16

Modular Neural Networks

In the previous chapters we have discussed different models of neural networks – linear, recurrent, supervised, unsupervised, self-organizing, etc. Each kind of network relies on a different theoretical or practical approach. In this chapter we investigate how those different models can be combined. We transform each single network in a module that can be freely intermixed with modules of other types. In this way we arrive at the concept of *modular neural networks*. Several general issues have led to the development of modular systems [153]:

- *Reducing model complexity.* As we discussed in Chap. 10 the way to reduce training time is to control the degrees of freedom of the system.
- *Incorporating knowledge.* Complete modules are an extension of the approach discussed in Sect. 10.3.5 of learning with hints.
- *Data fusion and prediction averaging.* Committees of networks can be considered as composite systems made of similar elements (Sect. 9.1.6)
- *Combination of techniques.* More than one method or class of network can be used as building block.
- *Learning different tasks simultaneously.* Trained modules may be shared among systems designed for different tasks.
- *Robustness and incrementality.* The combined network can grow gradually and can be made fault-tolerant.

Modular neural networks, as combined structures, have also a biological background: Natural neural systems are composed of a hierarchy of networks built of elements specialized for different tasks. In general, combined networks are more powerful than flat unstructured ones.

16.1 Constructive algorithms for modular networks

Before considering networks with a self-organizing layer, we review some techniques that have been proposed to provide structure to the hidden layer of feed-forward neural networks or to the complete system (as a kind of decision tree).

16.1.1 Cascade correlation

An important but difficult problem in neural network modeling, is the selection of the appropriate number of hidden units. The *cascade correlation algorithm*, proposed by Fahlman and Lebiere [131], addresses this issue by recruiting new units according to the residual approximation error. The algorithm succeeds in giving structure to the network and reducing the training time necessary for a given task [391].

Figure 16.1 shows a network trained and structured using cascade correlation. Assume for the moment that a single output unit is required. The algorithms starts with zero hidden units and adds one at a time according to the residual error. The diagram on the right in Figure 16.1 shows the start configuration. The output unit is trained to minimize the quadratic error. Training stops when the error has leveled off. If the average quadratic error is still greater than the desired upper bound, we must add a hidden unit and retrain the network.

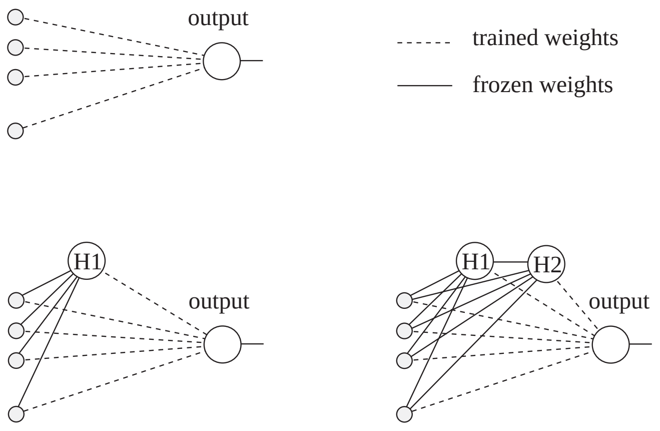

Fig. 16.1. The cascade correlation architecture

Denote the mean error of the network by $\bar{E}$ and the error for pattern $i = 1, 2, \ldots, p$, by E_i. A hidden unit is trained, isolated from the rest of the network, to *maximize* the absolute value of the correlation between $V_i - \bar{V}$ and $E_i - \bar{E}$, where V_i denotes the unit's output for the i-th pattern and $\bar{V}$ its average output. The quantity to be maximized is therefore

$$S = |\sum_{i=1}^{p}(V_i - \hat{V})(E_i - \hat{E})|. \qquad (16.1)$$

The motivation behind this step is to specialize the hidden unit to the detection of the residual error of the network. The backpropagation algorithm is applied as usual, taking care to deal with the sign of the argument of the absolute value operator. One way of avoiding taking absolute values at all, is to train a weight at the output of the hidden unit, so that it assumes the appropriate sign (see Exercise 1).

Once the hidden unit H1 has been trained, i.e., when the correlation level cannot be further improved, the unit is added to the network as shown in Figure 16.1 (left diagram). The weights of the hidden unit are frozen. The output unit receives information now from the input sites and from the hidden unit H1. All the weights of the output unit are retrained until the error levels off and we test if a new hidden unit is necessary. Any new hidden unit receives an input from the input sites and from all other previously defined hidden units. The algorithm continues adding hidden units until we are satisfied with the approximation error.

The advantage of the algorithm, regarding learning time, is that in every iteration a single layer of weights has to be trained. Basically, we only train one sigmoidal unit at a time. The final network has more structure than the usual flat feed-forward networks and, if training proceeds correctly, we will stop when the minimum number of necessary hidden units has been selected. To guarantee this, several hidden units can be trained at each step and the one with the highest correlation selected for inclusion in the network [98].

In the case of more than one output unit, the average of the correlation of all output units is maximized at each step. A summation over all output units is introduced in equation (16.1).

16.1.2 Optimal modules and mixtures of experts

Cascade correlation can be considered a special case of a more general strategy that consists in training special modules adapted for one task. As long as they produce continuous and differentiable functions, it is possible to use them in any backpropagation network. The learning algorithm is applied as usual, with the parameters of the modules frozen.

A related approach used in classifier networks is creating optimal independent two-class classifiers as building blocks for a larger network. Assume that we are trying to construct a network that can classify speech data as corresponding to one of 10 different phonemes. We can start training hidden units that learn to separate one phoneme from another. Since the number of different pairs of phonemes is 45, we have to train 45 hidden units. Training proceeds rapidly, then the 45 units are put together in the hidden layer. The output units (which receive inputs from the hidden units and from the original data) can be trained one by one. As in cascade correlation, we do not need to train more than a single layer of weights at each step. Moreover, if a new category is introduced, for example a new phoneme, we just have to train new hidden units for this new class and those already existing, add the new units to the hidden layer, and retrain the output layer.

Note that the classifiers constructed using this approach still satisfy the conditions necessary to potentially transform them into probability estimators. The proof of Proposition 13 is completely general: no requirements are put on the form of the classifier network other than enough plasticity.

A more general framework is the one proposed by Jacobs and Jordan with their *mixture of experts* approach [218]. Given a training set consisting of p pairs $(\mathbf{x}_1, \mathbf{y}_1) \ldots, (\mathbf{x}_p, \mathbf{y}_p)$, the complete network is built using several expert subnetworks. Each one of them receives the input $\mathbf{x}$ and generates the output $\mathbf{y}$ with probability

$$P(\mathbf{y}|\mathbf{x}, \theta_i)$$

where θ_i is the parameter vector of the i-th expert. The output of each expert subnetwork is weighted by a gating subnetwork which inspects the input $\mathbf{x}$ and produces for m expert subnetworks the set of gating values $g_1, g_2, \ldots, g_m$. They represent the probability that each expert network is producing the correct result. The final probability of producing $\mathbf{y}$ is given by the product of each expert's evaluation and its corresponding gating weight g_i. The parameters of the combined model can be found by gradient descent or using an Expectation Maximization (EM) algorithm proposed by Jordan and Jacobs [227]. The mixtures of experts can be organized in different levels, like an optimal decision tree. The resulting architecture is called a *Hierarchical Mixture of Experts* (HME).

16.2 Hybrid networks

In this section we consider some of the more promising examples of combined networks, especially those which couple a self-organizing with a feed-forward layer.

16.2.1 The ART architectures

Grossberg has proposed several different hybrid models, which should come closer to the biological paradigm than pure feed-forward networks or standard associative memories. His best-known contribution is the family of ART architectures (*Adaptive Resonance Theory*). The ART networks classify a stochastic sequence of vectors in clusters. The dynamics of the network consists of a series of automatic steps that resemble learning in humans. Specially phenomena like one-shot learning can be recreated with this model.

Figure 16.2 shows the task to be solved by the network. The weight vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$ represent categories in input space. All vectors located inside the cone around each weight vector are considered members of a specific cluster. The weight vectors are associated with computing units in a network.

Each unit fires a 1 only for vectors located inside its associated cone of radius r. The value r is inversely proportional to the "attention" parameter of the unit. If r is small, the classification of input space is fine grained. A large r produces a classification with low granularity, that is, with large clusters. The network is trained to find the weight vectors appropriate for a given data set.

Once the weight vectors have been found, the network computes whether new data can be classified in the existing clusters. If this is not the case (if, for example, a new vector is far away from any of the existing weight vectors) a new cluster is created, with a new associated weight vector. Figure 16.2 shows the weight vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ with their associated cones of radius r. If the new vector $\mathbf{w}_3$ is presented to the network, it is not recognized and a new cluster is created, with this vector in the middle. The network preserves its *plasticity*, because it can always react to unknown inputs, and its *stability*, because already existing clusters are not washed out by new information. For the scheme to work, enough potential weight vectors, i.e., computing units, must be provided.

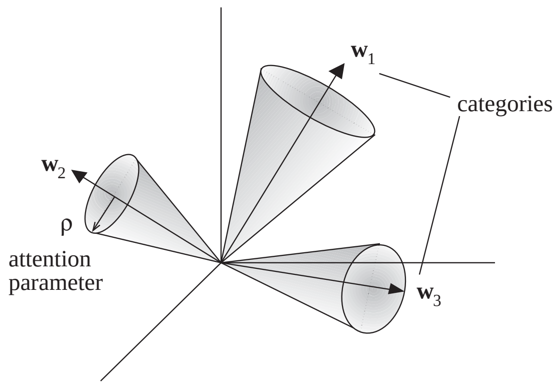

Fig. 16.2. Vector clusters and attention parameters

The ART-1 architecture was proposed by Grossberg to deal with binary vectors [168]. It was generalized, as ART-2, to real-valued vectors [81]. Later it was extended into the ART-3 model, whose dynamics is determined by differential equations similar to the ones involved in the description of chemical signals [82]. All three basic models have a similar block structure and the classification strategy is in principle the same. In what follows, we give only a simplified description of the ART-1 model. Some details have been omitted from the diagram to simplify the discussion.

Figure 16.3 shows the basic structure of ART-1. There are two basic layers of computing units. Layer F_2 contains elements which fire according to the "winner-takes-all" method. Only the element receiving the maximal scalar product of its weight vector and the input fires a 1. In our example, the

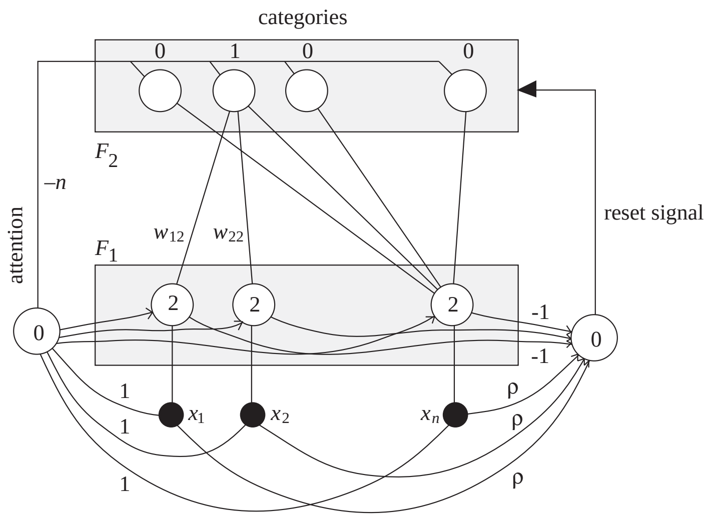

Fig. 16.3. The ART-1 architecture

second element from the left in the F_2 layer has just produced the output 1 and the other units remain silent. All weight vectors of the units in the F_2 layer are equal at the beginning (all components set to 1) and differentiate gradually as learning progresses.

Layer F_1 receives binary input vectors from the input sites. The units here have all threshold 2. This means that the attention unit (left side of the diagram) can only be turned off by layer F_2. As soon as an input vector arrives it is passed to layer F_1 and from there to layer F_2. When a unit in layer F_2 has fired, the negative weight turns off the attention unit. Additionally, the winning unit in layer F_2 sends back a 1 through the connections between layer F_2 and F_1. Now each unit in layer F_1 becomes as input the corresponding component of the input vector $\mathbf{x}$ and of the weight vector $\mathbf{w}$. The i-th F_1 unit compares x_i with w_i and outputs the product $x_i w_i$. The reset unit receives this information and also the components of $\mathbf{x}$, weighted by ρ, the attention parameter, so that it own computation is

$$\rho \sum_{i=1}^{n} x_i - \mathbf{x} \cdot \mathbf{w} \geq 0.$$

This corresponds to the test

$$\frac{\mathbf{x} \cdot \mathbf{w}}{\sum_{i=1}^{n} x_i} \leq \rho.$$

The reset unit fires only if the input lies outside the attention cone of the winning unit. A reset signal is sent to layer F_2, but only the winning unit is inhibited. This in turn activates again the attention unit and a new round of computation begins.

As we can see there is *resonance* in the network only if the input lies close enough to a weight vector $\mathbf{w}$. The weight vectors in layer F_2 are initialized with all components equal to 1 and ρ is selected to satisfy $0 < \rho < 1$. This guarantees that eventually an unused vector will be recruited to represent a new cluster. The selected weight vector $\mathbf{w}$ is updated by pulling it in the direction of $\mathbf{x}$. This is done in ART-1 by turning off all components in $\mathbf{w}$ which are zeroes in $\mathbf{x}$. In ART-2 the update step corresponds to a rotation of $\mathbf{w}$ in the direction of $\mathbf{x}$.

The purpose of the reset signal is to inhibit all units that do not resonate with the input. A unit in layer F_2, which is still unused, can be selected for the new cluster containing $\mathbf{x}$. In this way, a single presentation of an example sufficiently different from previous data, can lead to a new cluster. Modifying the value of the attention parameter ρ we can control how many clusters are used and how wide they are.

In this description we have assumed that the layer F_2 can effectively compute the angular distance between $\mathbf{x}$ and $\mathbf{w}$. This requires a normalization of $\mathbf{w}$, which can be made in substructures of layers F_1 or F_2, like for example a so-called *Grossberg layer* [168, 169].

The dynamics of the ART-2 and ART-3 models is governed by differential equations. Computer simulations consume too much time. Consequently, implementations using analog hardware or a combination of optical and electronic elements are more suited to this kind of model [462].

16.2.2 Maximum entropy

The strategy used by the ART models is only one of the many approaches that have been proposed in the literature for the clustering problem. It has the drawback that it tries to build clusters of the same size, independently of the distribution of the data. A better solution would be to allow the clusters to have varying radius (attention parameters in Grossberg terminology). A popular approach in this direction is the maximum entropy method.

The entropy H of a data set of N points assigned to k different clusters $c_1, c_2, \ldots, c_k$ is given by

$$H = -\sum_{i=1}^{k} p(c_i) \log(p(c_i)),$$

where $p(c_i)$ denotes the probability of hitting the i-th cluster, when an element of the data set is picked at random. If all data is assigned to one cluster, then $H = 0$. Since the probabilities add up to 1, the clustering that maximizes the entropy is the one for which all cluster probabilities are identical. This means that the clusters will tend to cover the same number of points.

A problem arises whenever the number of elements of each class is different in the data set. Consider the case of unlabeled speech data: some phonemes are more frequent than others and if a maximum entropy method is used, the boundaries between clusters will deviate from the natural solution and classify some data erroneously. Figure 16.4 illustrates this problem with a simple example of three clusters. The natural solution seems obvious; the maximum entropy partition is the one shown.

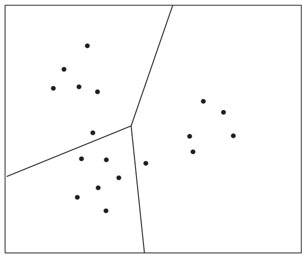

Fig. 16.4. Maximum entropy clustering

This problem can be solved using a *bootstrapped* iterative algorithm, consisting of the following steps:

Algorithm 16.2.1 *Bootstrapped clustering*

cluster: Compute a maximum entropy clustering with the training data. Label the *original data* according to this clustering.

select: Build a new training set by selecting from each class the same number of points (random selection with replacement). Go to the previous step.

The training set for the first iteration is the original data set. Using this algorithm, the boundaries between clusters are adjusted irrespective of the number of members of each actual cluster and a more natural solution is found.

16.2.3 Counterpropagation networks

Our second example of a hybrid model is that of counterpropagation networks, first proposed by Hecht-Nielsen [183]. In a certain sense, they are an extension of Kohonen's approach. The original counterpropagation networks were designed to approximate a continuous mapping f and its inverse f^{-1} using two symmetric sections. We describe the essential elements needed to learn an approximation of the mapping f.

Figure 16.5 shows the architecture of a counterpropagation network. The input consists of an n-dimensional vector which is fed to a a hidden layer consisting of h cluster vectors. The output layer consists of a single linear associator (when learning a mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}$). The weights between hidden layer and output unit are adjusted using supervised learning.

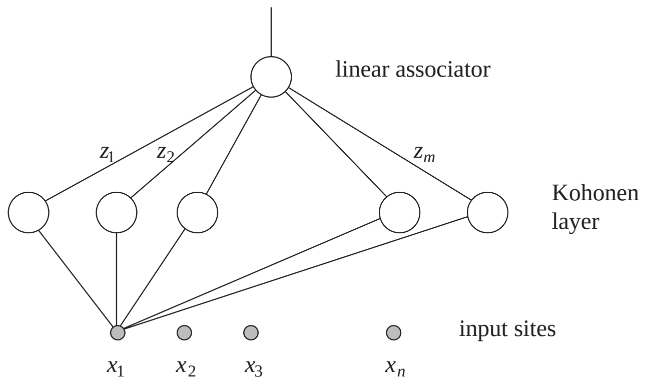

Fig. 16.5. Simplified counterpropagation network

The network learns in two different phases. Firstly, the hidden layer is trained using stochastically selected vectors from input space. The hidden layer produces a clustering of input space that corresponds to an n-dimensional Voronoi tiling (as shown in Figure 16.6). After this step, each element of the hidden layer has specialized to react to a certain region of input space. Note that the training strategy can be any of the known vector quantization methods and that the hidden layer can be arranged in a grid (Kohonen layer) or can consist of isolated elements. The output of the hidden layer can be controlled in such a way that only the element with the highest activation fires. The hidden layer is, in fact, a classification network.

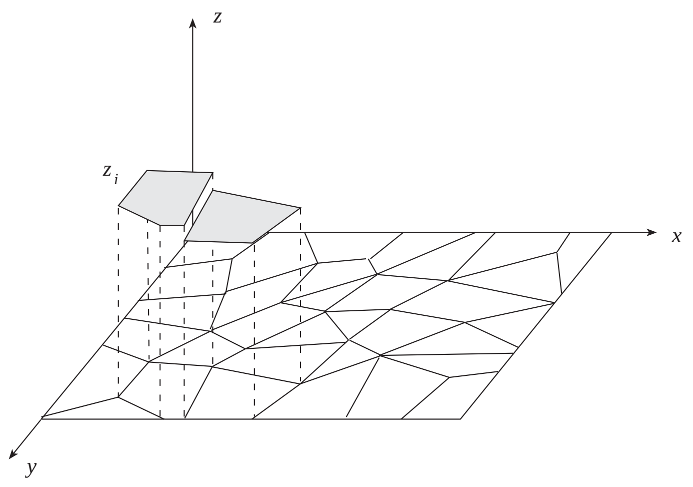

Fig. 16.6. Function approximation with a counterpropagation network

Function approximation can be implemented by assigning a value to each weight $z_1, z_2, \ldots, z_m$ in the network of Figure 16.5. This corresponds to fixing the value of the approximation for each cluster region, as shown in Figure 16.6. The polygon with height z_i in the diagram corresponds to the function approximation selected for this region. The second training step is the supervised adjustment of the z_i values. If hidden unit i fires when input vector $\mathbf{x}$ is presented, the quadratic error of the approximation is

$$E = \frac{1}{2}(z_i - f(\mathbf{x}))^2$$

Straightforward gradient descent on this cost function provides the necessary weight update

$$\Delta z_i = -\frac{dE}{dz_i} = \gamma(f(\mathbf{x}) - z_i),$$

where γ is a learning constant. After several iterations of supervised learning we expect to find a good approximation of the function f. Training of the hidden and the output layer can be intermixed, or can be made in sequence. The counterpropagation network can be trivially extended to the case of several output units.

16.2.4 Spline networks

The function approximation provided by a network with a clustering layer can be improved by computing a local linear approximation to the objective function. Figure 16.7 shows a network which has been extended with a hidden layer of linear associators. Each cluster unit is used to activate or inhibit one of the linear associators, which are connected to all input sites.

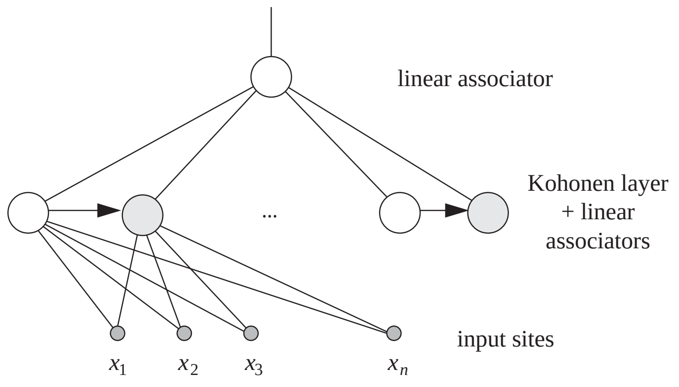

Fig. 16.7. Extended counterpropagation network

The clustering elements in the hidden layer are trained exactly as before. Figure 16.8 shows the final Voronoi tiling of input space. Now, the linear associators in the hidden layer are trained using the backpropagation algorithm. For each selected input, exclusively one of the clustering units activates one associator. Only the corresponding weights are trained. The net effect is that the network constructs a set of linear approximations to the objective function that are locally valid.

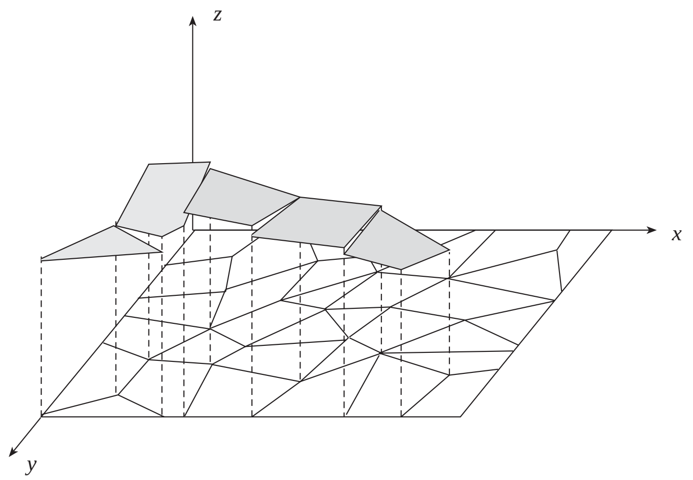

Fig. 16.8. Function approximation with linear associators

Figure 16.8 shows that the constructed approximation consists now of differently oriented polygons. The new surface has a smaller quadratic error than the approximation with horizontal tiles. The functional approximation can be made more accurate just by adding new units to the hidden layer, refining the Voronoi tiling in this way. The network can be trivially extended to the case of more than one output unit.

Ritter has recently proposed to use splines as construction elements for the function approximation. This is a generalization of the above approach, which seems to require many fewer hidden units in the case of well-behaved functions. Such *spline networks* have been applied to different pattern recognition problems and in robotics [439].

A problem that has to be addressed when combining a self-organized hidden layer with a conventional output layer, is the interaction between the two types of structures. Figure 16.9 shows an example in which the function to be learned is much more variable in one region than another. If input space is sampled homogeneously, the mean error at the center of the domain will be much larger than at the periphery, where the function is smoother.

The general solution is sampling input space according to the variability of the function, that is, according to its gradient. In this case the grid defined by, for example, a two-dimensional Kohonen network is denser at the center than

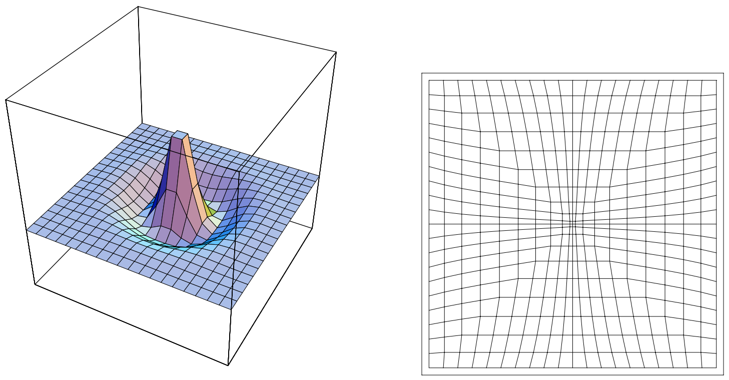

Fig. 16.9. A function and its associated differential tiling of input space

at the periphery (Figure 16.9), right). However, since only one set of training pairs is given, heuristic methods have to be applied to solve this difficulty (see Exercise 4).

16.2.5 Radial basis functions

A variant of hybrid networks with a Kohonen or clustering layer consists in using Gaussians as activation function of the units. The hidden layer is trained in the usual manner, but the input is processed differently at each hidden unit. All of them produce an output value which is combined by the linear associator at the output. It is desirable that the unit whose weight vector lies closer to the input vector fires more strongly than the other hidden units.

Moody and Darken [318] propose the architecture shown in Figure 16.10. Each hidden unit j computes as output

$$g_j(x) = \frac{\exp\left((\mathbf{x}-\mathbf{w}_j)^2/2\sigma_j^2\right)}{\sum_k \exp\left((\mathbf{x}-\mathbf{w}_k)^2/2\sigma_k^2\right)}$$

where $\mathbf{x}$ is the input vector and $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$ are the weight vectors of the hidden units. The constants σ_i are selected in an ad hoc way and can be set to the distance between the weight vector $\mathbf{w}_i$ and its nearest neighbor.

Note that the output of each hidden unit is normalized by dividing it with the sum of all other hidden outputs. This explains the horizontal connections between units shown in Figure 16.10. The output of the hidden layer consists therefore of normalized numbers.

The weights $z_1, z_2, \ldots, z_m$ are determined using backpropagation. The quadratic error is given by

$$E = \frac{1}{2}(\sum_i^n g_i(\mathbf{x})z_i - f(\mathbf{x}))^2.$$

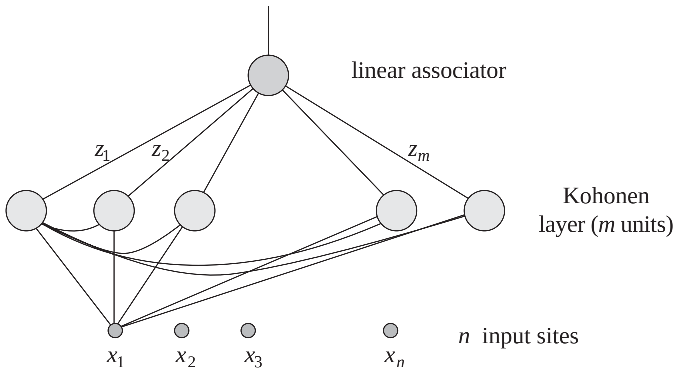

Fig. 16.10. Hybrid network with RBFs

The necessary weight updates are therefore given by

$$\Delta z_i = -\frac{dE}{dz_i} = \gamma g_i(\mathbf{x})(f(\mathbf{x}) - \sum_i^n g_i(\mathbf{x})z_i).$$

The mixture of Gaussians provides a continuous approximation of the objective function and makes unnecessary the computation of the maximum excitation in the hidden layer. Figure 16.11 shows an example of a function approximation with four Gaussians. The error can be minimized using more hidden units.

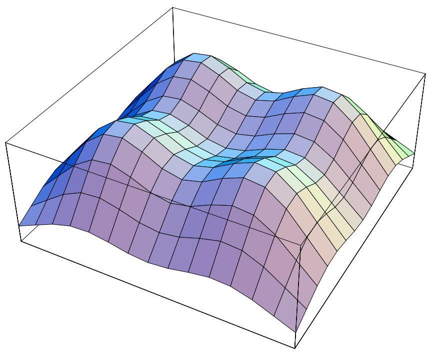

Fig. 16.11. Function approximation with RBFs

The main difference between networks made of radial basis functions and networks of sigmoidal units is that the former use locally concentrated functions as building blocks whereas the latter use smooth steps. If the function to be approximated is in fact a Gaussian, we need to arrange several sigmoidal steps in order to delimit the region of interest. But if the function to be approximated is a smooth step, we need many Gaussians to cover the region of interest. Which kind of activation function is more appropriate therefore depends on the concrete problem at hand.

16.3 Historical and bibliographical remarks

To the extent that neural network research has been approaching maturity, more sophisticated methods have been introduced to deal with complex learning tasks. Modular neural networks were used from the beginning, for example in the classic experiments of Rosenblatt. However, the probabilistic and algorithmic machinery needed to handle more structured networks has been produced only in the last few years.

The original motivation behind the family of ART models [168] was investigating phenomena like short-term and long-term memory. ART networks have grown in complexity and combine several layers or submodules with different functions. They can be considered among the first examples of the modular approach to neural network design.

Feldman et al. have called the search for algorithmic systems provided with the "speed, robustness and adaptability typically found in biology" the "grand challenge" of neural network research [137]. The path to its solution lies in the development of *structured connectionist architectures*. Biology provides here many sucessful examples of what can be achieved when modules are first found and then combined in more complex systems.

Ultimately, the kind of combined systems we would like to build are those in which symbolic rules are implemented and supported by a connectionist model [135, 136]. There have been several attempts to build connectionist experts systems [151] and also connectionist reasoning systems [400]. A modern example of a massively parallel knowledge based reasoning system is SHRUTI, a system proposed by Shastri, and which has been implemented in parallel hardware [289].

Exercises

1. Show how to eliminate the absolute value operation from equation (16.1) in the cascade correlation algorithm, so that backpropagation can be applied as usual.
2. Compare the cascade correlation algorithm with standard backpropagation. Use some small problems as benchmarks (parity, decoder-encoder, etc.).
3. Construct a counterpropagation network that maps the surface of a sphere onto the surface of a torus (represented by a rectangle). Select the most

appropriate coordinates. How must you train the network so that angles are preserved? This is the Mercator projection used for navigation.

4. Propose a method to increase sampling in those regions of input space in which the training set is more variable, in order to make a counterpropagation network more accurate.