

# Chapter 1

## Introduction

In this introductory Chapter we review some parts from the theory of neural networks which are of conceptual importance for the sequel of this book. A more extensive treatment of these topics can be found in [23; 41; 71; 110; 193; 230].

### 1.1 Multilayer perceptron neural networks

In the last decade many successful results have been obtained in different areas by applying neural network techniques. A major historical breakthrough was taking place after the introduction of multilayer perceptron (MLP) architectures together with the backpropagation method for learning the interconnection weights either off-line or on-line from given input/output patterns. Previously, one had realized that the perceptron was only able to realize a linear decision boundary in the input space by constructing a hyperplane. The perceptron was unable e.g. to learn the XOR problem which can only be solved by constructing a nonlinear decision boundary. In the sixties this caused some scepticism within the neural networks community about the general applicability of neural networks. However, thanks to the introduction of additional hidden layers more powerful multilayer perceptron architectures have been created in the age of backpropagation.

According to the McCulloch-Pitts model, a neuron is modelled as a simple static nonlinear element which takes a weighted sum of incoming signals  $x_i$  multiplied with interconnection weights  $w_i$ . After adding a bias term  $b$  (also called threshold) the resulting activation  $a = \sum_i w_i x_i + b$  is

sent through a static nonlinearity  $h(\cdot)$  (activation function) yielding the output  $y$  such that

$$y = h\left(\sum_{i=1}^n w_i x_i + b\right). \quad (1.1)$$

The nonlinearity is typically of the saturation type, e.g.  $\tanh(\cdot)$ . Biologically this corresponds to the firing of a neuron depending on gathered information of incoming signals that exceeds a certain threshold value. This McCulloch-Pitts model is a simple model for a biological neuron that has been frequently used within artificial neural network models. On the other hand one should also note that much more sophisticated models exist for biological neurons and the goal of artificial neural networks is not to mimic biology but rather to create a powerful class of mathematical models.

By means of a single neuron one can form a perceptron architecture as shown in Fig. 1.1. A more powerful model of a multilayer perceptron (MLP) is obtained by adding one or more hidden layers (Fig. 1.1). It is a static nonlinear model  $y = f(x)$  which is described as follows in matrix-vector notation:

$$y = W \tanh(Vx + \beta) \quad (1.2)$$

with input  $x \in \mathbb{R}^n$ , output  $y \in \mathbb{R}^{n_y}$  and interconnection matrices  $W \in \mathbb{R}^{n_y \times n_h}$ ,  $V \in \mathbb{R}^{n_h \times n}$  for the output layer and hidden layer, respectively. The bias vector is  $\beta \in \mathbb{R}^{n_h}$  and consists of the threshold values of the  $n_h$  hidden neurons. In these descriptions a linear activation function is taken for the output layer. Depending on the application one might choose other functions as well. For problems of nonlinear function estimation and regression, one takes a linear activation function in the output layer. Sometimes a neural network with two hidden layers is chosen.

One of the reasons for the neural networks success story is the fact that these are *universal approximators*. It has been mathematically proven that MLPs can approximate any continuous nonlinear function arbitrarily well over a compact interval to any degree of accuracy provided they contain one or more hidden layers. The history of these universal approximation theorems dates back in fact from the beginning of the previous century around 1900, when Hilbert formulated a list of challenging mathematical problems for the century to come. In his famous 13th problem he formulated the conjecture about the existence of analytical functions of three variables which

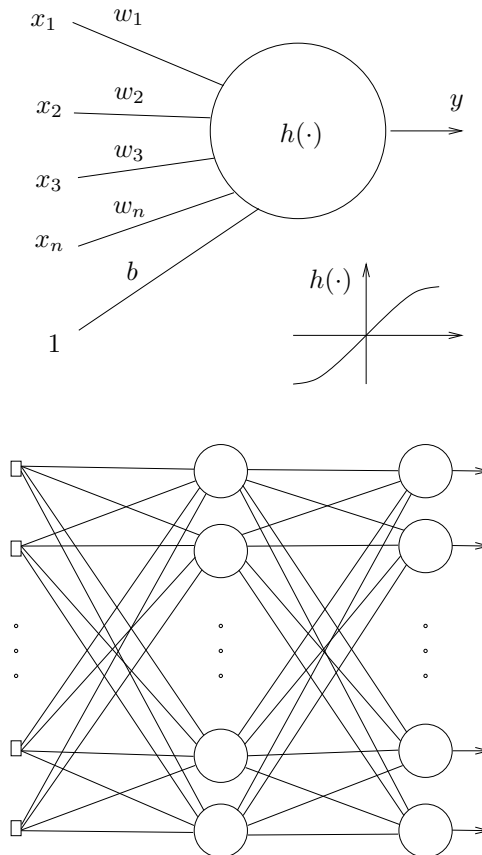


Fig. 1.1 (Top) McCulloch-Pitts model of a neuron. A model consisting of a single neuron gives a perceptron. (Bottom) A more powerful model shown in this figure is the multilayer perceptron which is a universal approximator provided it contains one or more hidden layers.

cannot be represented as a finite superposition of continuous functions of only two variables. This conjecture was refuted by Kolmogorov and Arnold in 1957. In 1987 Hecht-Nielsen made a connection between a refined version of the Kolmogorov theorem made by Sprecher in 1965 and showed that MLPs with two hidden layers can represent any continuous mapping. Later in 1989 it was shown by Hornik and others that one hidden layer is sufficient for universal approximation. The role of the activation function was

further studied by Leshno in 1993, who showed that multilayer feedforward neural networks with locally bounded piecewise continuous activation function can approximate any continuous function to any degree of accuracy if and only if the network's activation function is not a polynomial.

Universal approximation is a nice property of neural networks. However, one can argue that also polynomial expansions possess this property. So is there any reason to prefer neural networks instead of polynomial expansions? This question brings us to a second important property of neural networks which is that they are better able to cope with the *curse of dimensionality*. In 1993 Barron showed that neural networks can avoid the curse of dimensionality in the sense that the approximation error becomes independent of the dimension of the input space (under certain conditions), which is not the case for polynomial expansions. The approximation error for MLPs with one hidden layer is of order of magnitude  $\mathcal{O}(1/n_h)$ , but  $\mathcal{O}(1/n_p^{2/n})$  for polynomial expansions where  $n_h$  denotes the number of hidden units,  $n$  the dimension of the input space and  $n_p$  the number of terms in the expansion. Models that are based on MLPs will be able to better handle larger dimensional input spaces than polynomial expansions, which is an interesting property towards many real-life problems where one has to model dependencies between several variables.

## 1.2 Regression and classification

The first algorithm for the training of multilayer perceptrons and feedforward networks in general was the backpropagation algorithm. In the case of batch learning, this involves minimizing the residual squared error cost function in the unknown interconnection weights

$$\min_{\theta \in \mathbb{R}^p} J_{\text{train}}(\theta) = \frac{1}{N} \sum_{k=1}^N \|y_k - f(x_k; \theta)\|_2^2 \quad (1.3)$$

where  $\theta = [W(\cdot); V(\cdot); \beta] \in \mathbb{R}^p$  is a vector of  $p$  unknowns containing the weights and bias term. The training set of given input/output data is  $\{x_k, y_k\}_{k=1}^N$  where  $N$  is the number of training data. The basic backpropagation (BP) algorithm is in essence a steepest descent local optimization algorithm for minimizing this objective function. Improvements with momentum term and adaptive learning rate have been developed as well. The BP algorithm is an elegant method for obtaining an analytic expression for

the gradient of the cost function. The generalized delta rule gives recursive formulas for computing the gradient for an arbitrary number of hidden layers. The name backpropagation stems from the fact that the input patterns are propagated in a forward phase towards the output layer while the error made at the output layer is backpropagated in a backward phase towards the input layer. This is done by computing so-called delta variables based upon which the values of the interconnection weights are modified. BP can also be considered as an extension of the LMS algorithm which is well-known in the area of adaptive signal processing [109]. Several improved training methods have been developed both for off-line and on-line learning. For batch learning methods, insights from optimization theory have been taken e.g. in order to apply quasi-Newton and Levenberg-Marquardt algorithms. In these methods one usually makes approximations to the Hessian matrix. In comparison with BP, the convergence of these methods is much faster. Typically, for networks with more than about one thousand interconnection weights it might be better to apply conjugate gradient methods which don't require the storage of huge matrices. For efficient on-line learning extended Kalman filtering techniques have been applied.

A well-known problem which should be avoided with the training of neural networks is overfitting. This problem especially occurs when one optimizes a cost function of the form (1.3) in case one continues training until the local minimum is reached. Therefore, in daily neural networks practice one often divides the total number of data into a training set, validation set and test set. The validation set is used in order to decide about when to stop training (i.e. when the minimal error on this validation set is obtained). The test data are completely left untouched within the training and validation process (Fig. 1.2).

When using MLPs for function approximation one often works with the model (1.2) which contains one hidden layer of tanh activation functions and an output layer consisting of neurons with a linear characteristic. In the case of solving classification problems by MLPs one can also use tanh activation functions in the output layer as for the hidden layer. Nevertheless, often a linear characteristic is taken as in the function approximation case. The network is usually trained then in the form (1.2) and the classifier is then evaluated as  $y(x) = \text{sign}[f(x)]$  or

$$y(x) = \text{sign}[W \tanh(Vx + \beta)]. \quad (1.4)$$

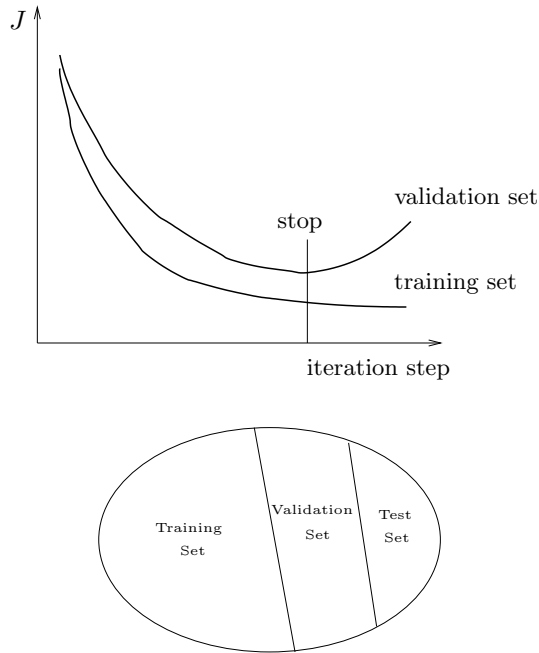


Fig. 1.2 Training, validation and test set where the validation set is used for early stopping of the training process in order to obtain a good generalization. The test set is completely left untouched during the training and early stopping process and is used to check the performance of the trained model on fresh data.

The network is trained then with desired output values  $y_k \in \{-1, +1\}$  for binary class problems. In the multi-class case one takes additional outputs in order to represent the classes. One is confronted then with the question of how to choose the coding-decoding of the classes. One has experienced that it is good to employ as many outputs as the number of classes, but better schemes might exist as we will discuss later in this book. In Fig. 1.3 an illustration is given on the recognition of the letters of the alphabet.

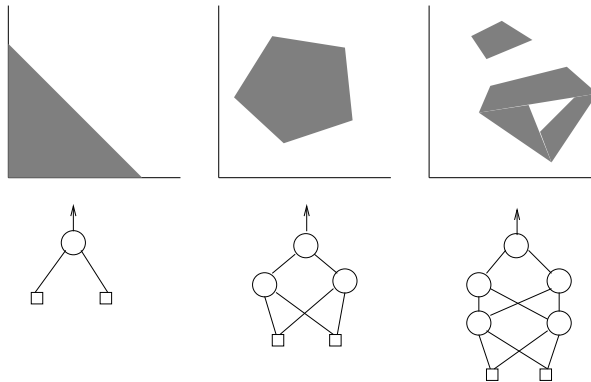
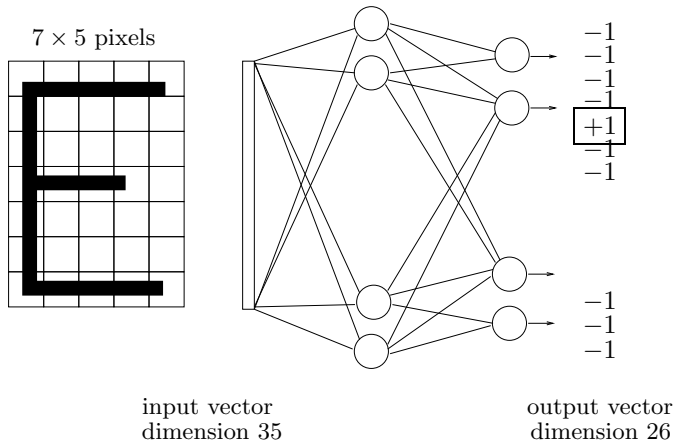


Fig. 1.3 (Top) Illustration of a regression approach of MLPs to a multiclass classification problem on alphabet recognition; (Bottom) Complex decision boundaries can be realized by adding hidden layers. A perceptron which consists of a single neuron can only realize a linear separating hyperplane.

## 1.3 Learning and generalization

### 1.3.1 Weight decay and effective number of parameters

After an initial boom of MLP applications with the BP algorithm in the early nineties, the problems have been investigated in an interdisciplinary

manner with contributions from several fields such as statistics, engineering and physics. One of the important ideas was to add a weight decay term to the cost function and modify (1.3) into

$$\min_{\theta \in \mathbb{R}^p} J_{\text{reg}}(\theta) = \nu \frac{1}{2} \theta^T \theta + \frac{1}{N} \sum_{k=1}^N \|y_k - f(x_k; \theta)\|_2^2 \quad (1.5)$$

which means that one should keep the values of the interconnection weights small in addition to obtaining a good fit of the data. Thanks to the regularization mechanism one can work with an effective number of parameters that is less than the number of interconnection weights depending on the choice of the regularization constant  $\nu$ . This means that even when the number of neurons is large, it is still possible that one can implicitly work with a much smaller number of parameters and obtain a good generalization.

The *effective number of parameters* is given by

$$d_{\text{eff}} = \sum_{i=1}^p \frac{\lambda_i}{\lambda_i + \nu} \quad (1.6)$$

where  $\lambda_i$  denotes the  $i$ -th eigenvalue of the (positive definite) Hessian matrix at a point in the interconnection weight space for the unregularized problem. When many of the values  $\lambda_i \ll \nu$ , the influence of many interconnection weights will be implicitly suppressed by the regularization mechanism [23; 149]. In this way the overfitting problem can be avoided without doing early stopping of the training process provided that a good choice is made of the regularization constant. Also one can show that an early stopping process implicitly corresponds to a certain form of regularization. The advantage of suppressing the influence of interconnection weights is clear from complexity criteria such as Moody's criterion which states that the generalized prediction error (GPE) equals

$$\text{GPE} = J_{\text{train}} + \frac{d_{\text{eff}}}{N} \sigma_e^2 \quad (1.7)$$

where  $\sigma_e^2$  denotes the variance of the noise. This can be considered as an extension of the well-known Akaike information criterion from linear to nonlinear models where the number of unknown parameters is replaced by the effective number of parameters.

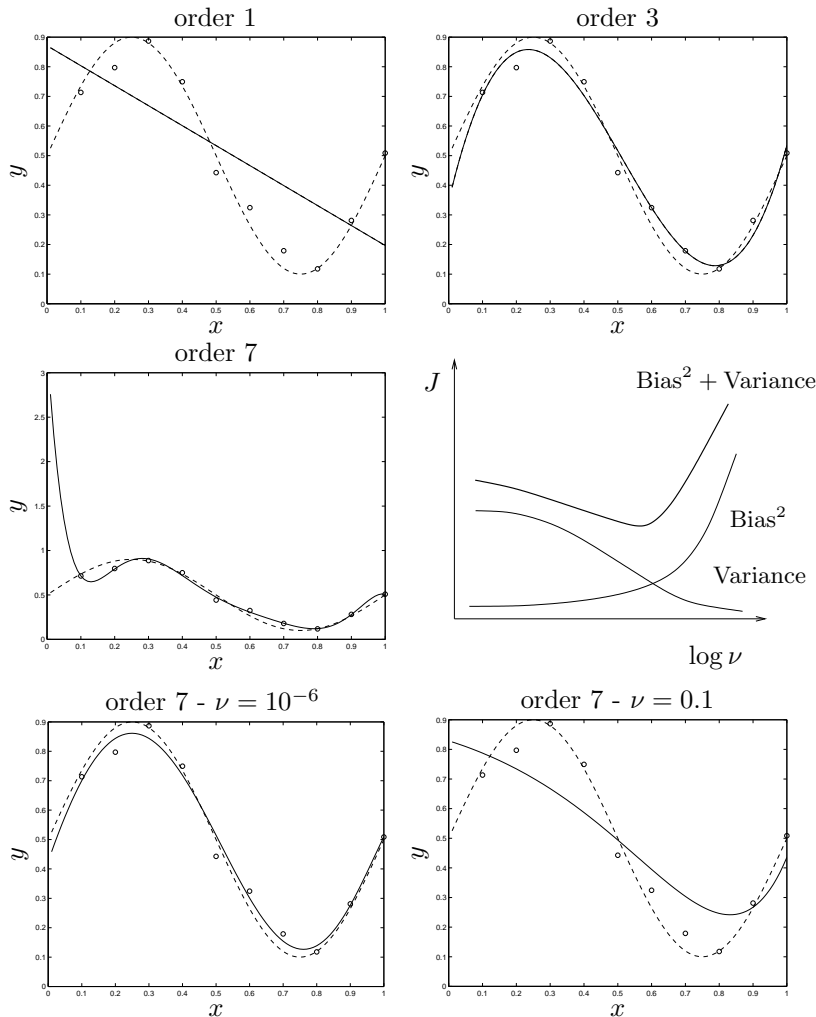


Fig. 1.4 Estimation of a noisy sine function: polynomial models of order 1 (Top-left), order 3 (Top-right) and order 7 (Middle-left); (Middle-right) Bias-variance trade-off when applying regularization; (Bottom) ridge regression applied to the polynomial model of order 7 with  $\nu = 10^{-6}$  (left) and for  $\nu = 0.1$  (right).

### 1.3.2 Ridge regression

The use of weight decay is a parametric form of regularization and is closely related to ridge regression. Let us illustrate this by means of a simple ex-

ample. Assume that training data are generated from the true underlying function  $f_0(x) = 0.5 + 0.4\sin(2\pi x)$  with training data  $x$  generated in the interval  $[0.1, 1]$  with steps of 0.1 and zero mean Gaussian noise with standard deviation 0.05 is added to the output values, giving a training data set  $\{x_k, y_k\}_{k=1}^{10}$ . From the training data polynomials of different degrees are estimated. For example, for the polynomial model of degree 3

$$y = a_1x + a_2x^2 + a_3x^3 + b \quad (1.8)$$

an overdetermined set of linear equations is constructed for the given data  $\{x_k, y_k\}_{k=1}^{10}$

$$\begin{bmatrix} x_1 & x_1^2 & x_1^3 & 1 \\ x_2 & x_2^2 & x_2^3 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{10} & x_{10}^2 & x_{10}^3 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{10} \end{bmatrix}. \quad (1.9)$$

This overdetermined system is of the form  $\mathcal{A}\theta = \mathcal{B}$  with  $\mathcal{A} \in \mathbb{R}^{q \times p}$  where  $q > p$ ,  $\theta = [a_1; a_2; a_3; b]$  and  $e = \mathcal{A}\theta - \mathcal{B}$ . By taking a cost function in a linear least squares sense,  $\min_{\theta} J_{LS}(\theta) = \frac{1}{2}e^T e = \frac{1}{2}(\mathcal{A}\theta - \mathcal{B})^T(\mathcal{A}\theta - \mathcal{B})$ , the condition for optimality  $\partial J_{LS}(\theta)/\partial\theta = 0$  gives the solution [98]

$$\theta_{LS} = (\mathcal{A}^T \mathcal{A})^{-1} \mathcal{A}^T \mathcal{B} = \mathcal{A}^\dagger \mathcal{B} \quad (1.10)$$

where  $\mathcal{A}^\dagger$  denotes the pseudo inverse matrix. One can observe that for the higher order polynomial (order 7) the solution starts oscillating and overfitting is obtained because the polynomial interpolates the given training data but fails to generalize well in between the given training data points.

Let us now modify the least squares cost function by an additional term which aims at keeping the norm of the solution vector small. In ridge regression one solves  $\min_{\theta} J_{\text{ridge}}(\theta) = J_{LS}(\theta) + \frac{1}{2}\nu\|\theta\|_2^2, \nu > 0$  which gives the following solution after taking the condition for optimality  $\partial J_{\text{ridge}}(\theta)/\partial\theta = 0$  [98]

$$\theta_{\text{ridge}} = (\mathcal{A}^T \mathcal{A} + \nu I)^{-1} \mathcal{A}^T \mathcal{B}. \quad (1.11)$$

This technique is useful when  $\mathcal{A}^T \mathcal{A}$  is *ill conditioned*. The results of ridge regression for the order 7 polynomial model is shown on Fig. 1.4 for different values of the regularization constant  $\nu$ . Figure 1.4 conceptually shows the bias-variance trade-off in terms of the regularization constant  $\nu$ . A large value  $\nu$  decreases the variance but leads to a larger bias. This value of

$\nu$  is chosen as a trade-off solution by minimizing the sum of the variance and the bias square contributions. According to e.g. [107](pp.196-200) the bias-variance trade-off can be understood as follows. Assume a model of the form  $Y = f(x) + e$  where  $\mathcal{E}[e] = 0$  and  $\text{Var}(e) = \sigma_e^2$  with random output variable  $Y$ . The expected prediction error of a regression fit  $\hat{f}(x)$  evaluated at input  $x = x_0$  is then given by the following expression when using a squared error loss function:

$$\begin{aligned} \text{PE}(x_0) &= \mathcal{E}[(Y - \hat{f}(x_0))^2 | x = x_0] \\ &= \sigma_e^2 + (\mathcal{E}[\hat{f}(x_0)] - f(x_0))^2 + \mathcal{E}[\hat{f}(x_0) - \mathcal{E}[\hat{f}(x_0)]]^2 \quad (1.12) \\ &= \sigma_e^2 + \text{Bias}^2[\hat{f}(x_0)] + \text{Var}[\hat{f}(x_0)]. \end{aligned}$$

The first term is the variance of the target around its mean  $f(x_0)$  and is an irreducible error no matter how well one estimates  $f(x_0)$ . The second term is the squared bias which characterizes the amount by which the average of our estimates differ from the true mean. The last term is the variance which is the expected squared deviation around its mean. The interpretation for the present example on ridge regression is that the more complex we make the model (higher degree of the polynomial) the lower the bias but the larger the variance.

In order to make a suitable choice for the regularization constant  $\nu$  one has several options then. The following methods are popular in the neural networks area: (a) taking a value for the regularization constant that minimizes the error on a separate validation set; (b) instead of working with a single validation set one can work with a  $n_{\text{CV}}$ -fold cross-validation procedure (Fig. 1.5) and take  $\nu$  in such a way that sum of the errors on the validation sets that are left out in the several runs is minimal; (c) applying methods of Bayesian inference which enable to do automatic selection of the hyperparameters.

### 1.3.3 Bayesian learning

In Bayesian learning [23; 149; 150; 151] a distribution is considered on the unknown parameter vector  $\theta$ , rather than considering  $\theta$  as a point in parameter space. For data  $\mathcal{D}$  and parameter vector  $\theta$  the application of Bayes' rule gives

$$p(\theta | \mathcal{D}) = \frac{p(\mathcal{D} | \theta)}{p(\mathcal{D})} p(\theta) \quad (1.13)$$

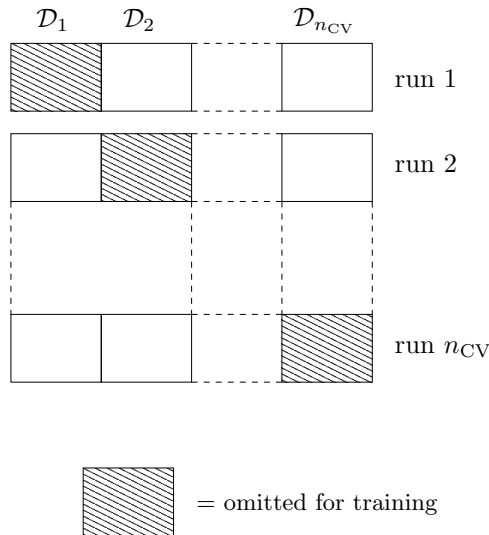


Fig. 1.5 In 10-fold cross-validation a number of  $n_{CV} = 10$  runs are done where in each run a different part of the data set is omitted and the validation error is finally checked as the sum of error costs on the sets that were omitted.

meaning that

$$\text{Posterior} = \frac{\text{Likelihood}}{\text{Evidence}} \times \text{Prior}. \quad (1.14)$$

For i.i.d. data  $x_k$  ( $k = 1, \dots, N$ ) one has the likelihood  $p(\mathcal{D}|\theta) = \prod_{k=1}^N p(x_k|\theta)$ . The normalization factor  $p(\mathcal{D}) = \int p(\theta') \prod_{k=1}^N p(x_k|\theta') d\theta'$  ensures  $\int p(\theta|\mathcal{D}) d\theta = 1$ . Figure 1.6 illustrates the process of obtaining an estimate for the posterior  $p(\theta|\mathcal{D})$  by combining the prior  $p(\theta)$  with the data  $\mathcal{D}$ . In advanced methods for Bayesian inference of neural networks, this Bayes rule is applied at different levels. At a first level one considers a density in the unknown parameter values of the interconnection weights. The regularization weight decay term is related to the prior (keeping the weights small), while the likelihood corresponds to the training set error. Inference at the second level is done in the unknown hyperparameters which are the regularization constants that penalize the importance of the regularization term versus the training set error. Application of the Bayes rule at this level

leads to implicit formulas for automatic selection of suitable regularization constants, which depend on the effective number of parameters. At the third level of inference model comparison can be done where the Bayes rule embodies the principle of Occam's razor which states that simple models are preferred.

More specifically, a Bayesian learning approach to regression starts from the following cost function with regularization

$$\min_{\theta \in \mathbb{R}^p} J_{\text{reg}}(\theta) = \mu \frac{1}{2} \theta^T \theta + \zeta \frac{1}{2} \sum_{k=1}^N \|y_k - f(x_k; \theta)\|_2^2 \quad (1.15)$$

where  $\theta \in \mathbb{R}^p$  denotes the unknown parameters of the neural network and  $\mu, \zeta$  are called *hyperparameters*. Bayesian inference is done at the following 3 levels:

- *Level 1* [inference of parameters]

$$p(\theta | \mathcal{D}, \mu, \zeta, \mathcal{H}_i) = \frac{p(\mathcal{D} | \theta, \mu, \zeta, \mathcal{H}_i)}{p(\mathcal{D} | \mu, \zeta, \mathcal{H}_i)} p(\theta | \mu, \zeta, \mathcal{H}_i) \quad (1.16)$$

- *Level 2* [inference of hyperparameters]

$$p(\mu, \zeta | \mathcal{D}, \mathcal{H}_i) = \frac{p(\mathcal{D} | \mu, \zeta, \mathcal{H}_i)}{p(\mathcal{D} | \mathcal{H}_i)} p(\mu, \zeta | \mathcal{H}_i) \quad (1.17)$$

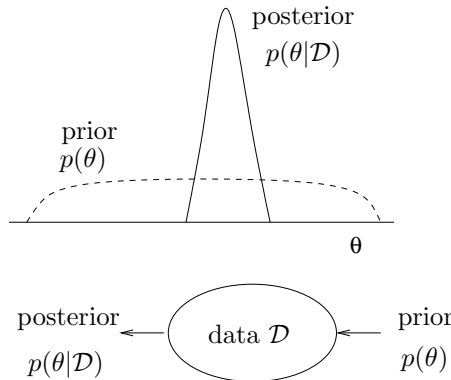


Fig. 1.6 Bayesian inference: starting from a prior  $p(\theta)$  with a large uncertainty on the parameter vector  $\theta$ , the data  $\mathcal{D}$  are used in order to generate a posterior  $p(\theta|\mathcal{D})$  which becomes more accurate.

- *Level 3* [model comparison]

$$p(\mathcal{H}_i|\mathcal{D}) \propto p(\mathcal{D}|\mathcal{H}_i)p(\mathcal{H}_i) \quad (1.18)$$

where  $\mathcal{H}_i$  denotes the  $i$ -th model ( $i = 1, \dots, n_{\mathcal{H}}$ ) with  $n_{\mathcal{H}}$  the number of models to be compared (this could for example be models with different numbers of hidden neurons). The connection between the posterior and prior on  $\theta$  at Level 1 and the cost function (1.15) is made as follows:

$$\begin{aligned} p(\theta|\mathcal{D}, \mu, \zeta, \mathcal{H}_i) &\propto \exp\left(-\zeta \frac{1}{2} \sum_{k=1}^N \|y_k - f(x_k; \theta)\|_2^2\right) \\ p(\theta|\mu, \zeta, \mathcal{H}_i) &\propto \exp\left(-\mu \frac{1}{2} \theta^T \theta\right). \end{aligned} \quad (1.19)$$

The connection between Level 1 and Level 2 is made by observing that the Evidence at Level 1 for the inference of  $\theta$  equals the Likelihood at Level 2 for the inference of the hyperparameters  $\mu, \zeta$ .

Other techniques that have been used in the neural networks area in order to improve generalization are *pruning methods* where one starts from a neural network with a large number of interconnection weights and gradually prunes the least relevant weights by computing so-called saliency values. Several versions have been developed such as optimal brain damage and optimal brain surgeon. One can also improve the generalization by combining several models and improve the results in view of the bias-variance trade-off. This is done by so-called committee networks (Fig. 1.7).

## 1.4 Principles of pattern recognition

### 1.4.1 *Bayes rule and optimal classifier under Gaussian assumptions*

The link between artificial neural networks and methods in pattern recognition and statistical decision theory has been quite well understood [23]. Suppose we consider a number of  $n_{\mathcal{C}}$  classes and continuous variables  $x$  belonging to the input space. Application of the Bayes rule gives then

$$P(\mathcal{C}_i|x) = \frac{p(x|\mathcal{C}_i)}{p(x)} P(\mathcal{C}_i) , i = 1, \dots, n_{\mathcal{C}} , x \in \mathbb{R}^n \quad (1.20)$$

with normalization  $\sum_{i=1}^{n_{\mathcal{C}}} P(\mathcal{C}_i|x) = 1$  and unconditional density  $p(x) = \sum_{i=1}^{n_{\mathcal{C}}} p(x|\mathcal{C}_i)P(\mathcal{C}_i)$ , where  $P(\mathcal{C}_i)$  denotes the prior class probability of class

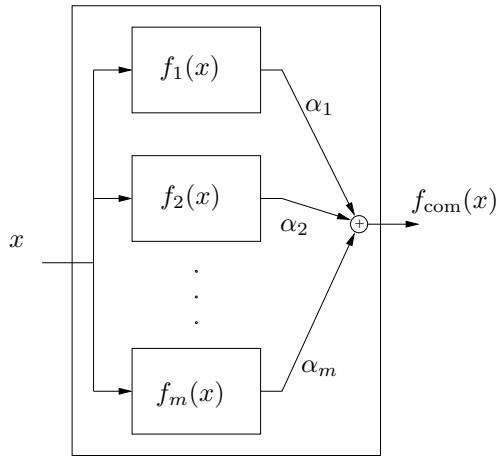


Fig. 1.7 Combining models  $f_i(x)$  for  $i = 1, \dots, m$  into a committee network  $f_{\text{com}}(x) = \sum_{i=1}^m \alpha_i f_i(x)$  in order to improve the estimate in view of the bias-variance trade-off.

$\mathcal{C}_i$ ,  $P(\mathcal{C}_i|x)$  the posterior class probability of class  $\mathcal{C}_i$  and  $p(x|\mathcal{C}_i)$  the class conditional density. If one assigns a pattern  $x$  to a class  $i^*$  such that

$$i^* = \arg \max_{i=1, \dots, n_c} P(\mathcal{C}_i|x) \quad (1.21)$$

one can show that this classification rule which is based on the posterior class probability leads to a *minimal probability of misclassification*. In Fig. 1.8 it is illustrated for a binary class problem that this decision rule leads to a minimal shaded area determined by the curves  $p(x|\mathcal{C}_1)P(\mathcal{C}_1)$  and  $p(x|\mathcal{C}_2)P(\mathcal{C}_2)$ . Choosing a threshold decision line at the intersection of the two curves yields a minimal misclassification.

In the context of discriminant functions the Bayes decision rule can be interpreted as assigning  $x$  to class  $\mathcal{C}_i$  such that

$$d_i(x) > d_j(x), \quad \forall i \neq j \quad (1.22)$$

where  $d_i(x)$ ,  $d_j(x)$  denote discriminant functions. The case

$$d_i(x) \propto p(x|\mathcal{C}_i)P(\mathcal{C}_i) \quad (1.23)$$

corresponds then to minimizing the probability of misclassification. Only relative magnitudes of discriminant functions are important. One may ap-

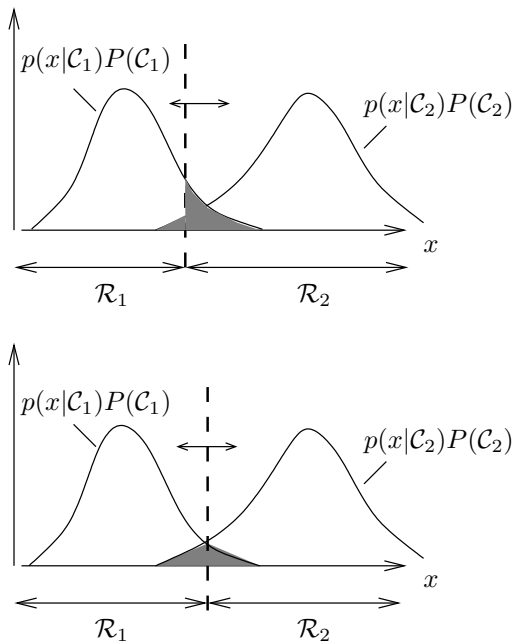


Fig. 1.8 A minimal shaded area corresponds to minimizing the probability of misclassification (bottom figure), when considering a moving threshold line (dashed vertical line).

ply a monotonic function  $g(\cdot)$  as  $g(d_i(x))$ , e.g.:  $d_i(x) = \log[p(x|\mathcal{C}_i)P(\mathcal{C}_i)] = \log p(x|\mathcal{C}_i) + \log P(\mathcal{C}_i)$ . The decision boundaries  $d_i(x) = d_j(x)$  are not influenced by the choice of this monotonic function. In the case of a binary classification problem one may take a reformulation by a single discriminant function  $d(x) = d_1(x) - d_2(x)$  with class  $\mathcal{C}_1$  if  $d(x) > 0$  and class  $\mathcal{C}_2$  if  $d(x) < 0$ . Instead of using two discriminant functions one can take a single one then.

It is often meaningful then to *assume* that the densities  $p(x|\mathcal{C}_i)$  are normally distributed for all classes  $i = 1, \dots, n_C$  and consider the discriminant functions [71; 255]

$$d_i(x) = \log p(x|\mathcal{C}_i) + \log P(\mathcal{C}_i). \quad (1.24)$$

Recall that a multivariate normal density is given by  $p(x) = ((2\pi)^n |\Sigma_{xx}|)^{-1/2} \exp(-\frac{1}{2}(x - \mu_x)^T \Sigma_{xx}^{-1} (x - \mu_x))$  with  $\int_{-\infty}^{\infty} p(x) dx = 1$ ,  $\mu_x = \mathcal{E}[x] \in \mathbb{R}^n$

the mean,  $\Sigma_{xx} = \mathcal{E}[(x - \mu_x)(x - \mu_x)^T] \in \mathbb{R}^{n \times n}$  the covariance matrix ( $\Sigma_{xx} = \Sigma_{xx}^T > 0$ ), and  $|\Sigma_{xx}|$  the determinant of  $\Sigma_{xx}$ . Under this Gaussian assumption one obtains the following discriminant functions

$$d_i(x) = -\frac{1}{2}(x - \mu_{x_i})^T \Sigma_{xx_i}^{-1} (x - \mu_{x_i}) - \frac{1}{2} \log |\Sigma_{xx_i}| + \log P(\mathcal{C}_i). \quad (1.25)$$

The decision boundaries which are characterized by  $d_i(x) = d_j(x)$  are quadratic forms in  $\mathbb{R}^n$ . For the binary classification case one obtains the following important special cases (Fig. 1.9):

- Case  $\Sigma_{xx_1} = \Sigma_{xx_2} = \Sigma_{xx}$ :

For equal covariance matrices the decision boundary becomes linear, for distributions having a small overlap as well as a large overlap. The fact that a linear decision boundary is optimal in the case of overlapping distributions means in fact that one has to tolerate misclassifications. Allowing no misclassifications in such a case on training data, which would have been generated from such underlying distributions, would lead to an overfitting situation in this classification context. This result also gives a better insight between neural networks and overfitting for classification problems.

- Case  $\Sigma_{xx_1} = \Sigma_{xx_2} = \Sigma_{xx} = \sigma_x^2 I$ :

Under this assumption one obtains

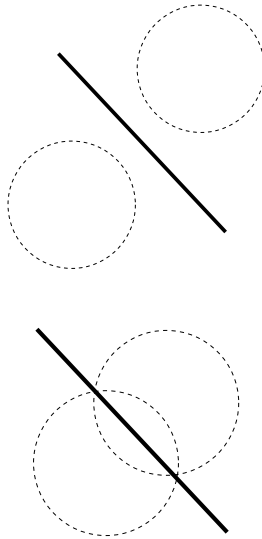
$$d_i(x) = -\frac{\|x - \mu_{x_i}\|_2^2}{2\sigma_x^2} + \log P(\mathcal{C}_i), \quad i \in \{1, 2\}. \quad (1.26)$$

If the prior class probabilities are equal, then the mean vectors  $\mu_{x_1}, \mu_{x_2}$  act as prototypes. Under these assumptions the classification can be done by a simple calculation of the Euclidean distance to  $\mu_{x_i}$ . When applying this kind of classification rule in a more general context one should be aware of the fact that this classification rule is only optimal under these restrictive assumptions.

The link between these results and neural network architectures can be made now as follows. For normally distributed class-conditional densities, the posterior probabilities can be obtained by a logistic one-neuron network

$$y = h(w^T x + b) \quad (1.27)$$

Case  $\Sigma_{xx_1} = \Sigma_{xx_2} = \Sigma_{xx}$



Case  $\Sigma_{xx_1} \neq \Sigma_{xx_2}$

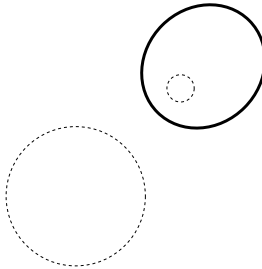


Fig. 1.9 Optimal linear and quadratic decision boundaries under different assumptions for the covariance matrices of the classes in a binary classification problem.

with logistic activation function  $h(a) = 1/(1 + \exp(-a)) \in [0, 1]$  and activation  $a = w^T x + b$ . For class-conditional densities with  $\Sigma_{xx_1} = \Sigma_{xx_2} = \Sigma_{xx}$  we had  $p(x|\mathcal{C}_i) = ((2\pi)^n |\Sigma_{xx_i}|)^{1/2} \exp(-\frac{1}{2}(x - \mu_{x_i})^T \Sigma_{xx_i}^{-1} (x - \mu_{x_i}))$ . This leads to the following posterior

$$P(\mathcal{C}_1|x) = h(w^T x + b) \quad (1.28)$$

with

$$\begin{aligned} w &= \Sigma_{xx}^{-1}(\mu_{x_1} - \mu_{x_2}) \\ b &= -\frac{1}{2}\mu_{x_1}^T \Sigma_{xx}^{-1} \mu_{x_1} + \frac{1}{2}\mu_{x_2}^T \Sigma_{xx}^{-1} \mu_{x_2} + \log \frac{P(C_1)}{P(C_2)}. \end{aligned} \quad (1.29)$$

Note that the bias term  $b$  depends on the prior class probabilities (which are often unknown in practice). Hence, different prior class probabilities will lead to a translational shift of the hyperplane or straight line in the case of a two dimensional feature space. In the other case when the above mentioned assumptions do not hold one may apply techniques for density estimation such as mixture models.

### 1.4.2 Receiver operating characteristic

For binary classification problems one can consider the following so-called confusion matrix shown in Fig. 1.10 with TP the number of correctly classified positive cases, TN the number of correctly classified negative cases, FP the number of wrongly classified positive cases, FN the number of wrongly classified negative cases. Negative/positive means class 1/class 2 (the meaning of negative or positive might depend on the specific application area, e.g. in biomedicine it means malignant/benign) and true/false means correctly/wrongly classified data, respectively.

It is convenient then to define

$$\begin{aligned} \text{Sensitivity} &= \text{TP}/(\text{TP} + \text{FN}) \\ \text{Specificity} &= \text{TN}/(\text{FP} + \text{TN}) \end{aligned} \quad (1.30)$$

$$\text{False positive rate} = 1 - \text{Specificity} = \text{FP}/(\text{FP} + \text{TN}).$$

The receiver-operating characteristic (ROC) curve [249; 250] shows the sensitivity with respect to the false positive rate (Fig. 1.10). The larger the area under the ROC curve the better the classifier, which is achieved when the classifier has a high sensitivity for a small false positive rate. This ROC curve method has been applied since the second world war where it was used on radar signals. Later it became popular in biomedical application areas. These performances should be evaluated not only on training data but also on test data. In Fig. 1.10 classifier  $C$  is better than  $B$  and the classifier with curve  $A$  has no discriminatory power at all. The points  $T_1, T_2, T_3$  illustrated on a ROC curve  $B$  on Fig. 1.10 are examples of different operating

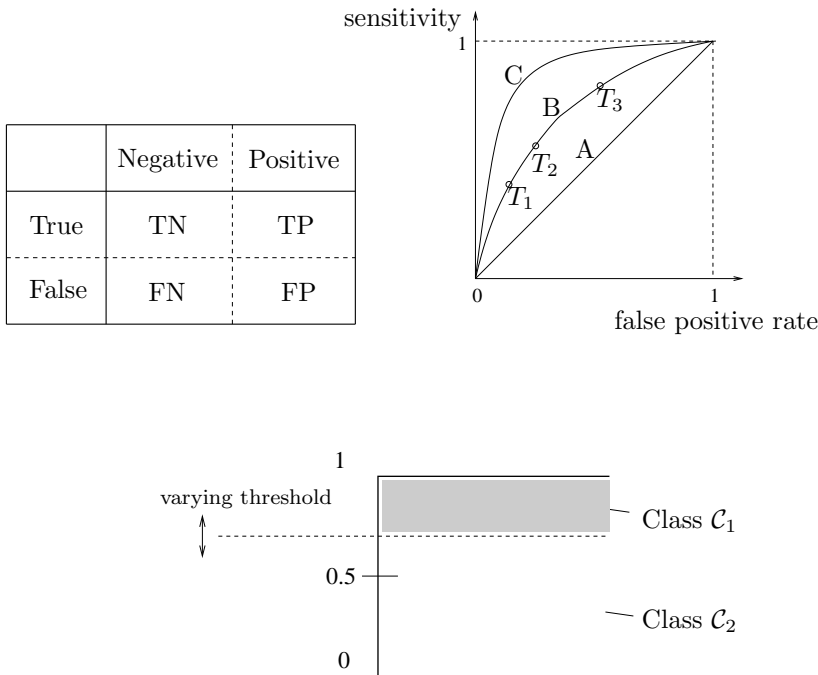


Fig. 1.10 (Top-Left) Confusion matrix; (Top-Right) ROC curve for three classifiers  $A, B, C$ . The classifier  $C$  has the best performance. It has the largest area under the ROC curve; (Bottom) An ROC curve is obtained by moving the threshold  $T$  of the classifier.

points on the ROC curve. These correspond to varying decision threshold values of an output unit. Note that in the case of logistic discrimination this corresponds in fact to varying the prior class probabilities, because the bias term explicitly depends on these probabilities.

## 1.5 Dimensionality reduction methods

In view of complexity criteria it is often useful to reduce the dimensionality of the input space. In this way a smaller amount of interconnection weights will be needed for the neural network.

A well-known and frequently used technique for dimensionality reduction is linear PCA analysis. Suppose one wants to map vectors  $x \in \mathbb{R}^n$

into lower dimensional vectors  $z \in \mathbb{R}^m$  with  $m < n$ . One proceeds then by estimating the covariance matrix:

$$\hat{\Sigma}_{\text{xx}} = \frac{1}{N-1} \sum_{k=1}^N (x_k - \bar{x})(x_k - \bar{x})^T \quad (1.31)$$

where  $\bar{x} = \frac{1}{N} \sum_{k=1}^N x_k$  and computes the eigenvalue decomposition

$$\hat{\Sigma}_{\text{xx}} u_i = \lambda_i u_i. \quad (1.32)$$

By selecting the  $m$  largest eigenvalues and the corresponding eigenvectors, one obtains the transformed variables (score variables)

$$z_i = u_i^T (x - \bar{x}), \quad i = 1, \dots, m. \quad (1.33)$$

One has to note, however, that these transformed variables are no longer real physical variables. The error  $\sum_{i=m+1}^n \lambda_i$  resulting from the dimensionality reduction is determined by the values of the neglected components.

Another possibility for dimensionality reduction, but in a supervised learning context, is the use of Automatic Relevance Determination (ARD) [151]. In this method one assigns additional regularization constants to each set of interconnection weights associated with a certain input. These additional hyperparameters are inferred at the second level of inference within the Bayesian learning context. In this case one finds the relevance of the original input variables itself instead of transformed variables in the PCA analysis case.

The case of linear PCA analysis can also be extended to nonlinear PCA analysis making use of neural nets [23]. Consider patterns  $x \in \mathbb{R}^n$  in the original space and transformed inputs  $z \in \mathbb{R}^m$  in a lower dimensional space with  $m \ll n$ . Conceptually this problem can be understood as an encoding/decoding problem to minimize the reconstruction error. For the encoder mapping  $z = G(x)$  and decoder mapping  $\hat{x} = F(z)$  one has the following objective of minimizing the squared distortion error between the original variables  $x \in \mathbb{R}^n$  and the reconstructed variables  $\hat{x} \in \mathbb{R}^n$

$$\begin{aligned} \min J &= \frac{1}{N} \sum_{k=1}^N \|x_k - \hat{x}_k\|_2^2 \\ &= \frac{1}{N} \sum_{k=1}^N \|x_k - F(G(x_k))\|_2^2 \end{aligned} \quad (1.34)$$

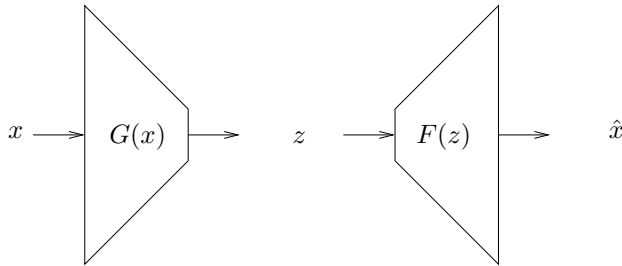


Fig. 1.11 *Information bottleneck in dimensionality reduction. Linear PCA is related to linear mappings  $F(\cdot), G(\cdot)$ . Nonlinear PCA analysis is obtained by considering nonlinear mappings that can be parameterized e.g. by MLPs.*

where the error measure is evaluated on a given training set of data points. An important special case is obtained when  $F(\cdot), G(\cdot)$  are linear mappings. It can be proven that this corresponds to linear PCA analysis. However, one can take these mappings also nonlinear and parameterize it e.g. by means of MLPs. In that case it leads to nonlinear PCA analysis. This insight gives an interesting interpretation of linear and nonlinear PCA analysis in terms of regression.

## 1.6 Parametric versus non-parametric approaches and RBF networks

Besides multilayer perceptrons another popular class of neural networks are Radial Basis Function (RBF) networks. While parametric regularization is important for reliable training of MLPs, the optimality of RBF networks can be understood in terms of non-parametric regularization. In [183] RBF networks have been derived from a viewpoint of functional analysis and calculus of variations by Poggio & Girosi, in relation to ill-posed problems [252]. Given a training data set  $\{x_k, y_k\}_{k=1}^N$  one considers the problem

$$\min_f J[f] = \sum_{k=1}^N (y_k - f(x_k))^2 + \nu \|Pf\|^2 \quad (1.35)$$

where  $\nu > 0$  denotes the regularization parameter,  $P$  is a differential operator and  $\|\cdot\|$  usually corresponds to the  $l_2$  norm defined on the function

space. Loosely speaking, this regularization term means that one keeps higher order derivatives small for the function to be estimated. One has shown that for a specific choice of the differential operator one obtains RBF networks as the optimal solution (for other choices one obtains e.g. spline networks). The optimal network model is given by

$$f^*(x) = \sum_{k=1}^N \alpha_k G(\|x - x_k\|) \quad (1.36)$$

where  $G(\cdot)$  denotes a Gaussian activation function. These are centered at each of the given points of the data set. The weighting coefficients follow from the solution to a linear system

$$(G + \nu I)\alpha = y \quad (1.37)$$

where  $y = [y_1; y_2; \dots; y_N]$ ,  $\alpha = [\alpha_1; \alpha_2; \dots; \alpha_N]$  and the  $kl$ -th entry of the matrix  $G$  equals  $G_{kl} = G(x_k, x_l) = G(\|x_k - x_l\|)$ . Note that the size of this system grows with the number of data. The form of the network is also related to methods in non-parametric statistics where one often employs kernel based models of the form

$$p(x) = \frac{1}{N} \sum_{k=1}^N \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{\|x - x_k\|_2^2}{2\sigma^2}\right) \quad (1.38)$$

especially in the context of density estimation, where a careful choice of the kernel width  $h$  should be made and no other parameters are to be determined.

RBF networks are inspired on (1.36) but are a *parameterized version* of it. One parameterizes a model into the form

$$f_{\text{rbf}}(x; w_i, c_i) = \sum_{i=1}^{n_h} w_i G(\|x - c_i\|) \quad (1.39)$$

where one takes a number of  $n_h$  hidden neurons. The unknown parameters are  $w_i \in \mathbb{R}$ ,  $c_i \in \mathbb{R}^n$  for  $i = 1, \dots, n_h$ . Note that the number of parameters to be estimated in this case is fixed and independent of the number of data which is not the case for (1.36), (1.37). The goal is then to minimize

$$\min_{w_i, c_i} J(w_i, c_i) = \sum_{k=1}^N (y_k - f_{\text{rbf}}(x_k; w_i, c_i))^2 + \nu \|P f_{\text{rbf}}(x_k; w_i, c_i)\|^2. \quad (1.40)$$

In the case of fixed centers the optimal solution is given by

$$w = (G^T G + \nu R)^{-1} G^T y \quad (1.41)$$

with  $G_{ki} = G(x_k, c_i)$  and  $R_{ij} = G(c_i, c_j)$ . Often, in a first stage, one tries to find a good choice of the centers by means of an unsupervised learning algorithm or cluster algorithm such as  $K$ -means and compute then the solution (1.41) for the output weights once the centers have been fixed. An alternative is to solve the entire nonlinear optimization problem in all unknowns  $w_i, c_i$  in a brute force way. One can also work with a weighted norm and consider the elements of the weighting matrix as additional unknowns to the optimization problem. Finally, one should note that in the case the centers  $c_i$  for  $i = 1, \dots, n_h$  coincide with the training data points  $x_k$  for  $k = 1, \dots, N$  one has  $R = G$  and  $G = G^T$ . One can then write  $w = (GG + \nu G)^{-1} G y = (G + \nu I)^{-1} y$  which equals (1.37). These insights clarify links and differences between the non-parametric (1.37) and parametric solution (1.41) and ridge regression.

## 1.7 Feedforward versus recurrent network models

When using neural networks in a dynamical systems context it is important to decide about the model structure. From a system and identification theory viewpoint [144; 216; 230] one has input/output (I/O) models and state space models. In the context of I/O models it is important to make a distinction between NARX (Nonlinear AutoRegressive with eXogenous input) and NOE (Nonlinear Output Error) models. In NARX models one has

$$\hat{y}_k = f(y_{k-1}, y_{k-2}, \dots, y_{k-q}, u_{k-1}, u_{k-2}, \dots, u_{k-q}) \quad (1.42)$$

where  $y_k$  denotes the true output at discrete time instant  $k$ ,  $u_k$  the input at time  $k$  and  $\hat{y}_k$  the estimated output at time  $k$ . The number  $q$  corresponds to the order of the system. In NOE models one has

$$\hat{y}_k = f(\hat{y}_{k-1}, \hat{y}_{k-2}, \dots, \hat{y}_{k-q}, u_{k-1}, u_{k-2}, \dots, u_{k-q}). \quad (1.43)$$

Note that one has a recursion now on the variable  $\hat{y}_k$  in contrast with the NARX model. From a neural networks perspective, the NARX model may be considered as a feedforward model, while the NOE model is *recurrent*.

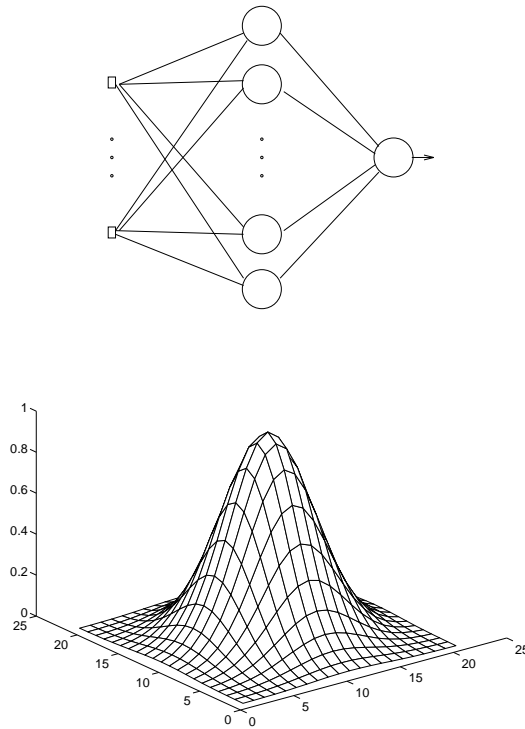


Fig. 1.12 Radial basis function network with Gaussian activation function.

The nonlinear function  $f(\cdot)$  is parameterized by means of a multilayer perceptron or an RBF network. The unknown parameters of the neural network are determined then by minimizing a suitable cost function for given training data.

Models for time-series prediction are closely related to these models by omitting the input variable  $u$ . One obtains then

$$\hat{y}_{k+1} = f(y_k, y_{k-1}, \dots, y_{k-q}) \quad (1.44)$$

which is parameterized by an MLP as

$$\hat{y}_{k+1} = w^T \tanh(V [y_k; y_{k-1}; \dots; y_{k-q}] + \beta). \quad (1.45)$$

It is not necessary that the past values  $y_k, y_{k-1}, \dots, y_{k-q}$  are subsequent in time; certain values could be omitted or values at different time scales could

be taken. In order to generate predictions, the true values  $y_k$  are replaced then by the estimated values  $\hat{y}_k$  and the iterative prediction is generated by the recurrent network

$$\hat{y}_{k+1} = w^T \tanh(V [\hat{y}_k; \hat{y}_{k-1}; \dots; \hat{y}_{k-q}] + \beta) \quad (1.46)$$

for a given initial condition.

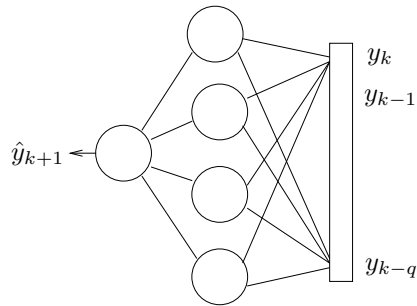
Instead of I/O models one may also take discrete time nonlinear state space descriptions

$$\begin{cases} \hat{x}_{k+1} &= f(\hat{x}_k, u_k) \\ \hat{y}_k &= g(\hat{x}_k) \end{cases} \quad (1.47)$$

where  $f(\cdot) : \mathbb{R}^n \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^n$  and  $g(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^{n_y}$  are nonlinear mappings. For the autonomous case (external inputs set to zero), (1.47) and (1.44) are related through Takens' embedding theorem. This states that by taking  $q \geq 2n$  one can reconstruct the state space from a single time series  $\{y_k\}$ . When one parameterizes these nonlinear functions by means of a feedforward neural network (such as MLP or RBF) one obtains a recurrent neural network.

Recurrent models are e.g. used in control applications, where one first identifies a model and then designs a controller based upon the identified model and applies it to the real system, either in a non-adaptive or adaptive setting. In [230] a stability theory has been developed for such neural control systems. When using neural networks in a dynamical systems context, one should be aware that even very simple recurrent networks can lead to complex behaviour such as chaos. In this sense stability issues of multi-layer recurrent networks are important e.g. towards applications in signal processing and control. The training of recurrent networks is also more complicated than for feedforward networks. In the recurrent network case a cost function is defined on a dynamical system (iterative system) which leads to more complicated analytic expressions for the gradient of the cost function. In order to compute the gradient one can simulate a sensitivity model together with the recurrent network model.

## Training mode



## Iterative prediction mode

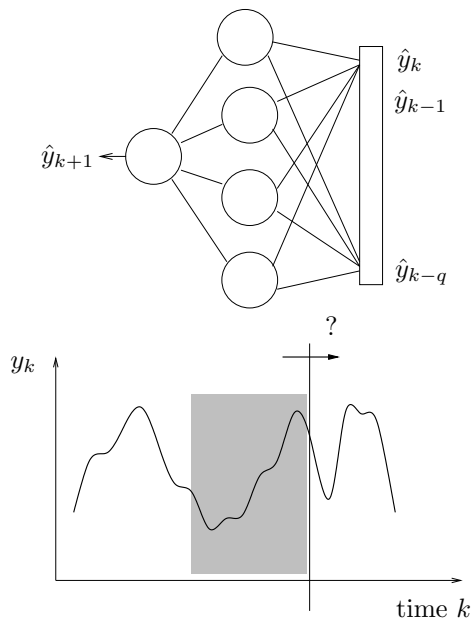


Fig. 1.13 Time-series prediction with neural networks: (Top) identification with a NARX model structure; (Bottom) iterative prediction as a recurrent network by replacing the true values  $y_k$  with the estimated values  $\hat{y}_k$  at the input of the network.