (right) error back-propagation and line minimisation (steepest descent).

is performed, the system is much more sensitive to local minima: only in 38.0% of all trials a global minimum was reached. The number of steps needed is shown in figure 2.7 (right). Notice, however, that each minimisation now takes 3 to 5 line minimisations; the average number of minimisation steps is therefore the average number of function evaluations (3661.7) divided by the average number of line minimisation steps (3 to 5).

The quasi-Newton method reaches a global minimum only in 34.1% of the cases, while the average number of learning iterations is 2141.1 (see figure 2.8 (left)). Fletcher-Reeves

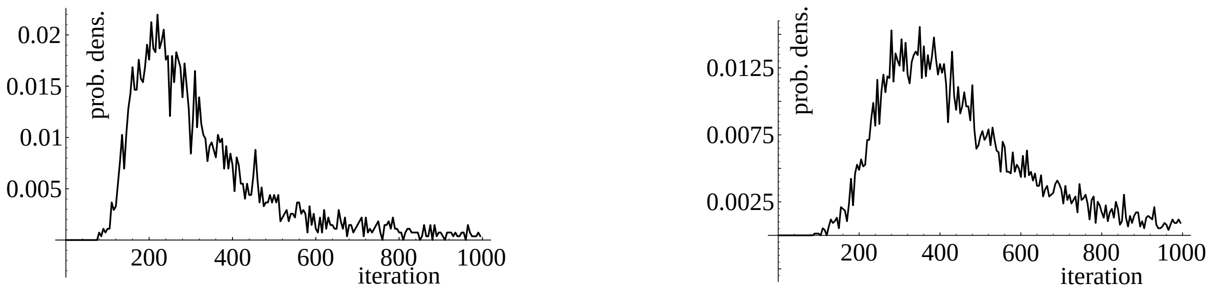

Figure 2.8: Training XOR with **dfp** and **fr**.
Probability density of the number of function evaluations required to reach a global minimum (summed squared error less than 0.025 per pattern) for XOR classification with (left) quasi-Newton DFP and (right) Fletcher-Reeves conjugate gradient.

conjugate gradient reaches the minimum in 81.5% of the cases, but needs, on the average, 523.0 iterations; see figure 2.8 (right). Finally, conjugate gradient with Powell restarts is 82.1% successful, and only needs 79.2 iterations (figure 2.9).

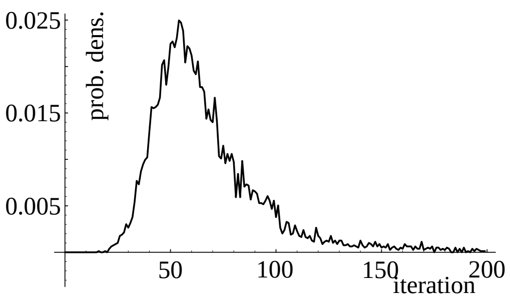

Figure 2.9: Training XOR with **cg**.
Probability density of the number of function evaluations required to reach a global minimum (summed squared error less than 0.025 per pattern) for XOR classification with Polak-Ribière conjugate gradient with Powell restarts.

2.4.2 Continuous function approximation

Secondly, the function $\sin(x)\cos(2x)$ was trained with 20 learning samples over the period $0\ldots 2\pi$ (see figure 2.10). The used network had 10 hidden units in one layer (Vyšniauskas *et al.*, 1992). Figures 2.11 and 2.12 show the number of function evaluations needed to

	XOR	$\sin(x)\cos(2x)$	$\tan(x)$
bp	91.3	15^*	0.0
sd	38.0	90	0.0
fr	81.5	49.5	4.5
dfp	34.1	35.7	40.0
cg	82.1	100.0	85.4

Table 2.1: Percentage of runs that lead to a global minimum.
A global minimum was considered reached when the summed squared error was less than 0.025 per pattern. The $*$ indicates that adaptive learning rates were used in this case.

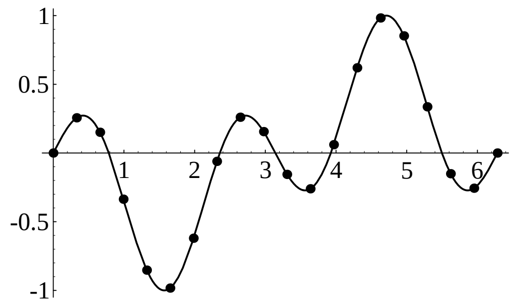

Figure 2.10: Learning samples from the function $\sin(x)\cos(2x)$.

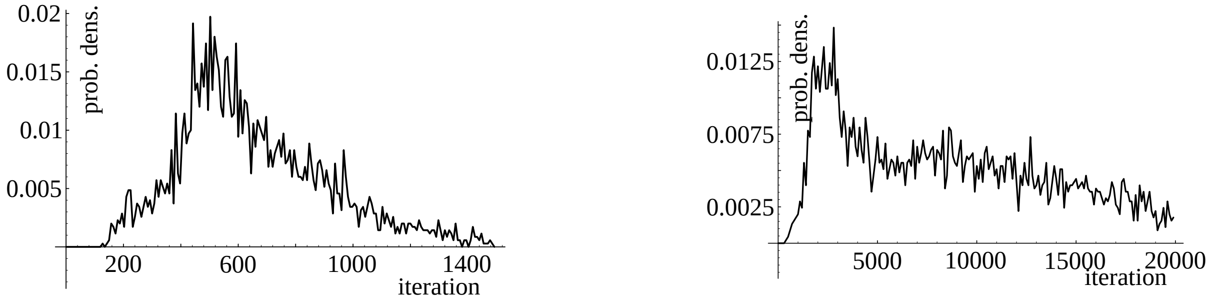

Figure 2.11: Training $\sin(x)\cos(2x)$ with $\mathbf{dfp}$ and $\mathbf{fr}$.
Probability density of the number of function evaluations required to reach a global minimum (summed squared error less than 0.025 per pattern) for $\sin(x)\cos(2x)$ approximation with (left) quasi-Newton DFP and (right) Fletcher-Reeves conjugate gradient.

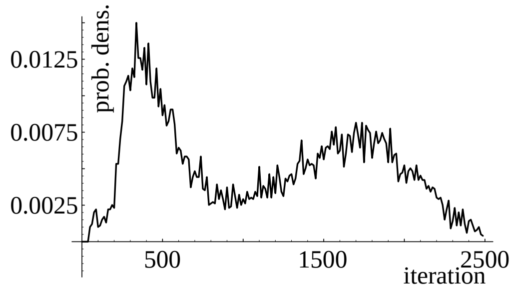

Figure 2.12: Training $\sin(x)\cos(2x)$ with **cg**.
Probability density of the number of function evaluations required to reach a global minimum (summed squared error less than 0.025 per pattern) for $\sin(x)\cos(2x)$ approximation with Polak-Ribière conjugate gradient with Powell restarts.

reach a summed squared error less than 0.025 per pattern. Although **dfp** is slightly better than **cg**, the latter always reaches a global minimum, whereas the former only reaches a minimum in 35% of the cases.

Steepest descent is not very sensitive to local minima here, but needs, on the average, $4 \cdot 10^6$ iterations to reach the minimum.

Error back-propagation was, when fixed stepsizes were used, never able to find the minimum; when the system was approaching a global minimum, the stepsize was not small enough to continue decreasing the error, such that overshoot resulted. To overcome this problem, adaptive learning rates (Silva & Almeida, 1990) were used. Still the algorithm gave poor results, often reaching local minima and needing over 2 million function evaluations.

2.4.3 Discontinuous function approximation

Finally, the function $\tan(x)$ over $0 \ldots \pi$, with one discontinuity, was tested with 20 learning samples equally distributed over the inputs space (see figure 2.13). A network with five hidden units in one hidden layer was used (Vyšniauskas *et al.*, 1992). Steepest descent

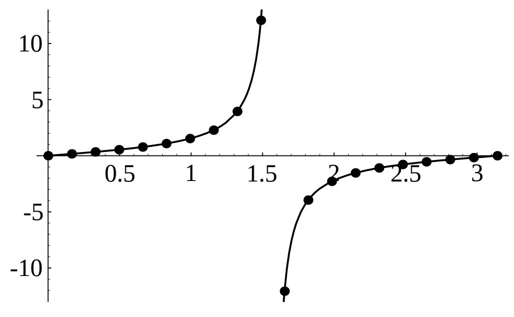

Figure 2.13: Learning samples from the function $\tan(x)$.

and error back-propagation never reached a global minimum. The other methods (see figures 2.14 and 2.15) could solve the problem.

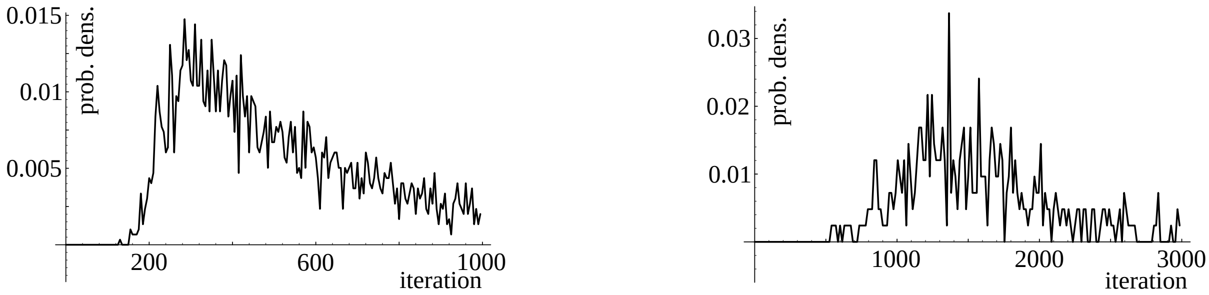

Figure 2.14: Training $\tan(x)$ with **dfp** and **fr**.
Probability density of the number of function evaluations required to reach a global minimum (summed squared error less than 0.025 per pattern) for $\tan(x)$ approximation with (left) quasi-Newton DFP and (right) Fletcher-Reeves conjugate gradient.

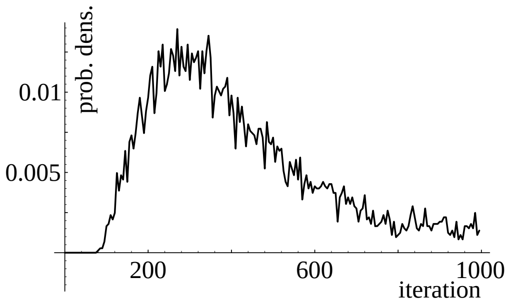

Figure 2.15: Training $\tan(x)$ with **cg**.
Probability density of the number of function evaluations required to reach a global minimum (summed squared error less than 0.025 per pattern) for $\tan(x)$ approximation with Polak-Ribière conjugate gradient with Powell restarts.

2.5 Discussion

In this chapter we have seen that a theoretical model of the approximation error using a feed-forward neural network can be found through simulation, such that an optimal number of learning samples and hidden units can be chosen for a particular problem. This method, introduced in (Vyšniauskas *et al.*, 1993), determines an asymptotic model of the approximation error as a function of the number of learning samples and hidden units used in the network. The best fit of this model to the approximation error of the feed-forward network can be used to predict the minimal computational resources to attain a certain error in the approximation.

Secondly it has been shown that, although standard error back-propagation is less sensitive to get stuck in local minima, second order minimisation methods are far superior with respect to learning time, especially in accurately approximating smooth functions. Conjugate gradient algorithms, which can be seen as error back-propagation with momentum where learning rates are optimally determined, were shown to be a better choice for feed-forward network training. In particular, Polak-Ribière conjugate gradient optimisation with Powell restarts shows promising results for training feed-forward networks due to the fact that it takes the non-quadratic behaviour of the minimised error function far from the minimum into account.

Although the problem of local minima is potentially serious, practice shows that the preferred minimisation method hardly suffers from this problem. First, it has been suggested that this problem can be alleviated by adding more dimensions to the search space (i.e., more parameters (weights) in the neural network by adding hidden units) of the optimisation method, thus creating 'escape routes' to escape from local minima. Secondly, in order to avoid getting stuck in local minima some authors suggest hybrid training algorithms (Gorse & Shepherd, 1992): switch to conjugate gradient optimisation near minima, and use error back-propagation with fixed stepsize otherwise. This method, which is also suggested in (Møller, 1990), is a technique from numerical analysis known as Levenberg-Marquardt optimisation (Press *et al.*, 1986).

The technique for finding the optimal neural network will be employed for the hand-eye coordination problem discussed in chapter 4. In all cases, the optimal Powell learning strategy will be used.