CHAPTER SEVEN

Foundations of Neural Networks

Contents

7.1. INTRODUCTION

Advances in clinical medical imaging have brought about the routine production of vast numbers of medical images that need to be analyzed. As a result, an enormous amount of computer vision research effort has been targeted at achieving automated medical image analysis. This has proved to be an elusive goal in many cases. The complexity of the problems encountered in analyzing these images has prompted considerable interest in the use of neural networks for such applications.

Artificial neural networks are an attempt to emulate the processing capabilities of biological neural systems. The basic idea is to realize systems capable of performing complex processing tasks by interconnecting a high number of very simple processing elements which might even work in parallel. They solve cumbersome and intractable problems by learning directly from data. An artificial neural network usually consists of a large amount of simple processing units, i.e., neurons, with mutual interconnections. It learns to solve problems by adequately adjusting the strength of the interconnections according to input data. Moreover, it can be easily adapted to new environments by learning. At the same time, it can deal with information that is noisy, inconsistent, vague, or probabilistic. These features motivate extensive research and developments in artificial neural networks.

The main features of artificial neural networks are their massive parallel processing architectures and the capabilities of learning from the presented inputs. They can be utilized to perform a specific task only by means of adequately adjusting the connection weights, i.e., by training them with the presented data. For each type of artificial neural network, there exists a corresponding learning algorithm by which we can train the network in an iterative updating manner. Those learning algorithms fit into two main categories: supervised learning and unsupervised learning.

For supervised learning, not only the input data but also the corresponding target answers are presented to the network. Learning is done by the direct comparison of the actual output of the network with known correct answers. This is also referred to as learning with a teacher. In contrast, if only input data without the corresponding target answers are presented to the network for learning, we have unsupervised learning. In fact, the learning goal is not defined at all in terms of specific correct examples. The available information is in the correlations of the input data. The network is expected to create categories from these correlations and to produce output signals corresponding to the input category.

Neural networks have been successfully employed to solve a variety of computer vision problems. They are systems of interconnected, simple processing elements. Generally speaking, a neural network aims to learn the nonlinear mapping from an input space describing the sensor information onto an output space describing the classes to which the inputs belong.

There are three basic mapping neural networks known in the literature [129]:

1. *Recurrent networks:* They are nonlinear, fully interconnected systems and form an autoassociative memory. Stored patterns correspond to stable states of a nonlinear system. The neural network is trained via a storage prescription that forces stable states to correspond to local minima of a network "energy" function. The memory capacity, or allowed number of stored patterns, is related to the network size. The Hopfield neural network [139] is the most popular recurrent neural network. A similar model is the bidirectional associative memory (BAM) [188].
2. *Universal feedforward neural networks:* They implement a nonlinear mapping between an input and output space and are nonlinear function approximators. Also, the network is able to extract higher-order statistics. The multilayer perceptron (MLP) [219], the backpropagation-type neural network [71], the radial basis neural network [255], and the support vector machine (SVM) [335] are the best-known universal feedforward neural networks.
3. *Local interaction-based neural networks:* These networks are based on competitive learning; the output neurons of the network compete among themselves to be activated. The output that wins the competition is called a winning neuron. One way of introducing competition among the output neurons is to use lateral inhibitory connections between them [314]. The local interaction is found in Kohonen maps [183], ART maps [117,118], and in the von der Malsburg model [76,390].

The classification in terms of architecture and learning paradigm of the neural networks described in this chapter is shown in Fig. 7.1.

There exist many types of neural networks that solve a wide range of problems in the area of image processing. There are also many types of neural networks, determined by

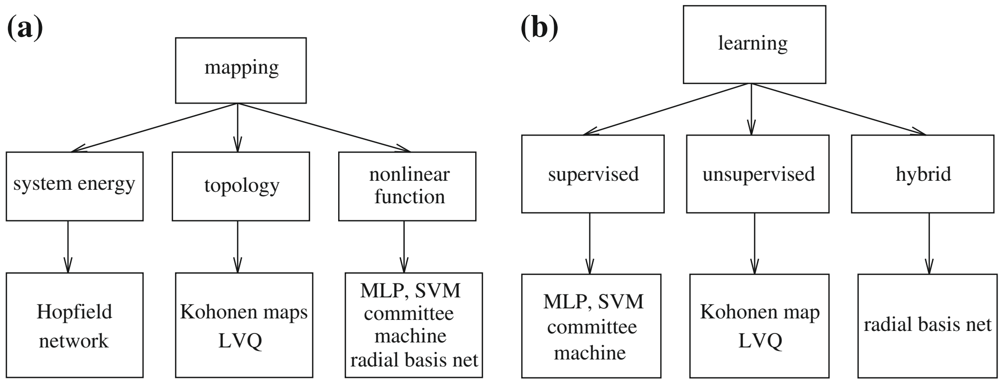

Figure 7.1 Classification of neural networks based on (a) architecture type and (b) learning algorithm.

the type of connectivity between the processing elements, the weights (synapses) of the connecting links, the processing elements' characteristics, and training or learning rules. These rules specify an initial set of weights and indicate how weights should be modified during the learning process to improve network performance.

The theory and representation of the various network types is motivated by the functionality and representation of biological neural networks. In this sense, processing units are usually referred to as neurons, while interconnections are called synaptic connections.

Although different neural models are known, all have the following basic components in common:

1. A finite set of neurons $x(1), x(2), \dots, x(n)$ with each neuron having a specific activity at time t, which is described by $x_t(i)$.
2. A finite set of neural connections $\mathbf{W} = (w_{ij})$, where w_{ij} describes the strength of the connection of neuron $x(i)$ with neuron $x(j)$.
3. A propagation rule $\tau_t(i) = \sum_{j=1}^{n} x_t(j) w_{ij}$.
4. An activation function f, which has τ as an input value and produces the next state of the neuron $x_{t+1}(i) = f(\tau_t(i) - \theta)$, where θ is a threshold and f is a nonlinear function such as a hard limiter, threshold logic, or sigmoidal function.

7.2. MULTILAYER PERCEPTRON (MLP)

Multilayer perceptrons are one of the most important types of neural nets because many applications are successful implementations of MLPs. The architecture of the MLP is completely defined by an input layer, one or more hidden layers, and an output layer. Each layer consists of at least one neuron. The input vector is processed by the MLP in a forward direction, passing through each single layer. Figure 7.2 illustrates the configuration of the MLP.

A neuron in a hidden layer is connected to every neuron in the layers above and below it. In Fig. 7.2 weight w_{ij} connects input neuron x_i to hidden neuron h_j and weight v_{jk} connects h_j to output neuron o_k. Classification starts by assigning the input neurons x_i, $1 \leq i \leq l$, equal to the corresponding data vector component. Then data propagates in a forward direction through the perceptron until the output neurons o_k, $1 \leq k \leq n$, are reached. Assuming that the output neurons are equal to either 0 or 1, then the perceptron is capable of partitioning its pattern space into 2^n classes.

Before employing any pattern recognition system, we need first to collect the sensor signals that form the input data. Then the data are normalized and filtered in order to remove the noise component. Features represent only the relevant data information that needs to be extracted from the available sensor information. The subsequent classification assigns the resulting feature vector to one of the given classes. While most classification techniques operate based on an underlying statistical model, neural networks estimate the necessary discriminant functions. Mathematically, the multilayer perceptron performs a nonlinear approximation using sigmoidal kernel functions as hidden units and linear

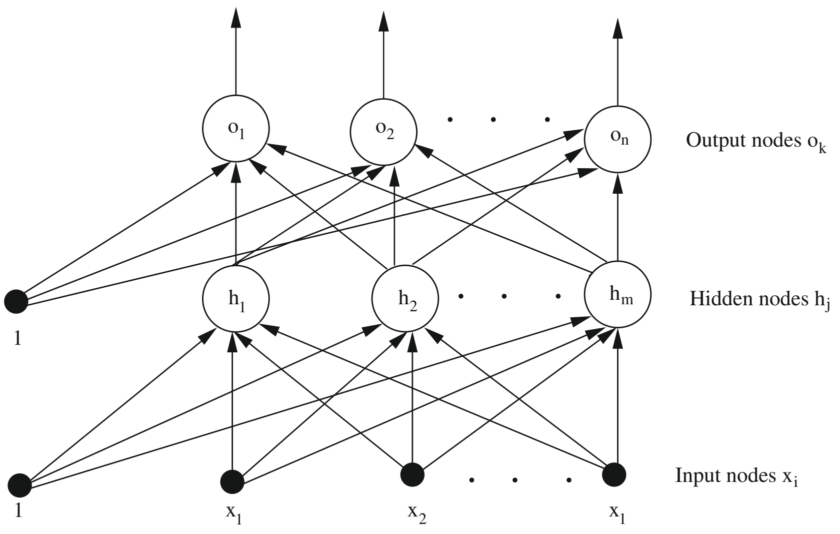

Figure 7.2 Two-layer perceptron.

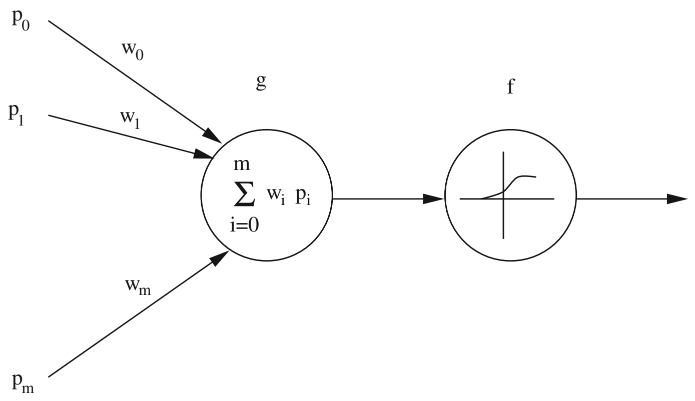

Figure 7.3 Propagation rule and activation function for the MLP network.

weights. During the learning of feature vectors, the weights are continuously adapted by minimizing the error between desired outputs and the computed network's outputs.

The steps that govern the data flow through the perceptron during classification are [310]:

1. Present the pattern $\mathbf{p} = [p_1, p_2, \ldots, p_l] \in \mathbf{R}^\mathbf{l}$ to the perceptron, that is, set $x_i = p_i$ for $1 \leq i \leq l$.
2. Compute the values of the hidden-layer neurons as illustrated in Fig. 7.3.

$$h_j = \frac{1}{1 + \exp\left[-\left(w_{0j} + \sum_{i=1}^{l} w_{ij} x_i\right)\right]}; \quad 1 \leq j \leq m \tag{7.1}$$

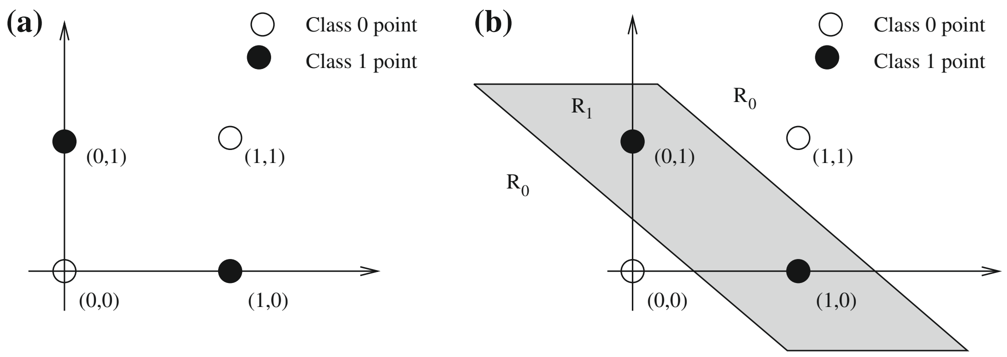

Figure 7.4 XOR problem and solution strategy using the MLP.

The activation function of all units in the MLP is given by the sigmoid function $f(x) = \frac{1}{1+\exp(-x)}$ and is the standard activation function in feedforward neural networks. It is defined as a monotonic increasing function representing an approximation between nonlinear and linear behavior.

3. Calculate the values of the output neurons based on

$$o_k = \frac{1}{1 + \exp\left(v_{0k} + \sum_{j=1}^{m} v_{jk} h_j\right)}; \quad 1 \le k \le n \tag{7.2}$$

4. The class $\mathbf{c} = [c_1, c_2, \ldots, c_n]$ that the perceptron assigns $\mathbf{p}$ must be a binary vector. So o_k must be the threshold of a certain class at some level τ and depends on the application.

5. Repeat steps 1,2,3, and 4 for each given input pattern.

Multilayer perceptrons are highly nonlinear interconnected systems and serve for both nonlinear function approximation and nonlinear classification tasks. A typical classification problem that can be solved only by the MLP is the XOR problem. Based on a linear classification rule, R^m can be partitioned into regions separated by a hyperplane. On the other hand, the MLP is able to construct very complex decision boundaries as depicted in Fig. 7.4.

Applied to image processing the MLP has as input features extracted from images or from regions of these images. Such features can be shape, size, or texture measures and attempt to capture the key aspects of an image.

Multilayer perceptrons have been applied to a variety of problems in image processing, including optical character recognition [320] and medical diagnosis [77,78].

7.2.1 Backpropagation-Type Neural Networks

Multilayer perceptrons (MLPs) have been applied successfully to sophisticated classification problems. The training of the network is accomplished based on a supervised learning technique that requires given input-output data pairs. The training technique, known as the error backpropagation algorithm, is bidirectional consisting of a forward and backward direction. During the forward direction a training vector is presented to the network and classified. The backward direction consists of a recursive updating of the weights in all layers based on the computed errors. The error terms of the output layer are a function of $\mathbf{c}^t$ and output of the perceptron $(o_1, o_2, \ldots, o_n)$.

The algorithmic description of the backpropagation is as follows [71]:

1. **Initialization:** Initialize the weights of the perceptron randomly with numbers between -0.1 and 0.1; that is,

$$\begin{aligned} w_{ij} &= \text{random}([-0.1, 0.1]), \quad 0 \le i \le l, \quad 1 \le j \le m \\ v_{jk} &= \text{random}([-0.1, 0.1]), \quad 0 \le j \le m, \quad 1 \le k \le n \end{aligned} \tag{7.3}$$

2. **Presentation of training patterns:** Present $\mathbf{p}^t = [p_1^t, p_2^t, \ldots, p_l^t]$ from the training pair $(\mathbf{p}^t, \mathbf{c}^t)$ to the perceptron and apply steps 1, 2, and 3 from the perceptron classification algorithm previously described.
3. **Forward computation (output layer):** Calculate the errors δ_{ok}, $1 \le k \le n$, in the output layer using

$$\delta_{ok} = o_k(1 - o_k)\left(c_k^t - o_k\right) \tag{7.4}$$

where $\mathbf{c}^t = [c_1^t, c_2^t, \ldots, c_n^t]$ represents the correct class of $\mathbf{p}^t$. The vector $(o_1, o_2, \ldots, o_n)$ describes the output of the perceptron.
4. **Forward computation (hidden layer):** Calculate the errors δ_{hj}, $1 \le j \le m$, in the hidden-layer neurons based on

$$\delta_{hj} = h_j(1 - h_j) \sum_{k=1}^{n} \delta_{ok} v_{jk} \tag{7.5}$$

5. **Backward computation (output layer):** Let v_{jk} denote the value of weight v_{jk} after the tth training pattern has been presented to the perceptron. Adjust the weights between the output layer and the hidden layer based on

$$v_{jk}(t) = v_{jk}(t-1) + \eta \delta_{ok} h_j \tag{7.6}$$

The parameter $0 \le \eta \le 1$ represents the learning rate.
6. **Backward computation (hidden layer):** Adjust the weights between the hidden layer and the input layer using

$$w_{ij}(t) = w_{ij}(t-1) + \eta \delta_{hj} p_i^t \tag{7.7}$$

7. **Iteration:** Repeat steps 2–6 for each pattern vector of the training data. One cycle through the training set defines an iteration.

7.2.2 Design Considerations

MLPs are global approximators and can be trained to implement any given nonlinear input-output mapping. In a subsequent testing phase, they prove their interpolation ability by generalizing even in sparse data space regions.

When designing a neural network, specifically deciding for a fixed architecture, performance and computational complexity considerations play a crucial role. Mathematically, it has been proved [126] that even one hidden-layer MLP is able to approximate the mapping of any continuous function.

As with all neural networks, the dimension of the input vector dictates the number of neurons in the input layer, while the number of classes to be learned dictates the number of neurons in the output layer. The number of chosen hidden layers and the number of neurons in each layer have to be empirically determined. As a rule of thumb, the neurons in the hidden layers are chosen as a fraction of those in the input layer. However, there is a trade-off regarding the number of neurons: Too many produce overtraining; too few affect generalization capabilities. Usually, after satisfactory learning is achieved, the neurons' number can be gradually reduced. Then subsequent retraining of a reduced-size network exhibits much better performance than the initial training of the more complex network.

Great care must be taken when selecting the training data set. It has to be representative of all data classes to allow a good generalization. During training, input vectors are presented in random order so that the network achieves a global learning and not class-selective learning.

7.2.3 Complexity of the Multilayer Perceptron

The complexity of a classifier is solely described by its adjustable parameters. Thus, the MLP's adaptable parameters are the number of weights and biases. Theoretically, there is always an optimal complexity for a given classification problem. If this optimal complexity for an MLP is exceeded, then the classification outcome becomes prone to the intrinsic data noise. Although the MLP learns all training examples very well, it fails to correctly recognize new feature vectors it did not learn. This effect is called overfitting and always occurs when the number of weights exceeds the number of input patterns. Underfitting occurs when we have a lower complexity than the optimum, and thus an insufficient discrimination capability. In other words, each classification problem requires a certain optimum MLP complexity, and thus complexity control becomes an issue.

Complexity control can be obtained by the following techniques:

- *Data preprocessing:* Feature extraction, feature selection, feature dimensionality reduction, as shown in Chapter 2.
- *Training scheme:* Early stopping, cross-validation.
- *Network structure:* Committee machines.

7.2.3.1 Cross-Validation

The simplest method to avoid overfitting is to provide more training samples so that the MLP cannot model the noise of the training data set. In practical applications, it may happen that there is a shortage of labeled data available.

A solution to this dilemma poses a technique from statistics known as cross-validation [352]. The MLP is trained on a disjoint subset of the input data, and then a cross-validation subset of the data is used. Thus, the MLP is trained on the training set and then the classification error is checked on the validation set. This is repeated, and then the training is stopped when the cross-validation measure given by the classification errors stops improving. This means that continued training results in overtraining on the training data subset of input data, and this implies that the MLP loses its generalization ability on the cross-validation set.

This procedure proves to be extremely useful when we want to achieve a good generalization (optimal number of hidden neurons) and when we want to decide when the training of the MLP has to be stopped.

7.2.3.2 The Process of Training and Validation

The goal of training a neural network is to achieve a good generalization. However, it is difficult to decide when it is best to stop training without ending up with overfitting. Overfitting can be easily detected by using cross-validation.

In general, the process of achieving good generalization with an MLP consists of three parts: (1) training, (2) validation, and (3) verification. This means, that we need three disjoint sets of input data: a training set that is the largest set, a validation set, and a verification set or test set. Validation has to be performed after cycles of several iterations during training to check when the classification error stops decreasing. The classification error is an ideal sensitivity measure in this respect since it deteriorates when the training of the MLP is too specialized on the training set. The advantage of this method is that we can stop the training before the classification error starts degrading and choose the respective MLP configuration as its final. Hence the name "early stopping." Figure 7.5

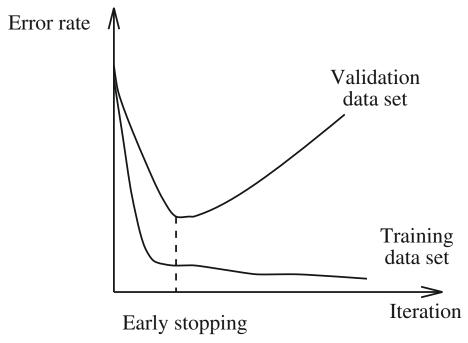

Figure 7.5 Early stopping method of training.

visualizes this technique. The validation subset is small compared to the training set. The final step represents the testing phase when the trained MLP is verified to demonstrate that it is indeed implementing a correct classification mechanism. Usually, the test set is half the size of the training set.

7.2.3.3 Committee Machines

Complex classification problems in practice require the contribution of several neural networks for achieving an optimal solution. Thus a complex task is split into smaller ones whose contributions are necessary to provide the global complex solution. In neural network terminology, it means to allocate the learning task among a number of experts, which in turn split the input data into a set of subspaces. This idea leads to the concept of the committee machine, which is based on a combination of concurrent experts, and thus implements the famous principle of divide and conquer. In this way, it combines the knowledge of every single expert and achieves a global decision that is superior to that achieved by a single expert. The idea of the committee machine was first introduced by Nilsson [262].

Committee machines fall into the category of universal approximators and can have either a static or a dynamic structure. In the dynamic structures, the input signal directly influences the mechanism that fuses the output decisions of the individual experts into an overall output decision. There are two groups of dynamic structures: mixture of experts and hierarchical mixture of experts.

- *Mixture of experts:* A single gating network represents a nonlinear function of the individual responses of the experts, and thus the divide-and-conquer algorithm is applied only once.
- *Hierarchical mixture of experts:* A hierarchy of several gating networks forms a nonlinear function of the individual responses of the experts, and thus the divide and conquer algorithm is applied several times at the corresponding hierarchy levels.

Figure 7.6 shows a hierarchical mixture of experts [157]. The architecture is a tree in which the gating networks sit at the nonterminals of the tree. These networks receive the vector $\mathbf{x}$ as input and produce scalar outputs that are a partition of unity at each point in the input space. The expert networks sit at the leaves of the tree. They are linear with the exception of a single output nonlinearity. Expert network (i, j) produces its output μ_{ij} as a generalized function of the input vector $\mathbf{x}$ and a weight matrix $\mathbf{U}_{ij}$

$$\mu_{ij} = \mathbf{U}_{ij}\mathbf{x} \tag{7.8}$$

The neurons of the gating networks are nonlinear.

Let ξ_i be an intermediate variable

$$\xi_i = \mathbf{v}_i^T\mathbf{x} \tag{7.9}$$

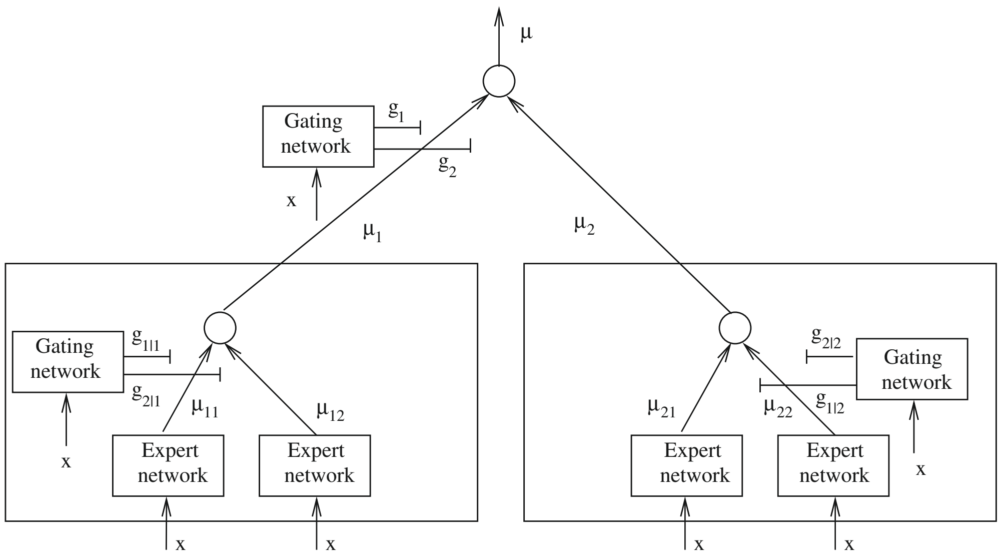

Figure 7.6 Hierarchical mixture of experts.

$\mathbf{v}_i$ is a weight vector. Then the ith output of the top-level gating network is the "softmax" function of ξ_i given as

$$g_i = \frac{\exp(\xi_i)}{\sum_k \exp(\xi_k)} \tag{7.10}$$

Note that $g_i > 0$ and $\sum_i g_i = 1$. The g_i values can be interpreted as providing a "soft" partitioning of the input space.

Similarly, the neurons of the gating networks at lower levels are also nonlinear. Define ξ_{ij} as

$$\xi_{ij} = \mathbf{v}_{ij}^T \mathbf{x} \tag{7.11}$$

Then

$$g_{i|j} = \frac{\exp(\xi_{ij})}{\sum_k \exp(\xi_{ik})} \tag{7.12}$$

is the output of the jth unit in the ith gating network at the second level of the architecture.

The output vector at each nonterminal of the tree is the weighted output of the experts below the nonterminal. That is, the output at the ith nonterminal in the second layer is

$$\mu_i = \sum_j g_{j|i}\, \mu_{ij} \tag{7.13}$$

and the output at the top level is

$$\mu = \sum_i g_i \mu_i \tag{7.14}$$

Both g and μ depend on the input $\mathbf{x}$; thus the total output is a nonlinear function of the input.

7.3. SELF-ORGANIZING NEURAL NETWORKS

A self-organizing map is implemented by a 1-D or 2-D lattice of neurons. The neurons become specialized to various input patterns or classes of input patterns while the network performs an unsupervised competitive learning process. The weights of the neurons that have close locations on the lattice are trained to represent similar input vectors. We obtain a network that preserves neighborhood relations in the input space, and thus preserves the topology of the input space. This map is known as a self-organizing feature map [184].

7.3.1 Self-Organizing Feature Map

Mathematically, the self-organizing map (SOM) determines a transformation from a high-dimensional input space onto a one- or two-dimensional discrete map. The transformation takes place as an adaptive learning process such that when it converges the lattice represents a topographic map of the input patterns. The training of the SOM is based on a random presentation of several input vectors, one at a time. Typically, each input vector produces the firing of one selected neighboring group of neurons whose weights are close to this input vector.

The most important components of such a network are:

1. A 1-D or 2-D lattice of neurons that encodes an input pattern of an arbitrary dimension into a specific location on it, as shown in Fig. 7.7a.
2. A method that selects a "winning neuron" based on the similarity between the weight vector and the input vector.
3. An interactive network that activates the winner and its neighbors simultaneously. A neighborhood $\Lambda_{i(\mathbf{x})}(n)$ which is centered on the winning neuron is a function of the discrete time n. Figure 7.7b illustrates such a neighborhood, which first includes the whole array and then shrinks gradually to only one "winning neuron," represented by the black circle.
4. An adaptive learning process that reinforces all neurons in the close neighborhood of the winning neuron, and inhibits all those that are farther away from the winner.

The learning algorithm of the self-organized map is simple and is described below:

1. **Initialization:** Choose random values for the initial weight vectors $\mathbf{w}_j(0)$ to be different for $j = 1, 2, \ldots, N$, where N is the number of neurons in the lattice. The magnitude of the weights should be small.

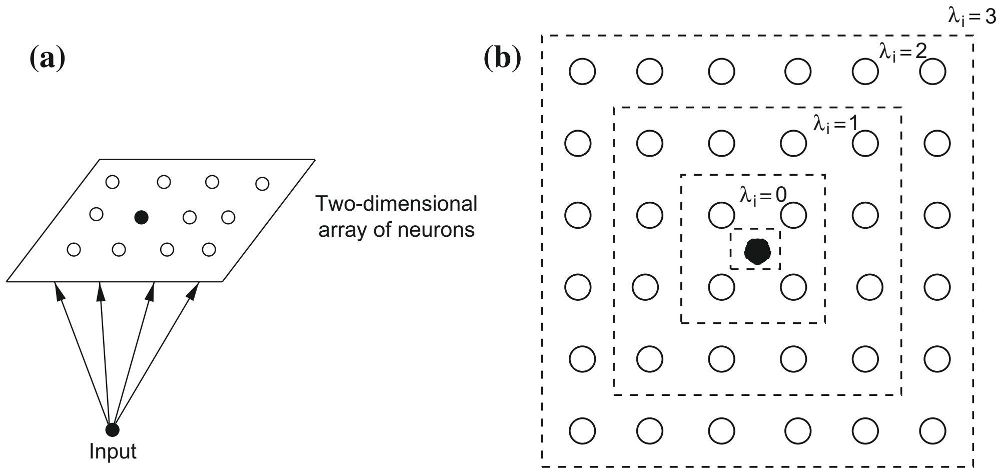

Figure 7.7 (a) Kohonen neural network; and (b) neighborhood Λ_i, of varying size, around "winning" neuron *i*, identified as a black circle.

2. **Sampling:** Draw a sample $\mathbf{x}$ from the input data; the vector $\mathbf{x}$ represents the new pattern that is presented to the lattice.
3. **Similarity matching:** Find the "winner neuron" $i(\mathbf{x})$ at time n based on the minimum distance Euclidean criterion:

$$i(\mathbf{x}) = \arg\min_j ||\mathbf{x}(n) - \mathbf{w}_j(n)||, \quad j = 1, 2, \ldots, N \tag{7.15}$$

4. **Adaptation:** Adjust the synaptic weight vectors of all neurons (winners or not), using the update equation

$$\mathbf{w}_j(n+1) = \begin{cases} \mathbf{w}_j(n) + \eta(n)[\mathbf{x}(n) - \mathbf{w}_j(n)], & j \in \Lambda_{i(\mathbf{x})}(n) \\ \mathbf{w}_j(n), & \text{else} \end{cases} \tag{7.16}$$

   where $\eta(n)$ is the learning-rate parameter, and $\Lambda_{i(\mathbf{x})}(n)$ is the *neighborhood function* centered around the winning neuron $i(\mathbf{x})$; both $\eta(n)$ and $\Lambda_{i(\mathbf{x})}$ are a function of the discrete time n and thus continuously adapted for optimal learning.
5. **Continuation:** Go to step 2 until there are no noticeable changes in the feature map.

The presented learning algorithm has some interesting properties, which are described based on Fig. 7.8.

Mathematically, the feature map represents a nonlinear transformation Φ from a continuous input space X to a spatially discrete output space A:

$$\Phi : X \rightarrow A \tag{7.17}$$

The map preserves the topological relationship that exists in the input space, if the input space is of lower or equal dimensionality compared to the output space. In all other cases,

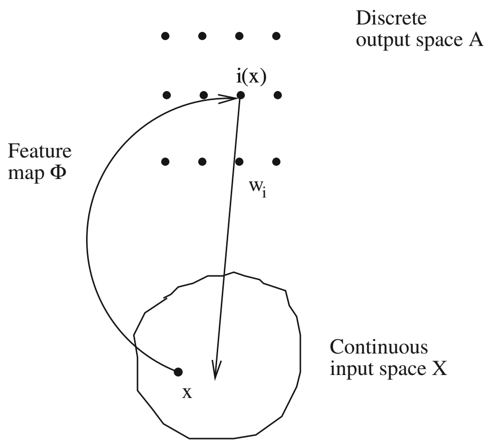

Figure 7.8 Mapping between input space *X* and output space *A*.

the map is said to be neighborhood preserving, in the sense that neighboring regions of the input space activate neurons that are adjacent on the lattice. Thus, in cases when the topological structure of the input space is not known a priori or is too complicated to be specified, Kohonen's algorithm necessarily fails in providing perfectly topology-preserving maps.

In the following, we will give some basic properties of self-organizing maps. The first property pertains to an approximation of the input space. The self-organizing feature map Φ, completely determined by the neural lattice, learns the input data distribution by adjusting its synaptic weight vectors $\{\mathbf{w}_j | j = 1, 2, \ldots, N\}$ to provide a good approximation to the input space X.

The second property refers to the topological ordering achieved by the nonlinear feature map: There is a correspondence between the location of a neuron on the lattice and a certain domain or distinctive feature of the input space.

Kohonen maps have been applied to a variety of problems in image processing, including texture segmentation [273] and medical diagnosis [190].

7.3.1.1 Design Considerations

A successful application of the Kohonen neural networks is highly dependent on the main parameters of the algorithm, namely, the learning-rate parameter η and the neighborhood function Λ_i. Since there is no theoretical foundation for the choice of these parameters, we have to rely on practical hints [183]: The learning-rate parameter $\eta(n)$ employed for adaptation of the synaptic vector $\mathbf{w}_j(n)$ should be time-varying. For the first 100 iterations $\eta(n)$ it should stay close to unity and decrease slowly thereafter, but remain above 0.1.

The neighborhood function Λ_i has to be wisely selected in order to ensure topological ordering of the weight vectors $\mathbf{w}_j$. Λ_i can have any geometrical form but should always include the winning neuron in the middle. In the beginning, the neighborhood function Λ_i includes all neurons in the network and then slowly shrinks with time. It takes in most cases about 1000 iterations for the radius of the neighborhood function Λ_i to shrink linearly with time n to a small value of only a couple of neighboring neurons.

7.3.2 Learning Vector Quantization

Vector quantization (VQ) provides an efficient technique for data compression. Compression is achieved by transmitting the index of the codeword instead of the vector itself.

VQ can be defined as a mapping that assigns each vector $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})^T$ in the n-dimensional space R^n to a codeword from a finite subset of R^n. The subset $\mathbf{Y} = \{\mathbf{y}_i : i = 1, 2, \ldots, M\}$ representing the set of possible reconstruction vectors is called a codebook of size M. Its members are called the codewords. In the encoding process, a distance measure is evaluated to locate the closest codeword for each input vector $\mathbf{x}$. Then, the address corresponding to the codeword is assigned to $\mathbf{x}$ and transmitted. The distortion between the input vector and its corresponding codeword $\mathbf{y}$ is defined by the distance, $d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||$, where $||\mathbf{x}||$ represents the norm of $\mathbf{x}$.

A vector quantizer achieving a minimum encoding error is referred to as a Voronoi quantizer. Figure 7.9 shows an input data space partitioned into four different regions, called Voronoi cells, and the corresponding Voronoi vectors. These regions describe the collection of only those input vectors that are very close to the respective Voronoi vector.

Recent developments in neural network architectures have led to a new VQ concept, the so-called learning vector quantization (LVQ). It represents an unsupervised learning algorithm associated with a competitive neural network consisting of one input and one output layer. The algorithm permits only the update of the winning prototype, that is, the

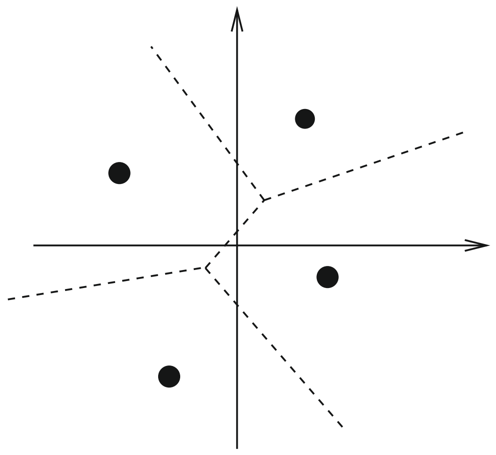

Figure 7.9 Voronoi diagram involving four cells. The small circles indicate the Voronoi vectors and are the different region (class) representatives.

closest prototype (Voronoi vector) of the LVQ network. If the class of the input vector and the Voronoi vector match, then the Voronoi vector is moved in the direction of the input vector $\mathbf{x}$. Otherwise the Voronoi vector $\mathbf{w}$ is moved away from this vector $\mathbf{x}$.

The LVQ algorithm is simple:

1. **Initialization:** Initialize the weight vectors $\{\mathbf{w}_j(0)|j = 1, 2, \ldots, N\}$ by setting them equal to the first N exemplar input feature vectors $\{\mathbf{x}_i|i = 1, 2, \ldots, L\}$.
2. **Sampling:** Draw a sample $\mathbf{x}$ from the input data; the vector $\mathbf{x}$ represents the new pattern that is presented to the LVQ.
3. **Similarity Matching:** Find the best matching codeword (Voronoi vector) $\mathbf{w}_j$ at time n based on the minimum distance Euclidean criterion:
$$\arg\min_j ||\mathbf{x}(n) - \mathbf{w}_j(n)||, \quad j = 1, 2, \ldots, N \tag{7.18}$$
4. **Adaptation:** Adjust only the best matching Voronoi vector, while the others remain unchanged. Assume that a Voronoi vector $\mathbf{w}_c$ is the closest to the input vector $\mathbf{x}_i$. By $C_{\mathbf{w}_c}$ we define the class associated with the Voronoi vector $\mathbf{w}_c$, and by $C_{\mathbf{x}_i}$ the class label associated to the input vector $\mathbf{x}_i$. The Voronoi vector $\mathbf{w}_c$ is adapted as follows:
$$\mathbf{w}_c(n+1) = \begin{cases} \mathbf{w}_c(n) + \alpha_n[\mathbf{x}_i - \mathbf{w}_c(n)], & C_{\mathbf{w}_c} = C_{\mathbf{x}_i} \\ \mathbf{w}_c(n) - \alpha_n[\mathbf{x}_i - \mathbf{w}_c(n)], & \text{otherwise} \end{cases} \tag{7.19}$$
where $0 < \alpha_n < 1$.
5. **Continuation:** Go to step 2 until there are no noticeable changes in the feature map.

The learning constant α_n is chosen as a function of the discrete time parameter n and decreases monotonically. The relative simplicity of the LVQ and its ability to work in unsupervised mode have made it a useful tool for image segmentation problems [190].

7.3.2.1 The "Neural-Gas" Algorithm

The "neural-gas" algorithm [236] is an efficient approach which, applied to the task of vector quantization, (1) converges quickly to low distortion errors, (2) reaches a distortion error E lower than that from Kohonen's feature map, and (3) at the same time obeys a gradient descent on an energy surface.

Instead of using the distance $||\mathbf{x} - \mathbf{w}_j||$ or the arrangement of the $||\mathbf{w}_j||$ within an external lattice, it utilizes a neighborhood ranking of the reference vectors $\mathbf{w}_i$ for the given data vector $\mathbf{x}$. The adaptation of the reference vectors is given by
$$\Delta\mathbf{w}_i = \epsilon e^{-k_i(\mathbf{x},\mathbf{w}_i/\lambda)}(\mathbf{x} - \mathbf{w}_i) \quad i = 1, \ldots, N \tag{7.20}$$
where N is the number of units in the network. The step size $\epsilon \in [0, 1]$ describes the overall extent of the modification, and k_i is the number of the closest neighbors of the reference vector $\mathbf{w}_i$. λ is a characteristic decay constant.

In [236], it was shown that the average change of the reference vectors can be interpreted as an overdamped motion of particles in a potential that is given by the negative data point density. Added to the gradient of this potential is a "force" in the direction of the space where the particle density is low. This "force" is based on a repulsive coupling between the particles (reference vectors). In form it is similar to an entropic force and tends to uniformly distribute the particles (reference vectors) over the input space, as is the case with a diffusing gas. Hence the name "neural-gas" algorithm. It is also interesting to mention that the reference vectors are slowly adapted, and therefore pointers that are spatially close at an early stage of the adaptation procedure might not be spatially close later. Connections that have not been updated for a while die out and are removed.

Another important feature of the algorithm compared to the Kohonen algorithm is that it does not require a prespecified graph (network). In addition, it can produce topology-preserving maps, which is only possible if the topological structure of the graph matches the topological structure of the data manifold. In cases, however, where an appropriate graph cannot be determined from the beginning, such as where the topological structure of the data manifold is not known in advance or is too complex to be specified, Kohonen's algorithm always fails in providing perfectly topology-preserving maps.

To obtain perfectly topology-preserving maps we employ a powerful structure from computational geometry: the Delaunay triangulation, which is the dual of the already mentioned Voronoi diagram [302]. In a plane, the Delaunay triangulation is obtained if we connect all pairs $\mathbf{w}_j$ by an edge if their Voronoi polyhedra are adjacent. Figure 7.10 shows an example of a Delaunay triangulation. The Delaunay triangulation arises as a graph matching to the given pattern manifold.

The neural gas algorithm is simple:

1. **Initialization:** Randomly initialize the weight vectors $\{\mathbf{w}_j | j = 1, 2, \ldots, N\}$ and the training parameters $(\lambda_i, \lambda_f, \epsilon_i, \epsilon_f)$, where λ_i, ϵ_i are initial values of $\lambda(t), \epsilon(t)$ and λ_f, ϵ_f are the corresponding final values.
2. **Sampling:** Draw a sample $\mathbf{x}$ from the input data; the vector $\mathbf{x}$ represents the new pattern that is presented to the neural gas network.

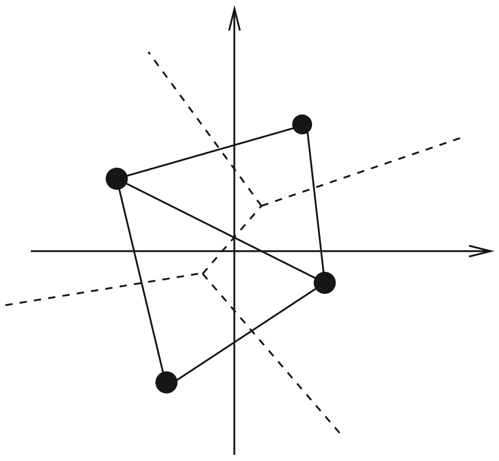

Figure 7.10 Delaunay triangulation.

3. **Distortion:** Determine the distortion set $D_{\mathbf{x}}$ between the input vector $\mathbf{x}$ and the weights $\mathbf{w}_j$ at time n based on the minimum distance Euclidean criterion:

$$D_{\mathbf{x}} = ||\mathbf{x}(n) - \mathbf{w}_j(n)||, \quad j = 1, 2, \ldots, N \tag{7.21}$$

   Then order the distortion set in ascending order.
4. **Adaptation:** Adjust the weight vectors according to

$$\Delta \mathbf{w}_i = \epsilon e^{-k_i(\mathbf{x}, \mathbf{w}_i/\lambda)}(\mathbf{x} - \mathbf{w}_i) \quad i = 1, \ldots, N \tag{7.22}$$

   where $i = 1, \ldots, N$. The parameters have the following time dependencies: $\lambda(t) = \lambda_i(\lambda_f/\lambda_i)^{\frac{t}{t_{max}}}$ and $\epsilon(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{\frac{t}{t_{max}}}$.
   Increment the time parameter t by 1.
5. **Continuation:** Go to step 2 until the maximum iteration number t_{max} is reached.

7.4. RADIAL BASIS NEURAL NETWORKS (RBNN)

Radial basis neural networks describe a new concept of classification, a so-called hybrid learning mechanism. Their architecture is quite simple, based on a three-layer model having only one hidden layer. The hybrid learning mechanism combines a self-organized learning of the hidden layers' neurons with a supervised learning applied to the output layer's weight adaptation.

The design of a neural network can be viewed as a curve-fitting (nonlinear approximation) problem in a high-dimensional space. Thus, learning is equivalent to finding the surface in a multidimensional space that provides the best match to the training data. To be specific, let us consider a system with n inputs and m outputs and let $\{x_1, \ldots, x_n\}$ be an input vector and $\{y_1, \ldots, y_m\}$ the corresponding output vector describing the system's answer to that specific input. After the training process is completed, the system has learned the input and output data distribution and is able to find for any input the correct output. In other words, we look for the "best" approximation function $\widehat{f}(x_1, \ldots, x_n)$ of the actual input-output mapping function [84,296].

In the following, we will describe the mathematical background that is necessary to understand radial basis neural networks. We will review the concept of interpolation and show how the interpolation problem is implemented by a radial basis neural network. Since there are an infinite number of solutions for a given approximation problem, we want to choose one particular solution, and therefore we have to impose some restrictions on the solutions. This leads to a regularization approach to the approximation problem and can be easily implemented by the radial basis neural network.

7.4.1 Interpolation Networks

The interpolation problem can be very elegantly solved by a three-layer feedforward neural network. The layers play entirely different roles in the learning process. The input layer is made up of source nodes that receive sensor information. The second layer, the only hidden layer, applies a nonlinear transformation from the input space to the hidden space. The nonlinear transformation is followed by a linear transformation from the hidden layer to the output layer. This leads to the theory of multivariable interpolation in the high-dimensional feature space.

The mathematical formulation of the interpolation problem is given below.

The Interpolation Prolem 7.4.1 *Assume that to N different points $\{\mathbf{m}_i \in \mathcal{R}^n | i = 1, \ldots N\}$ correspond N real numbers $\{d_i \in \mathcal{R} | i = 1, \ldots, N\}$. Then find a function $F : \mathcal{R}^n \to \mathcal{R}$ that satisfies the interpolation condition:*

$$F(\mathbf{m}_i) = d_i \quad \text{for } i = 1, \ldots, N \tag{7.23}$$

The radial basis function technique consists of choosing a function F that has the following form [296]:

$$F(\mathbf{x}) = \sum_{i=1}^{N} c_i h(||\mathbf{x} - \mathbf{m}_i||) + \sum_{i=1}^{M} d_i p_i(\mathbf{x}), \quad M \leq N \tag{7.24}$$

h is a smooth function, known as a radial basis function; $||.||$ is the Euclidian norm in $\mathcal{R}^n$; $\{p_i | i = 1, \ldots, m\} : \mathcal{R}^n \to \mathcal{R}$ represents a basis of algebraic polynomials in the linear space $\pi_{k-1}(\mathcal{R}^n)$ of maximum order $k - 1$ for a given k.

Let the radial basis function $h(r)$ be continuous on $[0, \infty)$ and its derivatives on $[0, \infty)$ strict monotone. Then we get a simplified form of the preceding equation:

$$F(\mathbf{x}) = \sum_{i=1}^{N} c_i h(||\mathbf{x} - \mathbf{m}_i||) \tag{7.25}$$

It now becomes evident that we can implement the approximation problem with a three-layer neural network as shown in Fig. 7.11. The number of input neurons corresponds to the dimension of the feature space; the number of hidden neurons is problem-specific and is determined during the clustering process. The output neurons perform the superposition of the weighted basis functions and thus solve the approximation problem. Figure 7.11 depicts the architecture of the network for a single output. Clearly, such an architecture can be readily extended to accommodate any desired number of network outputs.

The network architecture is solely determined by the input-output mapping to be achieved. Moreover, from the viewpoint of approximation theory, the network has three desirable properties:

1. The network is a universal approximator that approximates arbitrarily well any multivariate continuous function, given a sufficiently large number of hidden units. This can be shown based on the Stone-Weierstrass theorem [297].

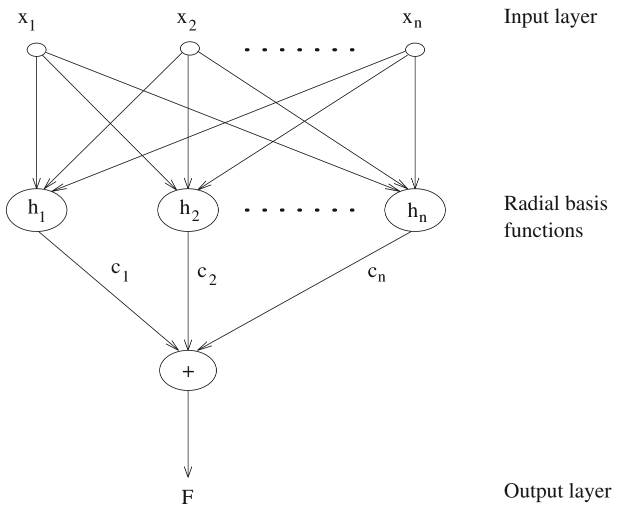

Figure 7.11 Approximation network.

2. The network has the best-approximation property. This means that given an unknown nonlinear function f, there always exists a choice of coefficients that approximate f better than other possible choices. This property holds for all other approximation schemes based on Green's functions as basis functions. It is not the case for a three-layer perceptron having sigmoid functions as activation functions. Sigmoid functions are not Green's functions, being additionally translation and rotation invariant.
3. By taking a closer look at the approximation approach,

$$F(\mathbf{x}) = \sum_{i=1}^{N} c_i h(||\mathbf{x} - \mathbf{m}_i||) \tag{7.26}$$

we can obtain a simplified form if we specify for each of the N radial basis functions a width σ_i,

$$F(\mathbf{x}) = \sum_{i=1}^{N} c_i\, g\left(\frac{||\mathbf{x} - \mathbf{m}_i||}{\sigma_i}\right) \tag{7.27}$$

In [286] it was shown that even then we can solve the approximation problem optimally, if all radial basis functions have the same width $\sigma_i = \sigma$. In neural network terminology, it means that neurons having the same Gaussian function as an activation function for the hidden nodes can approximate any given function.

7.4.2 Regularization Networks

The previous section dealt with the interpolation problem, and we showed how a function $F(\mathbf{x})$ can be determined to satisfy the interpolation condition. Assuming that the data are noise-corrupted, we immediately see that the problem is ill posed since it has an infinite number of solutions. To be able to choose one particular solution, we need to include some a priori knowledge of the function $F(\mathbf{x})$. This can be some prior smoothness information about the function, in the sense that similar inputs yield similar outputs. In [299], smoothness is defined as a measure describing the "oscillatory" behavior of a function. Thus, a smooth function exhibits few oscillations. Translated into the frequency domain, this means that a smooth function has less energy in the high-frequency range and therefore has a smaller bandwidth.

The regularization theory arose as a new approach for solving ill-posed problems [366]. The main idea of this method is to stabilize the solution by means of some auxiliary non-negative functional that includes a priori information about the solution. Thus, the solution to this ill-posed problem can be obtained from a variational principle, which includes both the data and prior smoothness information. Smoothness is considered by introducing a smoothness functional $\phi[F]$. A small value of $\phi[F]$ corresponds to smoother functions.

Let $\{\mathbf{m}_i \in \mathcal{R}^n | i = 1, \dots N\}$ be N input vectors and $\{d_i \in \mathcal{R} | i = 1, \dots, N\}$ be N real numbers and represent the desired outputs. The solution to the regularization problem is denoted by $F(\mathbf{x})$, and is required to approximate the data well and at the same time to be smooth. Thus, $F[\mathbf{x}]$ should minimize the following functional:

$$H[F] = \sum_{i=1}^{N} [d_i - F(\mathbf{m}_i)]^2 + \lambda\phi[F] \tag{7.28}$$

where λ is a positive number, called the regularization term. The first term in the equation enforces closeness to the data and measures the distance between the data and the desired solution F. The second term enforces smoothness and measures the cost associated with the deviation from smoothness. The regularization parameter $\lambda > 0$ controls the trade-off between these two terms.

The Principle of Regularization 7.4.1 *Determine the function $F(\mathbf{x})$ that minimizes the Tikhonov functional defined by*

$$H[F] = \sum_{i=1}^{N} [d_i - F(\mathbf{m}_i)]^2 + \lambda\phi[F] \tag{7.29}$$

with λ being the regularization parameter.

The solution for the regularization problem is derived in [299] and is given here in its simplified form

$$F(\mathbf{x}) = \frac{1}{\lambda} \sum_{i=1}^{N} [d_i - F(\mathbf{m}_i)]\, G(\mathbf{x}, \mathbf{m}_i) \tag{7.30}$$

The minimizing solution $F(\mathbf{x})$ to the regularization problem is thus given by a linear superposition of N Green's functions. The $\mathbf{m}_i$s represent the centers of the expansion and the weights $w_i = [d_i - F(\mathbf{m}_i)]\lambda$ represent the coefficients of the expansion.

Equation (7.30) can be similarly expressed as

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i G(\mathbf{x}_j, \mathbf{m}_i) \tag{7.31}$$

The Green's function is a symmetric function with $G(\mathbf{x}, \mathbf{m}_i) = G(\mathbf{m}_i, \mathbf{x})$. If the smoothness functional $\phi[F]$ is both translationally and rotationally invariant, then the Green's function $G(\mathbf{x}, \mathbf{m}_i)$ will depend only on $||\mathbf{x} - \mathbf{m}_i||$. Under these conditions, the Green's function has the same invariance properties, and eq. (7.31) takes a special form:

$$F_\lambda(\mathbf{x}) = \sum_{i=1}^{N} w_i G(||\mathbf{x} - \mathbf{m}_i||) \tag{7.32}$$

This solution is a strict interpolation, because all training data points are used to generate the interpolating function. The approximation scheme of eq. (7.31) can be easily interpreted as a network with only one hidden layer of units, which is referred to as a regularization network as shown in Fig. 7.11.

Let's take a closer look at the regularization parameter. It can be interpreted as an indicator of the sufficiency of the available data to specify the solution $F(\mathbf{x})$. If $\lambda \to 0$, the solution $F(\mathbf{x})$ is sufficiently determined from the available examples; if $\lambda = 0$, then this corresponds to pure interpolation. If $\lambda \to \infty$, this means the examples are unreliable. In practice, λ is between these two limits in order to enable both the sample data and the prior information to contribute to the solution $F(\mathbf{x})$.

The regularization problem represents an alleviation to a well-known and frequent problem in training neural networks, the so-called overtraining. If noise is added to input data, the neural network learns all the details of the input-output pairs but not their general form. This phenomenon is overtraining. A smart choice of the regularization parameter can prevent this.

Example 7.4.1 Figure 7.12 shows the effect of overtraining based on the example of learning an input-output mapping.

In practical applications, we have two choices for the Tikhonov functional in order to avoid overtraining:

- Zero-order regularization pertains to the weights of the network. The minimizer of the Tikhonov functional has a penalty term for the weights and in that way avoids overtraining:

$$\mathcal{E}(F) = \frac{1}{2} \sum_p \sum_i (d_i - F(\mathbf{m}_i))^2 + \lambda \sum_i \sum_j w_{ij}^2 \tag{7.33}$$

- Accelerated regularization pertains to the neurons of the hidden layer. This is a fast regularization network, since it tries to determine the centers and the widths of the radial basis functions.

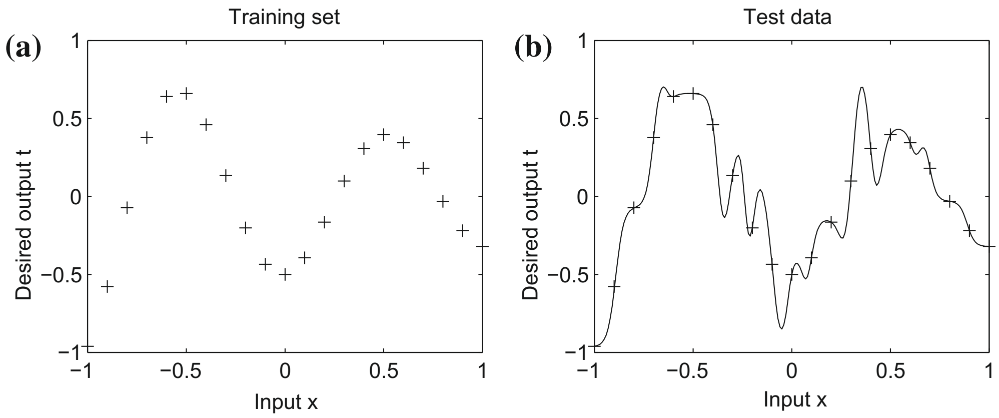

Figure 7.12 Overtraining: (a) learning data; and (b) testing data.

7.4.3 Data Processing in Radial Basis Function Networks

Data processing in a radial basis function neural network is quite different from standard supervised or unsupervised learning techniques. The MLP's backpropagation algorithm represented an optimization method known in statistics as stochastic approximation, while the "winner takes all" concept is neurobiologically inspired strategy used for training self-organizing feature maps or in LVQ.

In this section we present a different method by looking into the design of a neural network as an approximation problem in a high-dimensional space. In neural network parlance, the hidden units are kernel functions that constitute an arbitrary "basis" for the input patterns (vectors) when they are mapped onto the hidden-unit space; these functions are known as radial basis functions. However, there is a neurobiological equivalent to the mathematical concept of kernel or radial basis functions: the receptive field. It describes the local interaction between neurons in a cortical layer.

Fundamental contributions to the theory, design, and application of radial basis function networks are presented in an examplary manner in [255, 298].

The construction of a radial basis function (RBF) network in its most basic form involves three different layers. For a network with N hidden neurons, the output of the ith output node $f_i(\mathbf{x})$ when the n-dimensional input vector $\mathbf{x}$ is presented is given by

$$f_i(\mathbf{x}) = \sum_{j=1}^{N} w_{ij} \Psi_j(\mathbf{x}) \tag{7.34}$$

where $\Psi_j(\mathbf{x}) = \Psi(||\mathbf{x} - \mathbf{m}_j||/\sigma_j)$ is a suitable radially symmetric function that defines the output of the jth hidden node. Often $\Psi(.)$ is chosen to be the Gaussian function where the width parameter σ_j is the standard deviation. $\mathbf{m}_j$ is the location of the jth centroid, where each centroid is represented by a kernel/hidden node, and w_{ij} is the

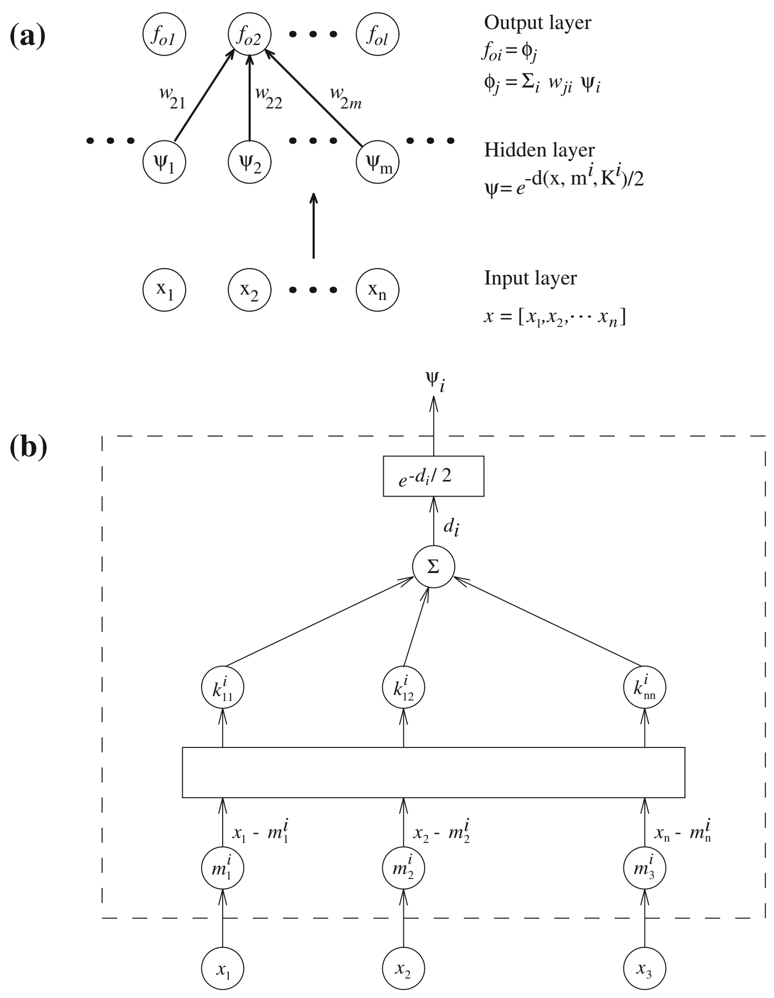

Figure 7.13 RBF network: (a) three-layer model; and (b) the connection between the input layer and the hidden layer neuron.

weight connecting the *j*th kernel/hidden node to the *i*th output node. Figure 7.13a illustrates the configuration of the network.

The steps that govern the data flow through the radial basis function network during classification and represent a typical RBF network learning algorithm are described below:

1. **Initialization:** Choose random values for the initial weights of the RBF network. The magnitude of the weights should be small. Choose the centers $\mathbf{m}_i$ and the shape matrices $\mathbf{K}_i$ of the N given radial basis functions.
2. **Sampling:** Draw randomly a pattern $\mathbf{x}$ from the input data. This pattern represents the input to the neural network.

3. **Forward computation of hidden layer's activations:** Compute the values of the hidden layer nodes as illustrated in Fig. 7.13b:

$$\psi_i = \exp\left(-d(\mathbf{x}, \mathbf{m}_i, \mathbf{K}_i)/2\right) \tag{7.35}$$

$d(\mathbf{x}, \mathbf{m}_i) = (\mathbf{x} - \mathbf{m}_i)^T \mathbf{K}_i (\mathbf{x} - \mathbf{m}_i)$ is a metric norm and is known as the Mahalanobis distance. The *shape matrix* $\mathbf{K}_i$ is positive definite and its elements k^i_{jk};

$$\mathbf{K}_i jk = \frac{h_{jk}}{\sigma_j * \sigma_k} \tag{7.36}$$

represent the correlation coefficients h_{jk} and σ_j the standard deviation of the ith shape matrix.

We choose for $h_{jk} : h_{jk} = 1$ for $j = k$ and $|h_{jk}| \leq 1$ otherwise.

4. **Forward computation of output layer's activations:** Calculate the values of the output nodes according to

$$f_{oj} = \phi_j = \sum_i w_{ji} \psi_i \tag{7.37}$$

5. **Updating:** Adjust weights of all neurons in the output layer based on a steepest descent rule.
6. **Continuation:** Continue with step 2 until no noticeable changes in the error function are observed.

The algorithm is based on the a priori knowledge of the radial basis centers and its shape matrices. In the following section, we will review some design strategies for RBF networks and show how we can determine the centers and widths of the radial basis functions.

RBF networks have been applied to a variety of problems in image processing, such as image coding and analysis [321], and in medical diagnosis [410].

7.4.3.1 Design Considerations

RBF networks represent, in contrast to the MLP, local approximators to nonlinear input-output mapping. Their main advantages are a short training phase and a reduced sensitivity to the order of presentation of training data. In many cases, however, we find that a smooth mapping is only achieved if the number of radial basis functions required to span the input space becomes very large. This hampers the feasibility of many practical applications.

The RBF network has only one hidden layer, and the number of basis functions and their shape is problem-oriented and can be determined online during the learning process [211,295]. The number of neurons in the input layer equals the dimension of the feature vector. Likewise, the number of nodes in the output layer corresponds to the number of classes.

7.4.4 Training Techniques for RBF Networks

As stated in the beginning, the learning process of the RBF networks is considered hybrid. We distinguish two different dynamics: the adjustment of the linear weights of the output neurons and the nonlinear activation functions of the hidden neurons. The weight adjustment of the output layer's neurons represents a linear process compared to the nonlinear parameter adjustment of the hidden layer neurons. Since the hidden and output layers of an RBF network perform different tasks, we must separate the optimization of the hidden and output layers, and thus employ different learning techniques. The output layer's synapses are updated according to a simple delta rule as shown in the MLP case.

We have some degrees of freedom in choosing the kernel functions of an RBF network: in the simple case they are fixed, or they can be adjusted during the training phase.

There are several techniques to design an RBF network, depending on how the centers of the radial basis functions of the network are determined. The best-known strategies chosen for practical applications are reviewed in the following.

7.4.4.1 Fixed Centers of the RBF Selected at Random

The simplest case is to assume fixed kernel functions with a center location chosen at random from the training set.

Specifically, a radial basis function centered at $\mathbf{m}_i$ is defined as

$$G(||\mathbf{x}-\mathbf{m}_i||^2) = \exp\left(-\frac{M}{d_{max}^2}||\mathbf{x}-\mathbf{m}_i||^2\right), \quad i = 1, \dots, M \tag{7.38}$$

M represents the number of centers, and d_{max}^2 is the maximum distance between the selected centers. In fact, the standard deviation of all neurons is fixed at

$$\sigma = \frac{d_{max}}{\sqrt{2M}} \tag{7.39}$$

This guarantees that they are neither too spiky nor too flat. The only parameters that have to be adjusted here are the linear weights in the output layer of the network. They can be determined based on the pseudoinverse method [39]. Specifically, we have

$$\mathbf{w} = \mathbf{G}^{+}\mathbf{y} \tag{7.40}$$

where $\mathbf{y}$ is the desired response vector in the training set.

$\mathbf{G}^{+}$ is the pseudoinverse of the matrix $\mathbf{G} = \{g_{ji}\}$, which is itself defined as $\mathbf{G} = \{g_{ji}\}$, where

$$g_{ji} = \exp\left(-\frac{M}{d_{max}^2}||\mathbf{x_j}-\mathbf{m}_i||^2\right), \quad j = 1, \dots, N; \quad i = 1, \dots, M \tag{7.41}$$

$\mathbf{x}_j$ is the jth input vector of the training sample.

7.4.4.2 Self-Organized Selection of the Centers of the RBFs

The locations of the centers of the kernel functions (hidden units) are adapted based on a self-organizing learning strategy, whereas the linear weights associated with the output neurons are adapted based on a simple supervised delta rule. It's important to mention that this procedure allocates hidden units only in preponderantly input data regions.

The self-organized learning process is based on a clustering algorithm that partitions the given data set into homogeneous subsets. The k-means clustering algorithm [84] sets the centers of the radial basis functions only in those regions where significant input data are present. The optimal number M of radial basis functions is determined empirically. Let $\{\mathbf{m}_i(n)\}_{i=1}^{M}$ be the centers of the radial basis functions at iteration n of the algorithm.

Then, the k-means algorithm applied to center selection in RBF neural networks proceeds as follows:

1. **Initialization:** Choose randomly the initial centers $\{\mathbf{m}_i(0)\}_{i=1}^{M}$ of the radial basis functions.
2. **Sampling:** Draw a sample $\mathbf{x}$ from the data space with a certain probability.
3. **Similarity matching:** Determine the best-matching (winning) neuron $k(\mathbf{x})$ at iteration n using the Euclidean distance:

   $$k(\mathbf{x}) = \overset{\text{argmin}}{k} \; ||\mathbf{x}(n) - \mathbf{m}_k(n)||, \quad k = 1, 2, \ldots M \tag{7.42}$$

   $\mathbf{m}_k(n)$ is the center of the kth radial basis function at iteration n.
4. **Updating:** Adjust the centers of the radial basis functions, using the update equation

   $$\mathbf{x}_k(n+1) = \begin{cases} \mathbf{x}_k(n) + \eta(n)[\mathbf{x}(n) - \mathbf{x}_k(n)], & k = k(\mathbf{x}) \\ \mathbf{x}_k(n), & \text{else} \end{cases} \tag{7.43}$$

   η is the learning rate that lies in the range $0 < \eta < 1$.
5. **Continuation:** Go back to step 2 and continue the procedure until no noticeable changes are observed in the radial basis centers.

After determining the centers of the radial basis functions, the next and final stage of the hybrid learning process is to estimate the weights of the output layer based, for example, on the least-mean-square (LMS) algorithm [285].

7.4.4.3 Supervised Selection of Centers

All free parameters of the kernel functions including centers are adapted based on a supervised learning strategy. In such a case, we deal with an RBF network in its most generalized form. This can be realized by an error-correction learning, which requests an implementation based on a gradient-descent procedure.

In the first step, we define a cost function

$$E = \frac{1}{2} \sum_{j=1}^{P} e_j^2 \tag{7.44}$$

Table 7.1 Adaptation formulas for the linear weights and the position and widths of centers for an RBF network [129].

1.	**Linear weights of the output layer**
	$\frac{\partial \mathcal{E}(n)}{\partial w_i(n)} = \sum_{j=1}^{N} e_j(n) G(\lvert\lvert \mathbf{x}_j - \mathbf{m}_i(n) \rvert\rvert)$
	$w_i(n+1) = w_i(n) - \eta_1 \frac{\partial \mathcal{E}(n)}{\partial w_i(n)}, \quad i = 1, \ldots, M$
2.	**Position of the centers of the hidden layer**
	$\frac{\partial \mathcal{E}(n)}{\partial \mathbf{m}_i(n)} = 2 w_i(n) \sum_{j=1}^{N} e_j(n) G'(\lvert\lvert \mathbf{x}_j - \mathbf{m}_i(n) \rvert\rvert) \mathbf{K}^{\mathbf{i}} [\mathbf{x}_j - \mathbf{m}_i(n)]$
	$\mathbf{m}_i(n+1) = \mathbf{m}_i(n) - \eta_2 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{m}_i(n)}, \quad i = 1, \ldots, M$
3.	**Widths of the centers of the hidden layer**
	$\frac{\partial \mathcal{E}(n)}{\partial \mathbf{k}^{\mathbf{i}}(n)} = -w_i(n) \sum_{j=1}^{N} e_j(n) G'(\lvert\lvert \mathbf{x}_j - \mathbf{m}_i(n) \rvert\rvert) \mathbf{Q}_{ji}(n)$
	$\mathbf{Q}_{ji}(n) = [\mathbf{x}_j - \mathbf{m}_i(n)][\mathbf{x}_j - \mathbf{m}_i(n)]^T$
	$\mathbf{K}^{\mathbf{i}}(n+1) = \mathbf{K}^{\mathbf{i}}(n) - \eta_3 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{K}^{\mathbf{i}}(n)}$

where P is the size of the training sample and e_j is the error defined by

$$e_j = d_j - \sum_{i=1}^{M} w_i G(||\mathbf{x}_j - \mathbf{m}_i||_{C_i}) \tag{7.45}$$

The requirement is to find the free parameters $w_i, \mathbf{m}_i$, and Σ_i^{-1} such that the error E is minimized.

The results of this minimization [129] are summarized in Table 7.1. From the table we can see that the update equations for $w_i, \mathbf{x}_i$, and Σ_i^{-1} have different learning rates, thus showing the different timescales. Unlike the backpropagation algorithm, the gradient-descent procedure described in Table 7.1 for an RBF network does not employ error backpropagation.

7.5. TRANSFORMATION RADIAL BASIS NEURAL NETWORKS

The selection of appropriate features is an important precursor to most statistical pattern recognition methods. A good feature selection mechanism helps to facilitate classification by eliminating noisy or nonrepresentative features that can impede recognition. Even features that provide some useful information can reduce the accuracy of a classifier when the amount of training data is limited. This so-called "curse of dimensionality," along with the expense of measuring and including features, demonstrates the utility of obtaining a minimum-sized set of features that allow a classifier to discern pattern classes well. Well-known methods in the literature applied to feature selection are the floating search methods [304] and genetic algorithms [341].

Radial basis neural networks are excellent candidates for feature selection. It is necessary to add an additional layer to the traditional architecture to obtain a representation

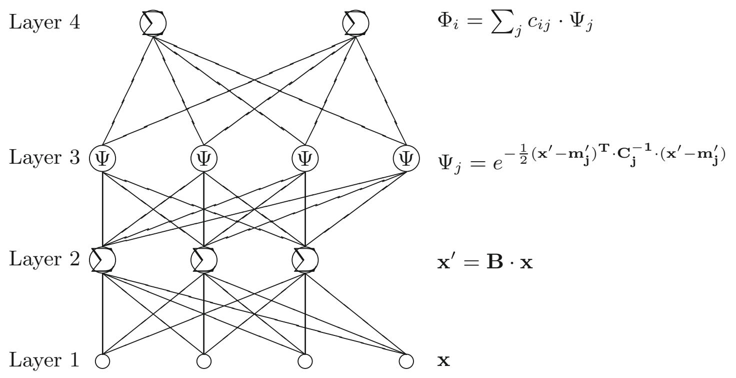
Layer 4
Layer 3
Layer 2
Layer 1
$\Phi_i = \sum_j c_{ij} \cdot \Psi_j$
$\Psi_j = e^{-\frac{1}{2}(\mathbf{x}'-\mathbf{m}_j')^T \cdot \mathbf{C}_j^{-1} \cdot (\mathbf{x}'-\mathbf{m}_j')}$
$\mathbf{x}' = \mathbf{B} \cdot \mathbf{x}$
$\mathbf{x}$

Figure 7.14 Linear transformation radial basis neural network.

of relevant features. The new paradigm is based on an explicit definition of the relevance of a feature and realizes a linear transformation of the feature space.

Figure 7.14 shows the structure of a radial basis neural network with the additional layer 2, which transforms the feature space linearly by multiplying the input vector and the center of the nodes by the matrix **B**. The covariance matrices of the input vector remain unmodified:

$$\mathbf{x}' = \mathbf{B}\mathbf{x}, \quad \mathbf{m}' = \mathbf{B}\mathbf{m}, \quad \mathbf{C}' = \mathbf{C} \tag{7.46}$$

The neurons in layer 3 evaluate a kernel function for the incoming input while the neurons in the output layer perform a weighted linear summation of the kernel functions:

$$\mathbf{y}(\mathbf{x}) = \sum_{i=1}^{N} w_i \exp\left(-d(\mathbf{x}', \mathbf{m}_i')/2\right) \tag{7.47}$$

with

$$d(\mathbf{x}', \mathbf{m}_i') = (\mathbf{x}' - \mathbf{m}_i')^T \mathbf{C}_i^{-1} (\mathbf{x}' - \mathbf{m}_i') \tag{7.48}$$

Here, N is the number of neurons in the second hidden layer, $\mathbf{x}$ is the n-dimensional input pattern vector, $\mathbf{x}'$ is the transformed input pattern vector, $\mathbf{m}_i'$ is the center of a node, w_i are the output weights, and $\mathbf{y}$ represents the m-dimensional output of the network. The $n \times n$ covariance matrix $\mathbf{C}_i$ is of the form

$$C_{jk}^{i} = \begin{cases} \frac{1}{\sigma_{jk}^2} & \text{if} \quad m = n \\ 0 & \text{otherwise} \end{cases} \tag{7.49}$$

where σ^{jk} is the standard deviation. Because the centers of the Gaussian potential function units (GPFUs) are defined in the feature space, they will be subject to transformation by **B**, as well. Therefore, the exponent of a GPFU can be rewritten as

$$d(\mathbf{x}, \mathbf{m}_i') = (\mathbf{x} - \mathbf{m}_i)^T \mathbf{B}^T \mathbf{C}_i^{-1} \mathbf{B} (\mathbf{x} - \mathbf{m}_i) \tag{7.50}$$

and is in this form similar to eq. (7.48).

For the moment, we will regard **B** as the identity matrix. The network models the distribution of input vectors in the feature space by the weighted summation of Gaussian normal distributions, which are provided by the Gaussian Potential Function Units (GPFU) Ψ_j. To measure the difference between these distributions, we define the relevance ρ_n for each feature x_n:

$$\rho_n = \frac{1}{PJ} \sum_p \sum_j \frac{(x_{pn} - m_{jn})^2}{2\sigma_{jn}^2} \tag{7.51}$$

where P is the size of the training set and J is the number of GPFUs. If ρ_n falls below the threshold ρ_{th}, one will decide to discard feature x_n. This criterion will not identify every irrelevant feature: If two features are correlated, one of them will be irrelevant, but this cannot be indicated by the criterion.

7.5.1 Learning Paradigm for the Transformation Radial Basis Neural Network

We follow the idea of [211] for the implementation of the neuron allocation and learning rules for the TRBNN. The network generation process starts initially without any neuron.

The mutual dependency of correlated features can often be approximated by a linear function, which means that a linear transformation of the input space can render features irrelevant.

First we assume that layers 3 and 4 have been trained so that they comprise a model of the pattern-generating process while **B** is the identity matrix. Then the coefficients B_{nr} can be adapted by gradient descent with the relevance ρ'_n of the transformed feature x'_n as the target function. Modifying B_{nr} means changing the relevance of x_n by adding x_r to it with some weight B_{nr}. This can be done online, that is, for every training vector $\mathbf{x_p}$ without storing the whole training set. The diagonal elements B_{nn} are constrained to be constant 1, because a feature must not be rendered irrelevant by scaling itself. This in turn guarantees that no information will be lost. B_{nr} will only be adapted under the condition that $\rho_n < \rho_p$, so that the relevance of a feature can be decreased only by some more relevant feature. The coefficients are adapted by the learning rule

$$B_{nr}^{new} = B_{nr}^{old} - \mu \frac{\partial \rho_n}{\partial B_{nr}} \tag{7.52}$$

with the learning rate μ and the partial derivative

$$\frac{\partial \rho_n}{\partial B_{nr}} = \frac{1}{PJ} \sum_p \sum_j \frac{(x'_{pn} - m'_{jn})}{\sigma_{jn}^2} (x'_{pr} - m'_{jr}) \tag{7.53}$$

In the learning procedure, which is based on, for example, [211], we minimize according to the LMS criterion the target function:

$$E = \frac{1}{2}\sum_{p=0}^{P} |y(\mathbf{x}) - \Phi(\mathbf{x})|^2 \tag{7.54}$$

where P is the size of the training set. The neural network has some useful features, such as automatic allocation of neurons, discarding of degenerated and inactive neurons, and variation of the learning rate depending on the number of allocated neurons.

The relevance of a feature is optimized by gradient descent:

$$\rho_i^{new} = \rho_i^{old} - \eta \frac{\partial E}{\partial \rho_i} \tag{7.55}$$

Based on the newly introduced relevance measure and the change in the architecture we get the following correction equations for the neural network:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= -(y_i - \Phi_i)\Psi_j \\ \frac{\partial E}{\partial m_{jn}} &= -\sum_i (y_i - \Phi_i) w_{ij} \Psi_j \sum_k (x'_k - m'_{jk}) \frac{B_{kn}}{\sigma_{jk}^2} \\ \frac{\partial E}{\partial \sigma_{jn}} &= -\sum_i (y_i - \Phi_i) w_{ij} \Psi_j \frac{(x'_n - m'_{jn})^2}{\sigma_{jn}^3} \end{aligned} \tag{7.56}$$

In the transformed space the hyperellipses have the same orientation as in the original feature space. Hence they do not represent the same distribution as before. To overcome this problem, layers 3 and 4 will be adapted at the same time as **B**. If these layers converge fast enough, they can be adapted to represent the transformed training data, providing a model on which the adaptation of **B** can be based. The adaptation with two different target functions (E and ρ) may become unstable if **B** is adapted too fast, because layers 3 and 4 must follow the transformation of the input space. Thus μ must be chosen $\ll \eta$. A large gradient has been observed, causing instability when a feature of extreme high relevance is added to another. This effect can be avoided by dividing the learning rate by the relevance, that is, $\mu = \mu_0/\rho_r$.

7.6. SUPPORT VECTOR MACHINES

The support vector machine [69,335,379] is an elegant universal feedforward network with a single hidden layer of nonlinear processing units and can be applied to a wide range of classification problems.

The basic idea of the SVM algorithm is to construct a hyperplane

$$f_{\mathbf{w}}, b(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{7.57}$$

with normal vector $\mathbf{w} \in \mathbf{R}^n$ and bias b, which separates the labeled training data into two classes with maximum-margin: given a set of N labeled training examples $\{(\mathbf{x}, y)_i\}, i = 1, \ldots, N, \ \mathbf{x}_i \in R^n$ belonging to two different classes $y_i \in \{-1, 1\}$, a maximum-margin hyperplane is determined which separates the training examples of the two different categories so that the distance between the hyperplane and the closest examples (the margin γ) is maximized. This hyperplane is fully specified by a subset of the training examples representing those points that lie closest to the decision surface and pose the biggest challenge in terms of classification.

Formally speaking, the margin $\gamma = \frac{1}{|\mathbf{w}|}$ of the hyperplane (7.57) is maximized by solving the following constrained optimization problem:

$$\min_{\mathbf{w},b} \qquad \mathbf{w} \cdot \mathbf{w} \tag{7.58}$$

$$\text{subject to:} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i = 1, \ldots, N \tag{7.59}$$

This optimization problem is solved by employing the Lagrange theory, leading to the maximum-margin hyperplane normal vector

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{7.60}$$

with α_i being the Lagrange coefficients. In practice, the decision function

$$f_{\mathbf{w},b}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b \tag{7.61}$$

is frequently determined by only a small subset of training examples—the so-called support vectors—with $\alpha_i > 0$, while the remaining examples with $\alpha_i = 0$ can be neglected.

In case of nonseparable data, we substitute in the above equation the inner product $\mathbf{x}_i \cdot \mathbf{x}_j$ by a nonlinear kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. This kernel function evaluates the inner product between two examples after their transformation by a nonlinear function $\Phi(\mathbf{x})$. By doing the same in the Lagrangian form of the constrained quadratic optimization problem, the hyperplane is optimized in a new feature space and corresponds to a nonlinear decision function in the original data space. A frequently used nonlinear kernel function is the Gaussian kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right). \tag{7.62}$$

Another frequently used kernel is the polynomial kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^p \tag{7.63}$$

which corresponds to a mapping Φ in the space of all monomials of degree p and becomes for $p = 1$ a linear kernel. Less employed is sigmoid kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tan h(\mathbf{x}_i, \mathbf{x}_j) \tag{7.64}$$

Solving multiclass problems with the SVM algorithm requires a suitable decomposition of the classification task into a sequence of binary subtasks, which each can be handled by employing the standard SVM algorithm. The outputs of the binary classifiers are then recombined to the final multiclass prediction of the multiclass SVM (MSVM).

As an example, let's assume that three different tissue classes have to be distinguished. Each tissue class is considered in one of the binary subtasks as the target class to be distinguished from the union of the remaining classes (one-vs-all decomposition scheme). The MSVM then returns three-dimensional vectors with components reflecting the outcomes $f^c_{\mathbf{w},b}(\mathbf{x}),\ c = 1, \ldots, 3$ of the three binary SVMs. In order to increase the interpretability of the classification outcome, it is transformed into posteriori probabilities by postprocessing with a parameterized softmax function

$$P(\text{class}_k|\mathbf{x}) = \frac{\exp(a^k_1 f^k_{\mathbf{w},b}(\mathbf{x}) + a^k_0)}{\sum_{c=1}^{3} \exp(a^c_1 f^c_{\mathbf{w},b}(\mathbf{x}) + a^c_0)} \tag{7.65}$$

The parameters a^c_0, a^c_1 are estimated by minimizing the cross-entropy error on a subset of the training data.

7.7. HOPFIELD NEURAL NETWORKS

Hopfield neural networks represent a new neural computational paradigm by implementing an autoassociative memory. They are recurrent or fully interconnected neural networks. There are two versions of Hopfield neural networks: in the binary version all neurons are connected to each other but there is no connection from a neuron to itself, and in the continuous case all connections including self-connections are allowed.

A pattern, in N-node Hopfield neural network parlance, is an N-dimensional vector $\mathbf{p} = [p_1, p_2, \ldots, p_N]$ from the space $\mathbf{P} = \{-1, 1\}^N$. A special subset of $\mathbf{P}$ represents the set of stored or reference patterns $\mathbf{E} = \{\mathbf{e^k} : 1 \le k \le K\}$, where $\mathbf{e^k} = [e^k_1, e^k_2, \ldots, e^k_N]$. The Hopfield net associates a vector from $\mathbf{P}$ with a certain stored (reference) pattern in $\mathbf{E}$. The neural net splits the binary space $\mathbf{P}$ into classes whose members bear in some way similarity to the reference pattern that represents the class. The Hopfield network finds a broad application area in image restoration and segmentation.

As already stated in the Introduction, neural networks have four common components.

For the Hopfield net we have the following:

Neurons: The Hopfield network has a finite set of neurons $\mathbf{x}(i), 1 \le i \le N$, which serve as processing units. Each neuron has a value (or state) at time t described by $\mathbf{x}_t(i)$. A neuron in the Hopfield net has one of the two states, either -1 or $+1$; that is, $\mathbf{x}_t(i) \in \{-1, +1\}$.

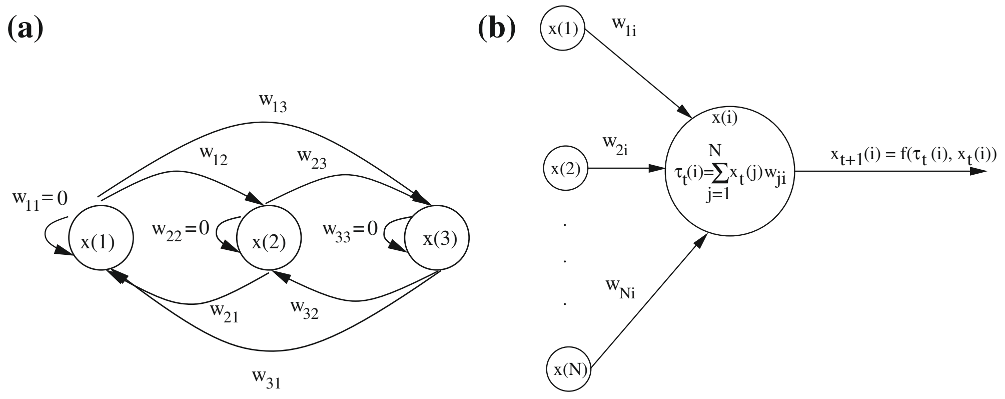

Figure 7.15 (a) Hopfield neural network and (b) propagation rule and activation function for the Hopfield network.

Synaptic connections: The learned information of a neural net resides within the interconnections between its neurons. For each pair of neurons, $\mathbf{x}(i)$ and $\mathbf{x}(j)$, there is a connection w_{ij} called the synapse between $\mathbf{x}(i)$ and $\mathbf{x}(j)$. The design of the Hopfield net requires that $w_{ij} = w_{ji}$ and $w_{ii} = 0$. Figure 7.15a illustrates a three-node network.

Propagation rule: This defines how states and synapses influence the input of a neuron. The propagation rule $\tau_t(i)$ is defined by

$$\tau_t(i) = \sum_{j=1}^{N} \mathbf{x}_t(j) w_{ij} + b_i \tag{7.66}$$

b_i is the externally applied bias to the neuron.

Activation function: The activation function f determines the next state of the neuron $\mathbf{x}_{t+1}(i)$ based on the value $\tau_t(i)$ computed by the propagation rule and the current value $\mathbf{x}_t(i)$. Figure 7.15b illustrates this fact. The activation function for the Hopfield net is the hard limiter defined here:

$$\mathbf{x}_{t+1}(i) = f(\tau_t(i), \mathbf{x}_t(i)) = \begin{cases} 1, & \text{if} \quad \tau_t(i) > 0 \\ -1, & \text{if} \quad \tau_t(i) < 0 \end{cases} \tag{7.67}$$

The network learns patterns that are N-dimensional vectors from the space $\mathbf{P} = \{-1, 1\}^N$. Let $\mathbf{e^k} = [e_1^k, e_2^k, \ldots, e_n^k]$ define the kth exemplar pattern where $1 \le k \le K$. The dimensionality of the pattern space is reflected in the number of nodes in the net, such that the net will have N nodes $\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(N)$.

The training algorithm of the Hopfield neural network is simple and is outlined below:

1. **Learning:** Assign weights w_{ij} to the synaptic connections:

$$w_{ij} = \begin{cases} \sum_{k=1}^{K} e_i^k e_j^k, & \text{if} \quad i \ne j \\ 0, & \text{if} \quad i = j \end{cases} \tag{7.68}$$

Keep in mind that $w_{ij} = w_{ji}$, so it is necessary to perform the preceding computation only for $i < j$.

2. **Initialization:** Draw an unknown pattern. The pattern to be learned is now presented to the net. If $\mathbf{p} = [p_1, p_2, \ldots, p_N]$ is the unknown pattern, set

$$\mathbf{x}_0(i) = p_i, \quad 1 \leq i \leq N \tag{7.69}$$

3. **Adaptation:** Iterate until convergence. Using the propagation rule and the activation function we get for the next state,

$$\mathbf{x}_{t+1}(i) = f\left(\sum_{j=1}^{N} \mathbf{x}_t(j) w_{ij}, \mathbf{x}_t(i)\right) \tag{7.70}$$

   This process should be continued until any further iteration will produce no state change at any node.
4. **Continuation:** For learning a new pattern, repeat steps 2 and 3.

In case of the continuous version of the Hopfield neural network, we have to consider neural self-connections $w_{ij} \neq 0$ and choose as an activation function a sigmoid function. With these new adjustments, the training algorithm operates in the same way.

The convergence property of Hopfield's network depends on the structure of $\mathbf{W}$ (the matrix with elements w_{ij}) and the updating mode. An important property of the Hopfield model is that if it operates in a sequential mode and $\mathbf{W}$ is symmetric with nonnegative diagonal elements, then the energy function

$$\begin{aligned} E_{hs}(t) &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i(t)x_j(t) - \sum_{i=1}^{n} b_i x_i(t) \\ &= -\frac{1}{2}\mathbf{x}^T(t)\mathbf{W}\mathbf{x}(t) - \mathbf{b}^T\mathbf{x}(t) \end{aligned} \tag{7.71}$$

is nonincreasing [138]. The network always converges to a fixed point.

Hopfield neural networks are applied to solve many optimization problems. In medical image processing, they are applied in the continuous mode to image restoration, and in the binary mode to image segmentation and boundary detection.

The continuous version will be extensively described in Chapter 8 as a subclass of additive activation dynamics.

7.8. COMPARING STATISTICAL, SYNTACTIC, AND NEURAL PATTERN RECOGNITION METHODS

The delimitations between statistical, syntactic, and neural pattern recognition approaches are not necessarily clear. All these approaches share common features and have a correct classification result as a common goal. The decision to choose a particular approach over another is based on analysis of underlying statistical components, or grammatical structure or on the suitability of neural network solution.

Table 7.2 Comparing statistical, syntactic, and neural pattern recognition approaches.

	Statistical	**Syntactic**	**Neural**
Pattern generation basis	Probabilistic models	Formal grammars	Stable state or weight matrix
Pattern classification basis	Estimation or decision theory	Parsing	Neural network properties
Feature organization	Input vector	Structural relations	Input vector
Training mechanism			
Supervised	Density estimation	Forming grammars	Determining neural network parameters
Unsupervised	Clustering	Clustering	Clustering
Limitations	Structural information	Learning structural rules	Semantic information

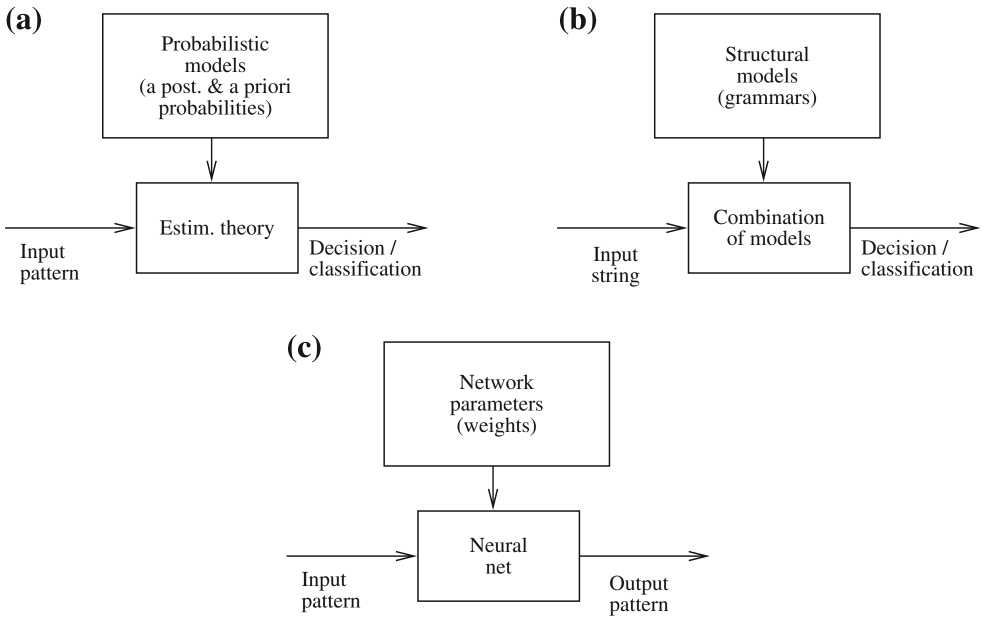
(a)
Probabilistic models (a post. & a priori probabilities)
Estim. theory
Input pattern
Decision / classification
(b)
Structural models (grammars)
Combination of models
Input string
Decision / classification
(c)
Network parameters (weights)
Neural net
Input pattern
Output pattern

Figure 7.16 Pattern recognition approaches: (a) statistical approach; (b) syntactic approach; and (c) neural approach.

Table 7.2 and Fig. 7.16 summarize the differences between the three pattern recognition approaches [329].

For statistical pattern recognition, the pattern structure is considered insignificant. If structural information from pattern is available, we choose the syntactic approach. The neural network approach, sometimes considered an alternative technique, provides useful methods for processing patterns derived from biological systems and can emulate the computational paradigm of the human brain.

A more sophisticated approach is the newly introduced concept of biologically oriented neural networks [235]. They are robust and powerful classification techniques which disambiguate suspicious feature identification during a computer-aided diagnosis. They are derived from the newest biological discoveries aiming to imitate decision-making and sensory processing in biological systems. They have an advantage over other artificial intelligence methods since they directly incorporate brain-based mechanisms, and therefore make CAD systems more powerful.

There are also some drawbacks with the described methods: The statistical method cannot include structural information, the syntactic method does not operate based on adaptation rules, and the neural network approach doesn't contain semantic information in its architecture.

7.9. PIXEL LABELING USING NEURAL NETWORKS

Pixel labeling is a data-driven pixel classification approach leading to image segmentation. The goal is to assign a label to every pixel in the image. For a supervised classifier, the labels reflect the tissue types of the pixels. For an MRI image, appropriate labels will be gray matter, white matter, cerebrospinal fluid, bone, and fat. For an unsupervised classifier, the labels are arbitrarily assigned to pixels. A subsequent process will determine the correspondence between labels and tissue types.

7.9.1 Noncontextual Pixel Labeling

For this type of labeling, no pixel neighborhood (context information) is considered. Instead data obtained from several sources, so-called multispectral data, are used. For MRI, multispectral data are T_1, T_2, and proton density weighted images. All or only a subset of the multispectral data can be taken to generate a feature vector for each pixel location. For example, if T_1 and T_2 weighted images are used, then the vector $\mathbf{x}$ generated for a pixel at position (m, n) is

$$\mathbf{x}_{m,n} = \left(T_{1_{mn}}, T_{2_{mn}}\right) \tag{7.72}$$

Figure 7.17 shows an example of a feature vector for a single pixel.

The resulting feature vectors are classified by assigning a single label to each pixel position. This leads to cluster of points described by pixel feature vectors from a single tissue type. It has been shown [30] that even using multispectral data from MRI images, the characteristics of different tissue types overlap. This represents a problem in MRI segmentation.

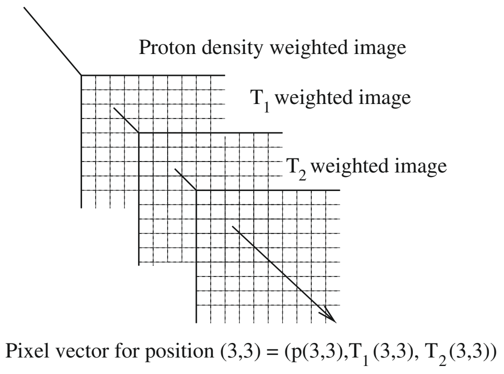
Proton density weighted image
T_1 weighted image
T_2 weighted image
Pixel vector for position (3,3) = (p(3,3),T_1(3,3), T_2(3,3))

Figure 7.17 Multispectral data describing a feature vector for a pixel in an image.

7.9.2 Contextual Pixel Labeling

Noncontextual pixel labeling often leads to noisy results. An alternative technique involves the context information of pixels and uses the data in the spatial neighborhood of the pixel to be classified. Figure 7.18 visualizes this procedure. The feature vector may include both textural and contextual information. Usually, pixel intensity information across scales or measurements based on pixel intensity are used as features. In mammography, this pixel labeling is quite often used for tumor segmentation [77,410]. A multiscale approach for MRI segmentation was proposed in [125]. Each pixel in an image was assigned a feature vector containing a limited number of differential geometric invariant features evaluated across a range of different scales at the pixel location. The features measured at different scales are products of various first- and second-order image derivatives.

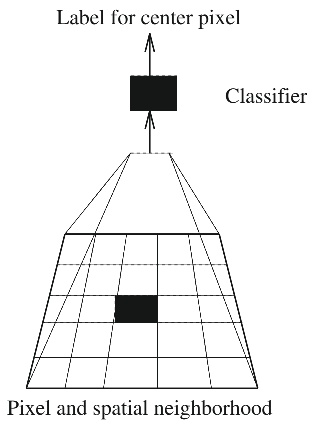

Figure 7.18 Contextual pixel labeling.

7.10. CLASSIFICATION STRATEGIES FOR MEDICAL IMAGES

There are two different strategies for classifying individual regions in medical images: region-based and pixel-based classification. For the region-based classification, the feature vector contains measurements of a set of features from a region in an image, while for the pixel-based classification, contextual or noncontextual information about every single pixel is obtained.

7.10.1 Pixel-Based Classification

Some features have been designed specifically for pixel-based classification in which a feature value is computed for each pixel. Each pixel in the image is then classified individually. Laws' texture energy measures are features which have been shown to discriminate well between various textured regions in an image when individual pixels are classified. These measures were successfully employed for spiculated lesion detection [393]. A pixel-level feature is considered a feature which is computed in the neighborhood of a pixel and is associated with that pixel. The neighborhood pixel will have a separate, and possibly different, feature value.

As stated before, classification at pixel level includes either contextual or noncontextual information of each image pixel. This technique is illustrated for MRI corresponding to acoustic neuromas [80]. A 7×7 pixel patch size, with a central square size of 5×5 pixels, was employed as the scheme to encode contextual information. The classification result achieved by an MLP for each pixel was a real value in the continuous range $[0,1]$. The result values were used to construct result images whereby each pixel in the image being processed was replaced by its classification result value in the result image constructed. This is shown in Fig. 7.19.

The classification results achieved based on this method are shown in Fig. 7.20.

7.10.2 Region-Based Classification

In a region-based classification scheme, an object is first segmented from an image. The object is then classified based on features that are usually determined to describe the whole object.

Extracted features from regions are shape, texture, intensity, size, position within the image, contextual features, and fractal characteristics. In other words, an object-level feature is computed over an entire object.

Subsequent to image segmentation, the individual detected regions have to be classified in terms of anatomical structures. A set of features from each region is extracted, and then the regions are classified based on the generated feature vectors. This procedure is explained in Fig. 7.21.

A common procedure is to first extract a larger feature set and then refine this, such that all classes of anatomical structures are correctly represented. An application of this

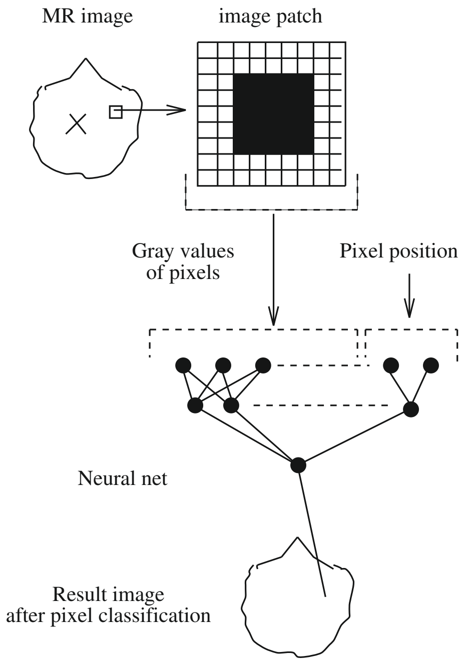

Figure 7.19 Pixel-based classification. An MLP is applied to the classification of all pixels in an image.

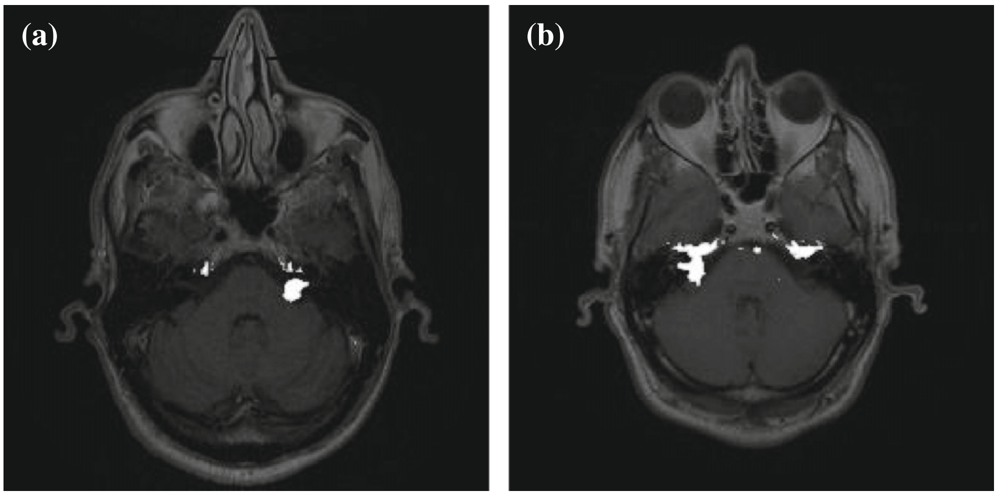

Figure 7.20 Pixel-level classification for detecting acoustic neuromas in MRI: (a) typical classification result; (b) fp-tumor adjacent to the acoustic neuroma [80]. *(Images courtesy of Prof. B. Thomas, University of Bristol.)*

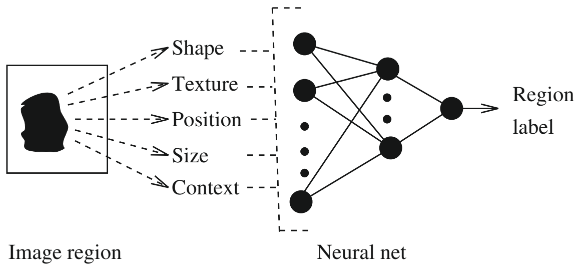

Figure 7.21 Region-based classification.

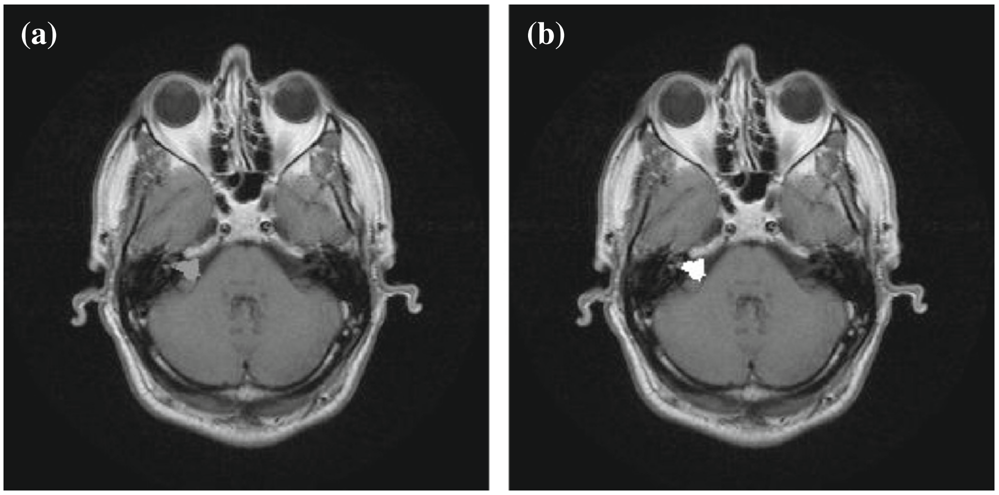

Figure 7.22 Region-level classification. A candidate tumor region cluster was erroneously classified as acoustic neuroma: (a) original image; and (b) classification result showing fp-tumor. *(Images courtesy of Prof. B. Thomas, University of Bristol.)*

method is visualized in Fig. 7.22. It shows a classification error whose structure is similar to that of an acoustic neuroma.

7.11. PERFORMANCE EVALUATION OF CLUSTERING TECHNIQUES

Determining the optimal number of clusters represents one of the most crucial classification problems. This task is known as cluster validity. The chosen validity function enables the validation of an accurate structural representation of the partition obtained by a clustering method. While a visual visualization of the validity is relatively simple for two-dimensional data, in case of multidimensional data sets this becomes very tedious. In this sense, the main objective of cluster validity is to determine the optimal number of clusters that provide the best characterization of a given multi-dimensional data set. An incorrect assignment of values to the parameter of a clustering algorithm results in a data partitioning scheme that is not optimal and thus leading to wrong decisions.

In this section, we evaluate the performance of the clustering techniques in conjunction with three cluster validity indices, namely Kim's index, Calinski Harabasz (CH) index, and the intraclass index. These indices were successfully applied before in biomedical time-series analysis [113]. In the following, we describe the above-mentioned indices.

Calinski Harabasz index [46]:This index is computed for m data points and K clusters as

$$\mathrm{CH} = \frac{[\mathrm{trace} B/(K-1)]}{[\mathrm{trace} W/(m-K)]} \tag{7.73}$$

where B and W represent the between and within cluster scatter matrices.

The maximum hierarchy level is used to indicate the correct number of partitions in the data.

Intraclass index [113]:This index is given as

$$I_W = \frac{1}{n}\sum_{k=1}^{K}\sum_{i=1}^{n_k} ||\mathbf{x}_i - \mathbf{w}_k||^2 \tag{7.74}$$

where n_k is the number of points in cluster k and $\mathbf{w}_k$ is a prototype associated with the kth cluster. I_W is computed for different cluster numbers. The maximum value of the second derivative of I_W as a function of cluster number is taken as an estimate for the optimal partition. This index provides a possible way of assessing the quality of a partition of K clusters.

Kim's index [176]:This index equals the sum of the over-partition $v_o(K, \mathbf{X}, \mathbf{W})$ and under-partition $v_u(K, \mathbf{X}, \mathbf{W})$ function measure

$$I_{Kim} = \frac{v_u(K) - v_{umin}}{v_{umax} - v_{umin}} + \frac{v_o(K) - v_{omin}}{v_{omax} - v_{omin}} \tag{7.75}$$

where $v_u(K)$ is the under-partitioned average over the cluster number of the mean intra-cluster distance and measures the structural compactness of each class, v_{umin} is its minimum while v_{umax} the maximum value. $v_u(K, \mathbf{X}, \mathbf{W})$ is given by the average of the mean intra-cluster distance over the cluster number K and measures the structural compactness of each and every class. $v_o(K, \mathbf{X}, \mathbf{W})$ is given by the ratio between the cluster number K and the minimum distance between cluster centers, describing intercluster separation. $\mathbf{X}$ is the matrix of the data points and $\mathbf{W}$ is the matrix of the prototype vectors. Similarly, $v_o(K)$ is the over-partitioned measure defined as the ratio between the cluster number and the minimum distance between cluster centers measuring the intercluster separation. v_{omin} is its minimum while v_{omax} the maximum value. The goal is to find the optimal cluster number with the smallest value of I_{Kim} for a cluster number $K = 2$ to K_{max}.

7.12. CLASSIFIER EVALUATION TECHNIQUES

There are several techniques for evaluating the classification performance in medical imaging problems. The most known are the confusion matrix, ranking order curves, and ROC curves.

7.12.1 Confusion Matrix

When conducting classification experiments, one possibility of evaluating the performance of a system is to determine the number of correctly and wrongly classified data. This gives a first impression of how correctly the classification was performed. In order to get a better understanding of the achieved classification, it is necessary to know which classes of data were most often misplaced. A convenient tool when analyzing results of classifier systems in general is the confusion matrix, which is a matrix containing information about the actual and predicted classes. The matrix is two-dimensional and has as many rows and columns as there are classes. The columns represent the true classifications and the rows represent the system classifications. If the system performs perfectly, there will be scores only in the diagonal positions. If the system has any misclassifications, these are placed in the off-diagonal cells. Table 7.3 shows a sample confusion matrix. Based on the confusion matrix, we can easily determine which classes are being confused with each other.

7.12.2 Ranking Order Curves

Ranking order curves represent a useful technique for estimating the appropriate feature dimension. They combine in a single plot several feature selection results for different classifiers where each of them visualized the function of error rate over the number of features.

Usually, we first see an improvement in classification performance when considering additional features. But after some time a saturation degree is reached, and the selection of new features leads to a deterioration of the classification performance because of overtraining. Based on ranking order curves, it is possible to determine the feature dependence. Some features alone do not necessarily deteriorate the classification performance. However, in combination with others they have a negative effect on the performance.

Table 7.3 Confusion matrix for a classification of three classes A, B, C.

	Output		
Input	***A*%**	***B*%**	***C*%**
A	90	0	10
B	0	99	1
C	0	90	10

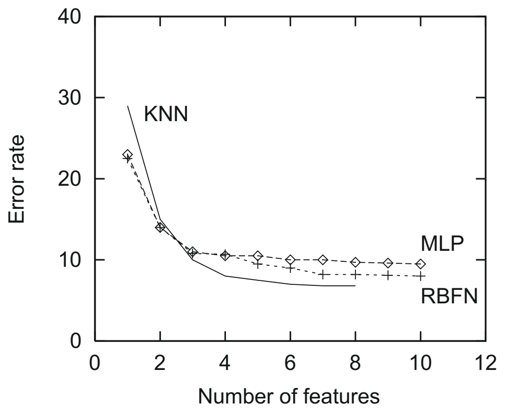

Figure 7.23 Example of ranking order curves showing feature selection results using three different classifiers (KNN, MLP, RBFN).

The classifier type plays an important role as well. In other words, an optimal performance can be achieved only as a combination of an appropriate classifier architecture and the best selected feature subset.

Ranking order curves are an important tool when it comes to determining potentially useful features, monitoring overtraining, and excluding redundant and computationally expensive features. This requires a vast number of simulations such that the ranking order of the available features becomes evident. Based on the determined histograms, it becomes possible to detect the redundant features so that in subsequent simulations they can be excluded. Figure 7.23 visualizes three feature ranking order curves for different classifiers.

7.12.3 ROC Curves

Receiver operating characteristic (ROC) curves have their origin in signal detection theory. Since the outcome of a particular condition in a yes-no signal detection experiment can be represented as an ordered pair of values (the hit and false-alarm rates), it is useful to have a way to graphically present and interpret them. This is usually done by plotting hit rate against false-alarm rate. Such a plot is called a receiver operating characteristic or ROC.

In pattern recognition ROC curves play an important role in providing information about the overlap between two classes. In many cases, the mean values of distinct classes may differ significantly, yet their variances are large enough to impede a correct class distinction.

Figure 7.24a illustrates the overlapping of two pdfs describing the distribution of a feature in two classes. The threshold in the figure is given by a perpendicular line. For a better understanding, one pdf is inverted as suggested in [364]. We will assume that all values on the left of the threshold belong to class ω_1, while those on the right belong

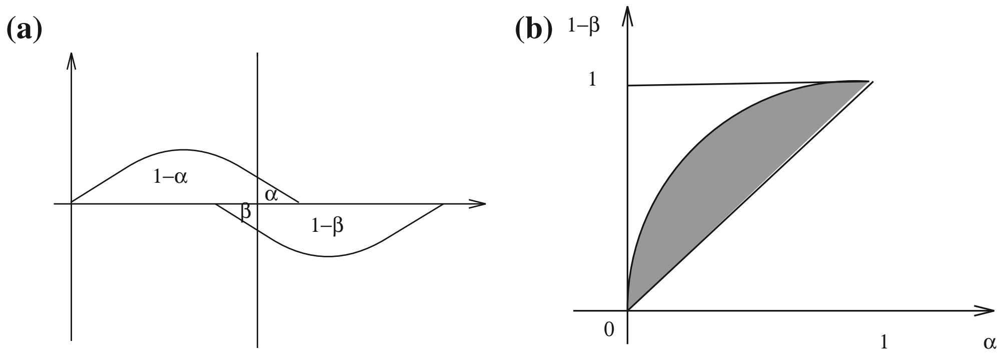

Figure 7.24 Example of (a) overlapping pdfs of the same feature in two classes and (b) the resulting ROC curve.

to class ω_2. However, this decision has an error probability of α while the probability of a correct decision is given by $1 - \alpha$. Let β be the probability of a wrong decision concerning class ω_2 while $1-\beta$ is the probability of a correct decision. The threshold can be moved in both directions, and this will lead for every position to different values for α and β. In case of perfect overlap of the two curves, for every single threshold position $\alpha = 1 - \beta$ holds. This corresponds to the bisecting line in Fig. 7.24b where the two axes are α and $1 - \beta$. If the overlap gets smaller, the corresponding curve in Fig. 7.24b moves apart from the bisecting line. In the ideal case, there is no overlap and we obtain $1-\beta = 1$. In other words, the less the class overlap, the larger the area between the curve and the bisecting line.

We immediately see that the area varies between zero for complete overlap and 1/2 (area of the upper triangle) for complete separation. This means we have been provided with a measure of the class separability of a distinct feature.

In practical problems, the ROC curve can be easily determined by moving the threshold and computing the correct and false classification rate over the available training vectors.

7.13. EXERCISES

1. Consider a biased input of the form

$$\tau_t(i) = \sum_k a_t(i) w_{ik} + b \tag{7.76}$$

and a sigmoidal activation function. What bias b is necessary for $f(0) = 0$? Is this acceptable? Are there any other possibilities?

2. A more general output activation is given by

$$o_j = f(\tau_j) = \frac{1}{1 + \exp -\{\frac{\tau_j - \theta_j}{\theta_0}\}} \tag{7.77}$$

   a) Explain the contribution of the parameter θ_j.
   b) Explain the role of θ_0.
3. For $f(\tau_j)$ given as in Exercise 4:
   a) Determine and plot $f'(\tau_j)$ for $\tau_j = 0$ and $\theta_0 = 5$.
   b) Repeat this for $\tau_j = 0$ and $\theta_0 = 50$ and $\theta_0 = 0.5$.
4. Show that if the output activation is given by

$$o_j = f(\tau_j) = \frac{1}{1 + \exp \tau_j} \tag{7.78}$$

   then we obtain for its derivative

$$\frac{\partial f(\tau_j)}{\partial \tau_j} = f'(\tau_j) = o_j(1 - o_j) \tag{7.79}$$

   Is it possible to have a τ_j such that we obtain $f(\tau_j) = 0$?
5. Design a feedforward network such that for given binary inputs A and B it implements
   a) The logical NAND function.
   b) The logical NOR function.
6. The necessary number of neurons in a hidden layer is always problem-oriented. Develop and discuss the advantages of removing or adding iteratively neurons from the hidden layer.
7. A method to increase the rate of learning yet to avoid the instability is to modify the weight updating rule

$$w_{ij}(n) = w_{ij}(n-1) + \eta \delta_{hj} p_i^t \tag{7.80}$$

   by including a momentum term as described in [71]

$$\Delta w_{ij}(n) = \alpha \Delta w_{ij}(n-1) + \eta \delta_{hj} p_i^t \tag{7.81}$$

   where α is a positive constant called the momentum constant. Describe why this is the case.
8. The momentum constant is in most cases a small number with $0 \leq \alpha < 1$. Discuss the effect of choosing a small negative constant with $-1 < \alpha \leq 0$ for the modified weight updating rule given in Exercise 7.
9. Set up two data sets, one set for training an MLP, and the other one for testing the MLP. Use a single-layer MLP and train it with the given data set. Use two possible nonlinearities: $f(x) = \sin x$ and $f(x) = e^{-x}$. Determine for each of the given nonlinearities:

   a) The computational accuracy of the network by using the test data.
   b) The effect on network performance of varying the size of the hidden layer.

10. Consider the updating rule for the Kohonen map given by eq. (7.16). Assume that the initial weight vectors $\mathbf{w}_j$ are not different. Comment on the following two distinct cases:
   a) They are nonrandom but distinct from each other.
   b) They are random but some weights can be the same.

11. In [183] it is stated that if the input patterns $\mathbf{x}$ have as a pdf a given $p(\mathbf{x})$, then the point density function of the resulting weight vectors $\mathbf{w}_j$ approximates $p(\mathbf{x})$. Describe based on eq. (7.16) why this is the case.

12. Comment on the differences and similarities between the Kohonen map and the "neural gas" network.

13. When is the Kohonen map preserving the topology of the input space? What condition must the neural lattice fulfill?

14. Consider a Kohonen map performing a mapping from a 2-D input onto a 1-D neural lattice of 60 neurons. The input data are random points uniformly distributed inside a circle of radius 1 centered at the origin. Compute the map produced by the neural network after 0, 25, 100, 1000, and 10,000 iterations.

15. Consider a Kohonen map performing a mapping from a 3-D input onto a 2-D neural lattice of 1,000 neurons. The input data are random points uniformly distributed inside a cube defined by $\{(0 < x_1 < 1), (0 < x_2 < 1), (0 < x_3 < 1)\}$. Compute the map produced by the neural network after 0, 100, 1,000, and 10,000 iterations.

16. Radial basis neural networks are often referred to as "hybrid" neural networks. Comment on this property by taking into account the architectural and algorithmic similarities among radial basis neural networks, Kohonen maps, and MLPs.

17. Find a solution for the XOR problem using an RBF network with two hidden units where the two radial basis function centers are given by $\mathbf{m}_1 = [1, 1]^T$ and $\mathbf{m}_2 = [0, 0]^T$. Determine the output weight matrix $\mathbf{W}$.

18. Find a solution for the XOR problem using an RBF network with four hidden units where four radial basis function centers are given by $\mathbf{m}_1 = [1, 1]^T$, $\mathbf{m}_2 = [1, 0]^T$, $\mathbf{m}_3 = [0, 1]^T$, and $\mathbf{m}_4 = [0, 0]^T$. Determine the output weight matrix $\mathbf{W}$.