Part III
Multilayer Networks

7 Counter-Propagation

learning objectives

- the first example of a network having more than one layer
- how supervised learning is used when you have a set of "correct" answers
- how data flows through a "counter-propagation" network: the correct answers flow backwards
- content-dependent data storage and retrieval ("associative memory")
- lookup tables and models, and how neural nets can serve as either
- how input data are prepared by "normalization"
- application of counter-propagation to simulate a simple tennis match
- usefulness of counter-propagation for creating lookup tables

7.1 Transition from One to Two Layers

The neural networks we have discussed up to now have had only one layer of neurons. The input signal is transferred to the active neuron layer via the input net, but this layer has more or less only a formal meaning. From a design point of view, it is easy to connect one neuron layer to a lower one, this one to still another one and so on. What we need is only to determine the number of neurons in one layer and the way the outputs of the neurons in this layer are connected to the synapses (weights) in the layer below.

The neurons of two layers can be fully, partially or randomly connected (Figure 7-1). *Full* connection means that each neuron in one layer is connected to all the neurons in the layer below. This is the

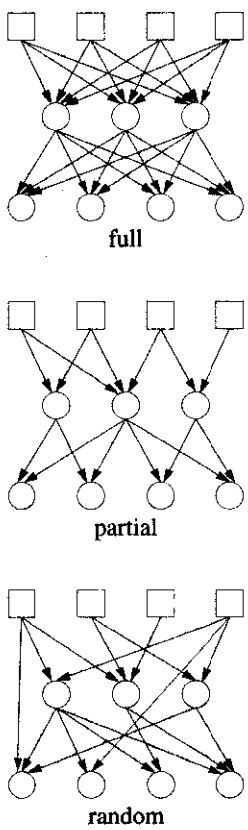

Figure 7-1: Connection of neurons among layers: full, partial and random.

most common scheme used in artificial neural networks. *Partial* and *random* connection are self-explanatory. The partial scheme is used when some aspect of the problem suggests it. When the neurons are connected randomly, the concept of layers becomes meaningless; the designer can determine only the average number of neurons to be connected between the two layers.

Once the links are made and the weights are determined, the signals can flow from the input through many layers towards the output very quickly. This calculation is a natural application for parallel computers, because each neuron processes data independently of all the others; hence (if enough processors are available), the calculations in one layer can be done simultaneously, i.e., in the time it takes to evaluate **one** signal on a sequential computer.

While it is easy to connect the layers of neurons among themselves, determining the appropriate weights for a given problem is much more difficult.

In the counter-propagation approach to modifying weights, the known answers will be sent towards the inputs back through the network to correct the weights of the neuron. As in all supervised learning schemes, the weights are adapted by comparing the actual output with an ideal output. The output of the counter-propagation network is not obtained from the weights of **one** output **neuron** (as in a Kohonen network) or as an output vector from **all neurons** (as in a back-propagation network – Chapter 8). Rather, the output is taken from **all weights** between one, the winning neuron of the Kohonen network, and all output neurons.

7.2 Lookup Table

Due to the nature of counter-propagation network learning, which will be explained in detail later on, we can treat the trained counter-propagation network as a lookup table (we can call it a multi-dimensional spreadsheet as well), which is an area of memory where the answers to certain complex questions are ordered in such a manner that we can readily find the "box" with the right answer.

For example, calculating the sine or cosine of an angle can be very time-consuming when it occurs within deeply nested loops; scientific programs often store values of the sine, tabulated at some convenient interval, and use the angle argument as an **index** to retrieve the proper value from this lookup table.

Thus, we calculate an **address** rather than a value from the input data. There are other situations, too, where lookup table is useful, for example, cases where the value to be retrieved is only a weak function of the input (i.e., where there is a wide degree of tolerance in the answers) or (which is equivalent to this) when corrupted input has to be used. We have already discussed the problem of retrieving the answer to a corrupted input in Chapters 4 and 5; the procedure for finding the sought answer on the basis of imperfect or fuzzy data is called content-dependent retrieval. It requires that **only similar** inputs cause a given box to be selected (the definition of "similar" being up to the user).

There are other *content-dependent retrieval* methods (e.g., the hash algorithm: Reference 7-8, or three-distance clustering: Reference 7-9). See the literature for a detailed discussion.

The concept of the lookup table is distinct from the concept of a *model*, where the goal is to obtain a "function" or a procedure that will yield an answer that is different for each different set of variables; for example, instead of obtaining the sine of an angle by lookup table, we might call the sine subprogram, which approximates (models) the actual value by a series expansion. (Figure 7-2).

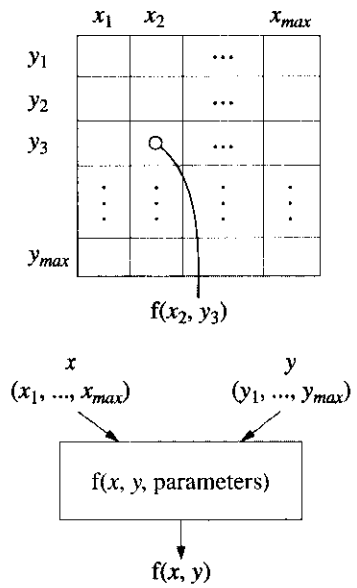

Figure 7-2: The difference between a lookup table and a model. X can have values between x_1 and x_{max} and y between y_1 and y_{max}.

These two methods differ not only in application goals, but in the realm of data to which they can be applied. Modeling typically requires only a few data (the sine, for example, requires only an angle argument and the coefficients of the series expansion), whereas a lookup table requires much more (depending on the resolution, maybe several hundred sine values).

That is, in constructing (or training) a model, the number of experiments needs to be larger than the number of parameters describing the model. However, a lookup table requires **at least one** experiment for each possible or expected answer. In order to fill all the boxes with information, a relatively dense distribution of input data (experiments) over the entire variable space is required.

The counter-propagation network is not well suited for modeling; other techniques, such as "back-propagation of errors" (see the next Chapter), can be employed much more efficiently. Counter-propagation is best used to generate lookup tables, where **all** the required answers (within some limits) are known in advance.

The larger the lookup table or the better the model, the smaller are the differences between the ideal and the supplied answers. However, a model can, in principle, yield an infinite number of answers, while a

lookup table can only provide as many answers as it has boxes. The counter-propagation lookup table is optimum if some of the input variables are missing, or if the input data are vague or very fuzzy. Remember that, in general, counter-propagation lookup tables are very robust, while models, especially those simulating high-order polynomial multivariate functions, can show unstable behavior for corrupted or fuzzy input data.

7.3 Architecture

The counter-propagation neural network will be our first example of a neural network architecture which has **two** active layers of neurons: a Kohonen layer and an output layer (Figure 7-3.). The inputs are *fully* connected to the Kohonen network, where competitive learning (Chapter 6) is performed; that is, each unit in the input layer is linked to all neurons in the Kohonen layer.

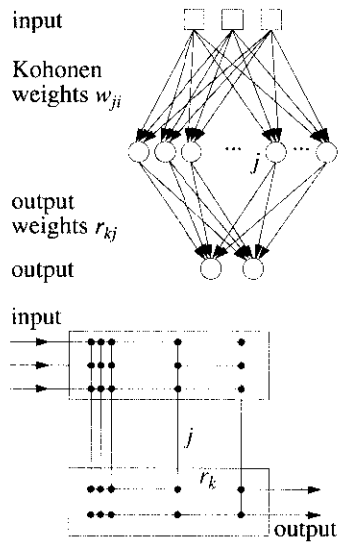

Figure 7-3: The layout of a counter-propagation network.

We will label the weights connecting the input unit i with the Kohonen neuron j as w_{ji}; each neuron in the Kohonen layer is described by a weight vector $\boldsymbol{W}_j$.

The neurons of the Kohonen layer are in turn connected to the neurons in the output layer. In principle, this is complete (full) connection. In practice, however, after each input, only a certain neighborhood of a given neuron is connected to the output neurons (Figure 7-4), and only the weights linking these neurons are allowed to change. The weights connecting the j-th neuron in the Kohonen layer with the k-th neuron in the output layer are labeled r_{kj}, and the weight vector belonging to a given output neuron is labeled $\boldsymbol{R}_k$.

The goal of constructing a lookup table immediately sets requirements on the architecture of the counter-propagation network. A good way to store numerous sets of answers is as the weights of the output neurons that receive signals from the Kohonen layer (Figure 7-4).

> A given answer is **not** stored as a set of weights in one neuron, but holographically, as **one** component of the weights of **all** the output neurons.

Such an organization requires the number of neurons in the Kohonen layer to be equal to the number of answers we would like to

store, and the number of neurons in the output layer to be equal to the number of variables comprising the output answer.

For example, one thousand answers, each consisting of four variables, requires a Kohonen network with one thousand neurons, and an output layer with four. The input layer should have the same number of units as there are input variables.

Because the counter-propagation network is described in the literature as requiring normalized input data, the standard architectures are shown with an additional normalizing neuron, which provides a position for an extra input (a normalizing variable comparable to the bias); this is determined in such a way that the magnitude of the new, augmented input vector $\boldsymbol{X}$ is equal to unity (Figure 7-5):

$$\boldsymbol{X} = (x_1, x_2, ..., x_m) \rightarrow \boldsymbol{X}'(x_1, x_2, ..., x_m, x_{m+1})$$

and

$$\|X'\| = \sqrt{\left(x_1^2 + x_2^2 + ... + x_m^2 + x_{m+1}^2\right)} = 1$$

from which it follows that x_{m+1} should be:

$$x_{m+1} = \sqrt{1 - \left(x_1^2 + x_2^2 + ... + x_m^2\right)} = \sqrt{\left(1 - \|\boldsymbol{X}\|^2\right)} \qquad (7.1)$$

The normalization procedure, however, is not strictly required, unless the learning strategy selects the neuron with the **largest** output; if it selects the neuron **most similar** to the input vector, the counter-propagation algorithm works without any normalization or renormalization procedures. This is explained below, using an example.

The normalizing variable is in some respects similar to the bias described in Chapter 2. The purpose of both is to change the actual input vector $\boldsymbol{X}$ into an $\boldsymbol{X}'$ so that the calculated net input values (see Section 2.6 for details) lie within the range where the methods work best. Such an adjustment requires the additional weight at each neuron as well as the normalization of weights at the beginning and renormalization at the end of each cycle of training.

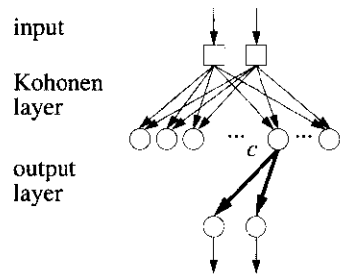

Figure 7-4: The answer (response) to a given input is stored as a weight vector connecting the selected neuron in the Kohonen network and all output neurons (bold). $\boldsymbol{W}_c$ is the vector of weights of the winning neuron c.

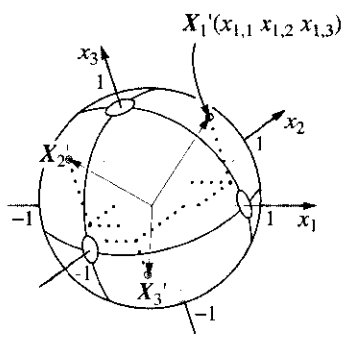

Figure 7-5: If the third variable x_3 is added to the two-dimensional objects $\boldsymbol{X}_s(x_{s1}, x_{s2})$, this third variable is easily determined in such a way that all three-dimensional points $\boldsymbol{X}_s'(x_{s1}, x_{s2}, x_{s3})$ will lie on the surface of a three-dimensional plane.

7.4 Supervised Competitive Learning

Supervised learning requires sets of pairs $(\boldsymbol{X}_s, \boldsymbol{Y}_s)$ for input: the actual input into the network is the vector $\boldsymbol{X}_s$, and the corresponding

target, or prespecified answer, is labeled $\boldsymbol{Y}_s$. The goal of any supervised learning is to form a black box that will give the correct answers $\boldsymbol{Y}_s$ for each vector $\boldsymbol{X}_s$ from the training set (see Figure 5-2). After the training has been completed successfully, it is hoped that the black box will give correct predictions for any new object $\boldsymbol{X}$.

It is hard to formalize the types of predictions which can be accomplished by a counter-propagation network; they can be of very different types. The simplest are those classifying multidimensional objects $\boldsymbol{X}$ into proper categories. More complex predictions involve content-dependent retrievals, where incomplete or fuzzy data are entered and the originals are recovered. For this kind of retrieval, counter-propagation is the optimum method. Still another type is modeling a complex multivariate, nonlinear function yielding a low-dimensional answer (usually 1D or 2D).

The problem with the counter-propagation network is that it needs large quantities of data covering all possible answers. Also the number of **different** answers the counter-propagation method can yield is limited by the size of the network; if there are not too many different answers in the given problem domain, the network may be small, but problems that require large numbers of different solutions cannot be solved satisfactorily.

Recall that the first active layer in the counter-propagation architecture is a Kohonen layer. After each input of an m-variate input $\boldsymbol{X}$, one neuron is selected as the "winner" exactly as shown in the previous chapter, either by choosing the neuron with the largest output signal out_c:

$$out_c = \max(out_j) = \max\left(\sum_{i=1}^{m} w_{ji} x_{si}\right) \qquad (6.1)$$

$$j = 1, 2, \ldots, n$$

or by choosing the neuron j with the corresponding weight vector $\boldsymbol{W}_j$ $(w_{j1}, w_{j2}, \ldots, w_{jm})$ most similar to the input $\boldsymbol{X}$ $(x_1, x_2, \ldots, x_m)$:

$$out_c \leftarrow \min\left[\sum_{i=1}^{m} (x_{si} - w_{ji})^2\right] \qquad (6.2)$$

$$j = 1, 2, \ldots, n$$

The choice between these strategies is more or less a matter of personal preference, because they show almost identical performance; neither has a decisive advantage over the other.

However, remember that when Equation (6.1) is used, the input vector $\boldsymbol{X}$ and the weights must be normalized so that their magnitudes are equal to 1. The normalization has two effects on the economy of the method; first, it requires an additional weight on each Kohonen neuron; and second, it requires an additional loop inside the correction algorithm in order to renormalize the corrected weights.

Since the method requires a large number of neurons in the Kohonen layer, an additional weight at each of them significantly increases the memory requirements. Moreover, large Kohonen networks require large neighborhoods of neurons to be corrected at the beginning of each pass; since this loop occurs in the innermost part of the algorithm, the time requirements are increased as well.

The benefit of normalization is that a numeric overflow of the weights is not possible; there is a significant danger of this occurring when inputs and weights are not normalized.

After the winning neuron has been selected, **two** types of corrections are made:

first, the correction of weights w_{ji} within the neurons of the Kohonen layer

second, the correction of weights in the output layer.

Corrections of the first type (Figure 7-6) are made using Equation (6.6):

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \eta\,(t)\,a\,(d_c - d_j)\left(x_i - w_{ji}^{(old)}\right) \qquad (6.6)$$

The neighborhood-dependent function $a(d_c - d_j)$ and the monotonically decreasing function $\eta(t)$ are discussed in Chapter 6 in connection with competitive learning in the Kohonen network.

After the corrected weight vector $\boldsymbol{W}_j^{(new)}$ $(w_{j1}^{(new)}, w_{j2}^{(new)}, \ldots, w_{jn}^{(new)})$ has been obtained, it has to be renormalized. If the selection of the winning neuron is made according to the largest output criterion (6.1), the renormalization (Equation (7.1)) is mandatory:

$$W_j^{(new)} = \frac{W_j^{(new)}}{\|W_j\|^2} = \frac{W_j^{(new)}}{\sqrt{\sum_{i=1}^{m}\left(w_{ji}^{(new)}\right)^2}} \qquad (7.2)$$

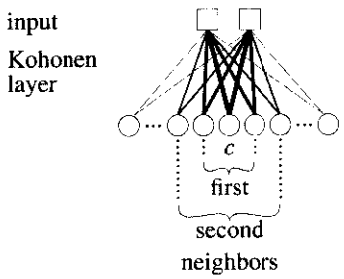

Figure 7-6: The first type of weight corrections affects the weights between the input and the Kohonen layer.

Otherwise (if the most similar neuron criterion, (6.2), is used) it is optional. Note that in Equation (7.1), the dimension m of the input vector $\boldsymbol{X}$ and thus that of the weight vector $\boldsymbol{W}_j$ should be larger by one than if (6.2) is used.

The second type of correction affects the weights between the Kohonen layer and the output layer. (In Section 7.3 we used "r" as the symbol for these weights. In the following discussion, $\boldsymbol{R}_k$ is the weight vector for neuron k; the (row) vectors $\boldsymbol{R}_k$ together make up the output weight matrix $\boldsymbol{R}$, whose columns will be designated $\boldsymbol{C}_j$.)

The answers are not stored in $\boldsymbol{R}_k$. Because only one neuron is activated in the Kohonen layer, per input component, the result vector must consist of **one** weight from each output neuron. This is equivalent to a **column** $\boldsymbol{C}_j$ of the output weight matrix $\boldsymbol{R}$ (the "j" reminds us that the vector $\boldsymbol{C}_j$ is actually associated with the Kohonen neuron j (Figure 7-7)).

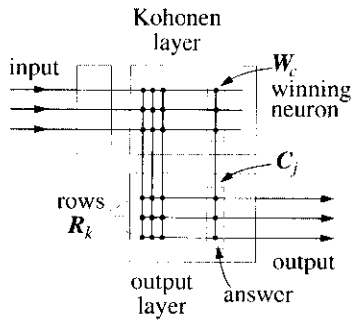

Figure 7-7: The outputs of the counter-propagation network are stored in the columns $\boldsymbol{C}_j$ and not in the rows $\boldsymbol{R}_k$ of the output weight matrix.

> Remember, when correcting output weights, that the weights are corrected within the column vector $\boldsymbol{C}_j$ and **not** within the row vector $\boldsymbol{R}_k$!

Equation (7.3) shows how output weights are to be corrected:

$$c_{ji}^{(new)} = c_{ji}^{(old)} + \eta\,(t)\,a\,(d_c - d_j)\left(y_i - c_{ji}^{(old)}\right) \qquad (7.3)$$

As before, c is the index of the winning Kohonen neuron; j is the index of the neighboring neuron being corrected; i runs over all the weights linking the Kohonen neuron j with the output neurons i. Each of the n output neurons represents one component (variable) of the target vector $\boldsymbol{Y} = (y_1, y_2, \ldots, y_n)$.

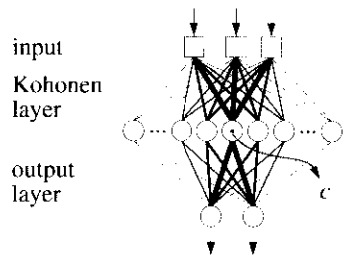

Figure 7-8: The thickness of the lines symbolizes the extent of the correction of the weights.

Figure 7-8 shows the weights and neurons to which the weights are connected in the entire procedure. The extent of the correction depends on the topological distance of the corrected neuron from the winning one, and the thickness of the lines used to draw them is in proportion to the degree of correction.

Therefore, the corrected weights c_{ji} need not be normalized, because they represent the result that should be used later as the output from the counter-propagation network.

Recall that, unlike the other networks we have studied before, the output from a counter-propagation network is obtained from **one** Kohonen network.

A counter-propagation network serving as a lookup table can be visualized as two boxes, one on top of the other (Figure 7-9). The two boxes may be of different height but they have the same length and width. The upper box represents the Kohonen layer and the lower one the output layer. The neurons are packed as columns in the boxes and are accessed by the indices j' and j'' describing the top plane of the box. The position of the Kohonen neuron most excited or most similar to the input object $\boldsymbol{X}$ is referenced by these coordinates j' and j''. The position (j', j'') also determines the position of the vertical column in the output box where the corresponding answers are stored.

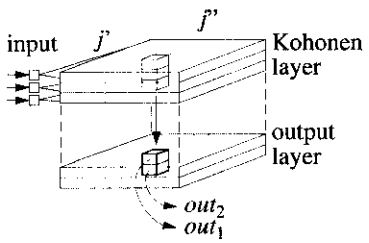

Figure 7-9: Counter-propagation network shown as two boxes. The upper box contains the weights $\boldsymbol{W}$ of the Kohonen network, while the lower one contains the weights $\boldsymbol{C}$ of the output layer.

In Figure 7-9 we have labelled the winning neuron c by the indices j' and j'' to indicate its position in the box and still retain the image of a two-dimensional Kohonen network. (In the equations in this Chapter we have combined the two indices j' and j'' into one index j, as is usually done.)

The Kohonen network is usually implemented as this box implies, as a three-dimensional array, $w_{j'j''i}$, where j' runs from 1 to the length of the map, j'' from 1 to the width of the map, and i indicates the number of input signals.

You may wonder why the results that are stored as weights on the output neurons are not just replaced by the actual targets instead of undergoing the tedious iteration involved in Kohonen mapping. This would be a valid observation if only binary vectors were involved, since in this case there is no need to adapt the output weights according to any scheme; they are either zero or one and can be substituted directly:

$$c_{ci}^{(new)} = y_i \tag{7.4}$$

where y_i is the desired target answer. There is no need to look at the neurons in the neighborhood of the winning one; the substitution is made only for the weights connecting the winning neuron c with all n output neurons (index c instead of j).

When nonbinary values are used as components of the target vector, we have to be aware that different inputs with slightly different targets may excite the same winning neuron and thus yield the same answer. Thus, the similarity of inputs shows in the similarity of outputs. The correlation between the input and the output variables as well as correlations within each group can easily be obtained by inspecting the resulting weights after the lookup table is generated.

These correlation maps are the most outstanding result of the counter-propagation network and will be discussed in more detail in the next chapter.

7.5 Learning to Play Tennis

In this example, we will demonstrate various capabilities of the counter-propagation network.

First, the counter-propagation network is capable of dealing with vectors (objects) representing real values on the input side as well as on the output (target) side.

Second, a lookup table can be formed from a comparably large set of experimental data and can be utilized as a substitute for a mathematical model (that is, an explicit functional relationship).

Third, the correlations among different variables can be obtained from the resulting layer of output weights.

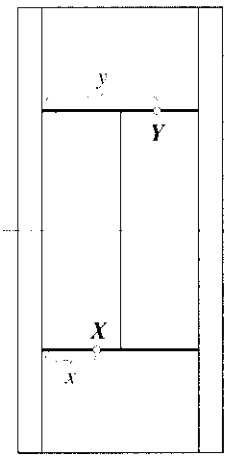

Figure 7-10: Model of a tennis court: x and y are the positions of players $\boldsymbol{X}$ and $\boldsymbol{Y}$, who move only along their service lines.

In real life, learning tennis requires a court and a lot of practice. The more different strokes you learn, the better your play. The two prime factors in tennis are getting to the right part of the court (in time), and returning the ball to an exact spot on your opponent's side.

Obviously, there are a lot of technical details, such as the speed and spin of the ball, or analyzing your opponent's weaknesses; but for now, let's assume that only these two issues (where you are, and how you hit the ball) are involved.

To "compute" your response, you need two data: your opponent's position and how he swings the racket. From these two data to estimate the trajectory of the ball (where it will land) and how to hit it.

For simplicity, we will assume that both players can move only on the service lines of the tennis court (bold lines in Figure 7-10). The players will be labeled as $\boldsymbol{X}$ and $\boldsymbol{Y}$ (trainer and trainee). Their positions on the service line are marked as x and y.

The input data that the trainee $\boldsymbol{Y}$ obtains for learning are the position x of the trainer $\boldsymbol{X}$ and the angle β at which the ball is flying against her. Therefore, the input $\boldsymbol{X}$ can be regarded as a two-dimensional vector:

$$\boldsymbol{X} = (x, \beta)$$

The trainee must position herself at position y (where the ball will cross the service line), holding her racket at the angle γ which will return the ball towards position z. We will simplify further by

assuming that z is fixed at the rightmost point of $\boldsymbol{X}$'s service line (Figure 7-11). The desired learning target is thus a two-dimensional vector $\boldsymbol{Y}$:

$$\boldsymbol{Y} = (y, \gamma)$$

The position of the trainer will be selected randomly between 0 and a (the width of the court). The range of the angle β at which he can send the ball depends on his position x; if he is at the point $x = 0$, then $\tan\beta$ can vary between 0 and a/b (b is the length of the court), while at the position $x = a$, $\tan\beta$ can range between $-a/b$ and 0. For a general position x of the trainer, $\tan\beta$ varies between $-x/b$ and $(a - x)/b$, so

$$\text{for any } x \in [0, a]$$

$$\tan\beta \in [-x/b,\ (a - x)/b]$$

If the trainee is a complete newcomer, her answers will be random; but if her best answer is always corrected towards even better performance, she will tend to select better and better answers. (Unlike a real tennis instructor, we must calculate the "right" answers using appropriate equations and then compare the trainee's answers with them.).

Figure 7-12 illustrates how the answer $\boldsymbol{Y}(y, \gamma)$ is calculated for each $\boldsymbol{X}(x, \beta)$. First we can see that:

$$\tan\beta = (y - x)/b$$

with which the first output y of the target $\boldsymbol{Y}(y, \gamma)$ can easily be calculated:

$$y = x + b\tan\beta$$

Next, the angle γ between the racket plane and the service line should be estimated. From Figure 7-12 we see that:

$$\gamma + \varphi = 0.5\,(\beta + \varphi)$$

This gives us:

$$\gamma = 0.5\,(\beta - \varphi)$$

With:

$$\tan\varphi = (z - y)/b$$

we obtain:

$$\gamma = 0.5\,\{\beta - \arctan[(z - y)/b]\} \qquad (7.5)$$

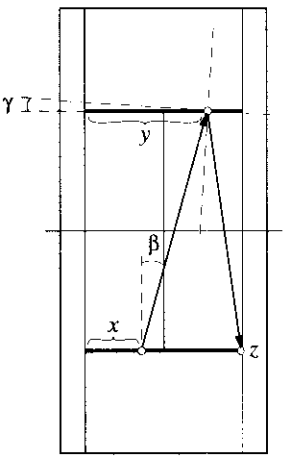

Figure 7-11: The variables (x, β) and (y, γ) representing the input (trainer's stroke) and desired output (trainee's response).

Thus, the output $\boldsymbol{Y}(y, \gamma)$ can be calculated from the input parameters $\boldsymbol{X}(x, \beta)$ and the dimensions of the tennis court a, b, and z using Equation (7.5) (z is the location to which the trainee wants to send the ball). Since in this example z is always equal to a, the goal of the training is, in effect, to teach the trainee how to hit the extreme right corner of the trainer's part of the court, regardless of where she is, or where the ball is coming from.

This can all be accomplished by the counter-propagation network. Initially, we construct a (2 x 625 x 2) counter-propagation network, as shown in Figure 7-13. (The 625 neurons come from a (25 x 25) neuron Kohonen layer used for competitive learning.) Then a number of input vectors $\boldsymbol{X}_s = (x_s, \beta_s)$, let us say 4000, are randomly selected, and to each of them a theoretically correct answer $\boldsymbol{Y}_s(y_s, \gamma_s)$ is assigned. The set of pairs $(\boldsymbol{X}_s, \boldsymbol{Y}_s)$ is then ready to be presented to the counter-propagation network. After having all weights set to small random numbers, the following procedure is launched:

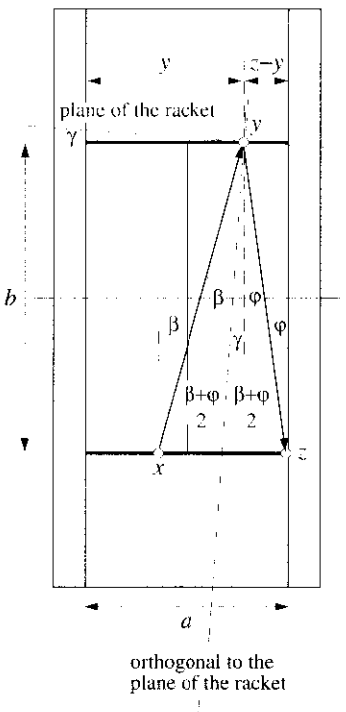

Figure 7-12: The calculation of the response $\boldsymbol{Y}(y, \gamma)$ for each $\boldsymbol{X}(x, \beta)$.

- input a vector $\boldsymbol{X}_s = (x_s, \beta_s)$
- evaluate n sums in all n neurons in the Kohonen layer:

$$out_j = (x_s - w_{j1})^2 + (\beta_s - w_{j2})^2$$
$$j = 1, 2, ..., n$$

- select the winning neuron c as the one having the minimum out_j:

$$out_c = \min \{out_1, out_2, ..., out_n\}$$
$$j = 1, 2, ..., n$$

- correct both weights in each neuron from a given neighborhood around the winning neuron c in the Kohonen layer (see Equation (6.6)):

$$w_{j1}^{(new)} = w_{j1}^{(old)} + \eta(t)\, a\, (d_c - d_j) \left(x_s - w_{j1}^{(old)} \right)$$
$$w_{j2}^{(new)} = w_{j2}^{(old)} + \eta(t)\, a\, (d_c - d_j) \left(\beta_s - w_{j2}^{(old)} \right)$$

- at the beginning of the learning procedure, the product $\eta(t)\, a\, (d_c - d_j)$ is about 0.5 (Equation (6.5) with $a_{max} = 0.5$ and $a_{min} = 0.01$); for each other neuron this product decreases as a function of which

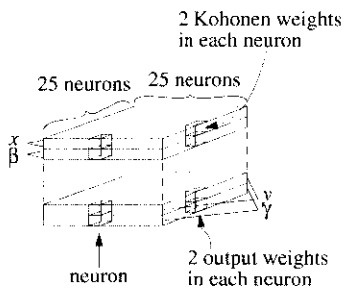

Figure 7-13: 625-neuron network for learning tennis.

neighbor-ring it is in, and how many iteration cycles have already occurred.

- correct the two weights in the output layer leading from all neurons in a given neighborhood of the winning neuron c towards both outputs (see Equation (7.3)):

$$c_{j1}^{(new)} = c_{j1}^{(old)} + \eta\,(t)\,\mathrm{a}\,(d_c - d_j)\left(y_s - c_{j1}^{(old)}\right)$$
$$c_{j2}^{(new)} = c_{j2}^{(old)} + \eta\,(t)\,\mathrm{a}\,(d_c - d_j)\left(\gamma_s - c_{j2}^{(old)}\right)$$

In the above correction, exactly the same value of the product $\eta(t)$ a $(d_c - d_j)$ can be employed as is used during the correction of the Kohonen weights. But if you have some good reason, the correction factor $\eta(t)$ a $(d_c - d_j)$ can be changed later on:

- change the factor $\eta(t)$ and the neighborhood range to which the function a $(d_c - d_j)$ is applied, and go to the first step; repeat until all pairs have been sent through the network.

The product $\eta(t)$ a $(d_c - d_j)$ is a kind of adjusting "dial" in the method; you have to find by trial and error which values work best for a given application. In our example, the entire (25 x 25) Kohonen layer has to be encompassed initially. Of course, the weights of neurons lying in the 25^{th} neighbor's ring away from the winning neuron will be changed very little; the correction can amount to only one twentyfifth of the correction applied to the winning neuron's weights (Figure 7-14).

If there are 4000 input vectors, the neighborhood can be shrunk by one ring after each 160 inputs; hence, in the last 160 inputs only the weights of the winning neurons will be corrected and nothing else. Of course, these corrections will be considerably smaller than at the beginning because the factor $\eta(t)$ decreases as we continue to provide inputs. In the present example, by the 4000^{th} input, η had decreased to 0.01 compared to the initial value of 0.5 (Figure 7-14).

Because the corrections of the output weights are implemented in the same loop as the corrections of the Kohonen weights, the learning procedure is quite efficient.

After 4000 inputs, the answers obtained by the (25 x 25) network (which is able to store 625 answers) were quite encouraging. The root-mean-square (RMS) error between the targets and the results given by the counter-propagation network is 0.33.

The RMS error is calculated from the general equation:

$$\text{RMS} = \sqrt{\frac{\sum_{s=1}^{n_i} \sum_{i=1}^{n} (y_{si} - out_{si})^2}{n_i n}} \tag{7.6}$$

where y_{si} is the i-th component of the desired target $\boldsymbol{Y}_s$, out_{si} is the i-th component of the output produced by the network for the s-th input vector, n_i is the number of inputs, and n is the number of output variables.

In our case, this takes the following form:

$$\text{RMS} = \sqrt{\frac{\sum_{k=1}^{4000} \left[(y_s - c_{c1})^2 + (\gamma_s - c_{c2})^2 \right]}{8000}}$$

c_{c1} and c_{c2} are the weights in the output layer selected by the winning Kohonen neuron (c) for the s-th input vector.

An RMS error of 0.33 corresponds to a tolerance of ±0.025 in the position y and a tolerance of ±1 arc degree in the angle γ. It is interesting to see what the correspondence is between this and the player's success rate. Table 7-1 shows the number of unsuccessful hits for different RMS errors (an unsuccessful hit means that her/his response misses the calculated value by more than the tolerance limits).

RMS error	number of unsuccessful hits per 1000
1.52	110 – 160
1.10	47 – 76
0.77	8 – 18
0.62	1 – 8
0.45	1 – 5
0.35	0 – 5
0.26	0 – 2
0.14	0

Table 7-1: Simulation of playing tennis; translation of RMS error into number of unsuccessful hits.

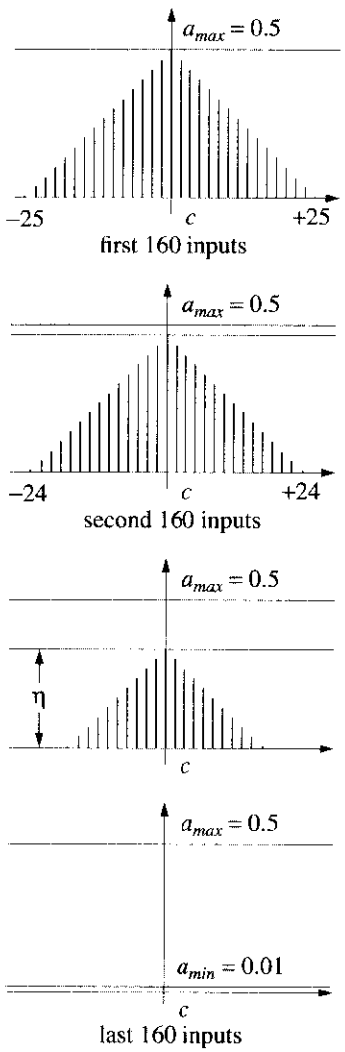

Figure 7-14: How the neighborhood and η change during the training.

This requires us to create many counter-propagation networks yielding different degrees of RMS error; for each one, a test of one thousand randomly selected inputs was repeated five times. Each net gave almost identical RMS errors in the five repeated tests, but different numbers of unsuccessful hits were recorded; hence, the range of unsuccessful hits in Table 7-1.

Additional information can be found in Section 8.6, where the tennis example is worked out more thoroughly.

7.6 Correlations Among the Variables

One of the most valuable properties of the counter-propagation network is that the final values of the output-layer weights contain information about the correlations (lack of functional independence) among the input variables.

Let us consider both "planes" of output weights that were generated during our tennis simulation experiment (Figure 7-15). The architecture of the counter-propagation network for this example can be represented as several square double pyramids. The two upper ones represent the Kohonen weights: one receiving the positions x and the other one the angles β, while the two upside down pyramids represent the output weights: one yielding the positions y and the other one the racket angles γ.

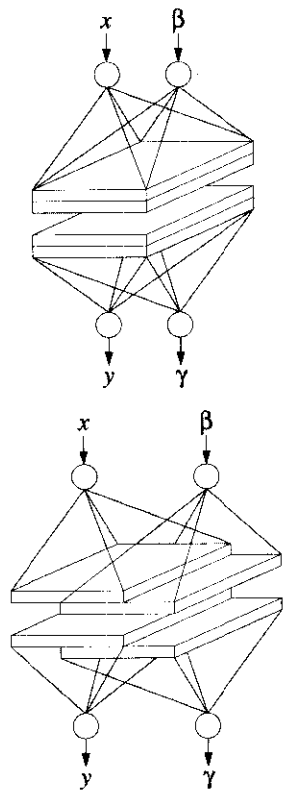

Figure 7-15: The counter-propagation network architecture for the tennis simulation (2 inputs, 2 outputs) visualized as four pyramids.

Computationally, all the pyramids can easily be separated and written in the form of a square matrix. The position of each weight is the position of the Kohonen neuron with which it is involved. In the case of Kohonen weights, these matrices represent the already familiar Kohonen maps (Section 6.4), showing topological relationships among the input variables; there is one for each variable.

Similarly, in the case of the output weights, the upside-down pyramids generate matrices containing the Kohonen maps of the output variables. Figure 7-16 shows all four Kohonen maps obtained in the present example. The upper two maps show the lines that connect points with the same x-position (*iso-x*-position-lines) and the lines with the same β-angle, while the lower two maps show the output's *iso-y*-position and *iso-*γ-angle lines.

These maps tell us that for a given position of x only a certain range of angles β is possible, and vice versa, that a specific angle is allowed only at a certain range of positions. The same is true for the position y and the angle γ for the trainee. That is, these variables are

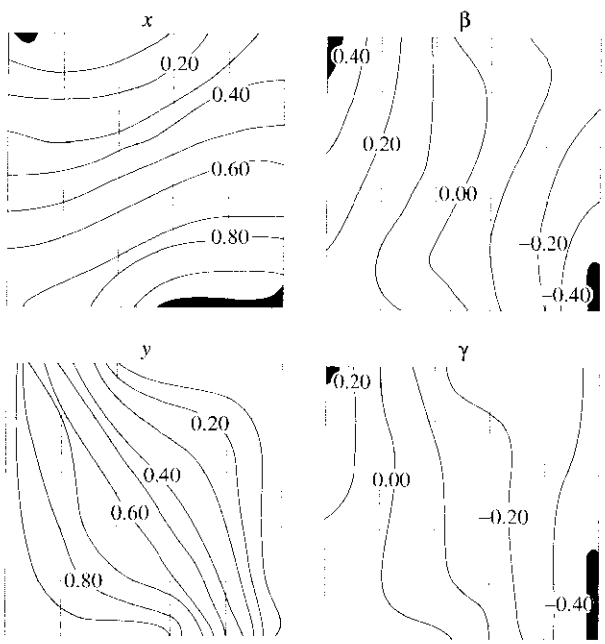

Figure 7-16: The four Kohonen maps for the tennis simulation problem obtained on a (25 x 25) matrix after 5000 random inputs (positions x and y between 0 and 1, angles β and γ in radians). The RMS error of the answers is 0.35.

correlated (not independent): the values of one depend on the value of some other one.

These maps are even more useful because of their "vertical" connectivity; if the maps are drawn on transparencies and overlaid (Figure 7-17), a vertical line going through all maps gives the output variables for a given set of input variables. By recording all values at the intersections of the Kohonen maps with the vertical line **travelling along** a selected *iso*-value line on the map of one variable, all the plots of the remaining variables are obtained at the constant value of the selected one. It is easy to program this *intersection retrieval* once the complete output weights have been obtained.

The procedure of "following the vertical intersections" enables us to obtain the answer to inverse questions, that is, questions relating to an **inverse** model – very hard problems to solve by standard analytical methods.

In the tennis example, such an inverse question would be: does a (trainer's) position x exist which would force the trainee standing at $y = 0.6$ to hold the racket at an angle of 0 degrees in order to hit the corner $x = 1$? To obtain the answer, the vertical line is put on the beginning of the $\gamma = 0$ line in the γ angle map (Figure 7-16d)). Then the vertical line is moved along the $\gamma = 0$ line until it encounters the

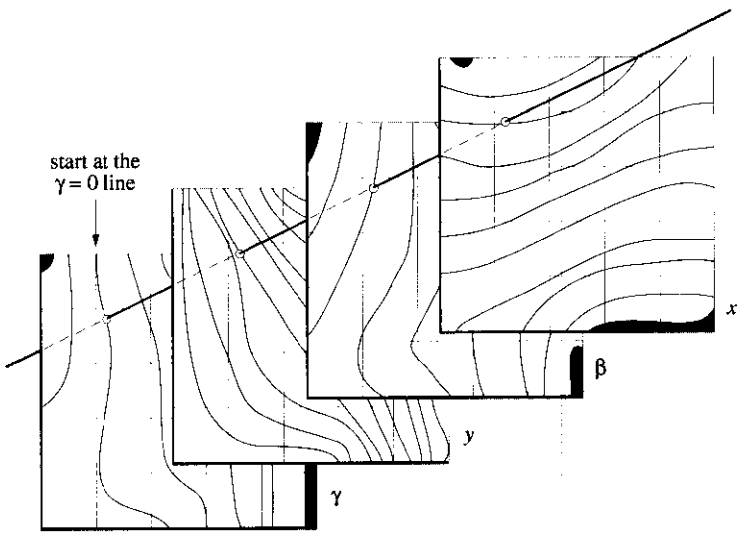

Figure 7-17: With the help of a vertical intersection line, the overlap of all four maps enables the retrieval of answers to *inverse questions*.

0.6-*iso*-y-line in the map above (Figure 7-16c)). Then the intersections of this line with Figures (a) and (b) give the values $x = 0.2$ and $\beta = 0.20$. This means that if the trainer at $x = 0.2$ sends the ball at angle 0.20 radians, the trainee should be at $y = 0.6$ holding the racket with $\gamma = 0$ in order to hit the corner $x = 1$ on the trainer's side of the court.

Additionally, this method allows you to determine whether some conditions can be met at all (and still yield an acceptable solution) – a quite important piece of information for many types of problems.

7.7 Essentials

- the counter-propagation network is basically a two-layer network
- it consists of a Kohonen layer (influenced by the inputs) and an output layer (influenced by the targets)
- it is trained very similarly to the Kohonen-type networks
- the input data are usually normalized
- it employs supervised learning, i.e. you must have a set of "correct" answers
- the answers are stored in the output layer as maps exactly corresponding to the maps generated in the Kohonen layer
- the output is taken from all weights between one Kohonen neuron and all the output neurons
- counter-propagation networks are used as lookup tables
- there is a one-to-one correspondence between the neurons in the Kohonen map and those in the output map

- **normalization**

$$\boldsymbol{X} = (x_1, x_2, ..., x_m) \rightarrow \boldsymbol{X}' (x_1, x_2, ..., x_m, x_{m+1})$$

$$x_{m+1} = \sqrt{\left(1 - x_1^2 + x_2^2 + ... + x_m^2\right)} = \sqrt{\left(1 - \|\boldsymbol{X}\|^2\right)} \tag{7.1}$$

- **finding the best neuron** c

largest output:

$$out_c = \max(out_j) = \max\left(\sum_{i=1}^{m} w_{ji} x_{si}\right) \qquad j = 1, 2, ..., n \tag{6.1}$$

best agreements with the weights:

$$out_c \leftarrow \min\left[\sum_{i=1}^{m} (x_{si} - w_{ji})^2\right] \qquad j = 1, 2, ..., n \tag{6.2}$$

corrections of weights:

- **Kohonen weights**

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \eta(t)\, a\, (d_c - d_j)\left(x_i - w_{ji}^{(old)}\right) \tag{6.6}$$

$$W_j^{(new)} = \frac{W_j^{(new)}}{\|W_j\|^2} = \frac{W_j^{(new)}}{\sqrt{\sum_{i=1}^{m} \left(w_{ji}^{(new)}\right)^2}} \tag{7.2}$$

correction of output weights

$$c_{ji}^{(new)} = c_{ji}^{(old)} + \eta(t)\, a\, (d_c - d_j)\left(y_i - c_{ji}^{(old)}\right) \tag{7.3}$$

RMS error

$$\text{RMS} = \sqrt{\frac{\sum_{s=1}^{n_i} \sum_{i=1}^{n} (y_{si} - out_{si})^2}{n_i n}} \tag{7.6}$$

decreasing learning rate

$$\eta(t) = (a_{max} - a_{min}) \frac{t_{max} - t}{t_{max} - 1} + a_{min} \tag{6.5}$$

7.8 References and Suggested Readings

7-1. R. Hecht-Nielsen, "Counterpropagation Networks", *Proceedings of the IEEE First International Conference on Neural Networks*, 1987 (II) 19 – 32.

7-2. R. Hecht-Nielsen, "Counterpropagation Networks", *Appl. Optics* **26** (1987) 4979 – 4984.

7-3. R. Hecht-Nielsen, "Applications of Counterpropagation Networks", *Neural Networks* **1** (1988) 131 – 140.

7-4. T. Kohonen, *Self-Organization and Associative Memory*, Springer Verlag, Berlin, FRG, 1989.

7-5. G. A. Carpenter, "Neural Network for Pattern Recognition and Associative Memory", *Neural Networks* **2** (1989) 243 – 257.

7-6. H. Ritter, T. Martinetz and K. Schulten, *Neuronale Netze, Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*", Addison-Wesley, Bonn, FRG, 1990.

7-7. J. Dayhoff, *Neural Network Architectures, An Introduction*, Van Nostrand Reinhold, New York, USA, 1990.

7-8. D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley, Reading, USA, 1975, Vol. *3*, p. 506.

7-9. J. Zupan, *Clustering of Large Data Sets*, Research Studies Press, Chichester, UK, 1982.

7-10. J. Gasteiger and J. Zupan, "Neuronale Netze in der Chemie", *Angew. Chem.* **105** (1993) 510 – 558; J. Gasteiger and J. Zupan, "Neural Networks in Chemistry", *Angew. Chem. Int. Ed. Engl.* **32** (1993) 503 – 527.

7-11. J. Zupan, "Introduction to Artificial Neural Network (ANN) Methods: What They Are and How to Use Them", *Acta Chim. Slov.* **41** (1994) 327 – 352.

7-12. J. Zupan, M. Novic and I. Ruisanchez, "Kohonen and Counterpropagation Artificial Neural Networks in Analytical Chemistry", *Chemom. Intell. Lab. Syst.* **38** (1997) 1 – 23.

7-13. I. Ruisanchez, J. Lozano, M. S. Larrechi, F. X. Rius and J. Zupan, "On-line Automated Analytical Signal Diagnosis in Sequential Injection Analysis Systems Using Artificial Neural Networks", *Anal. Chim. Acta* **348** (1997) 113 – 128.

7-14. J. Zupan and M. Novic, "Counter-propagation Learning Strategy in Neural Networks and its Application in Chemistry",

in *Further Advances in Chemical Information*, Ed. H. Collier, Royal Soc. Chem., Cambridge, UK, 1994, pp. 92 – 108.

7-15. J. Zupan, M. Novic and J. Gasteiger, "Neural Networks with Counter-propagation Learning Strategy Used for Modelling", *Chem. Intell. Lab. Syst.* **27** (1995) 175 – 188.

7-16. J. Lozano, M. Novic, F.X. Rius and J. Zupan, "Modelling Metabolic Energy by Neural Networks", *Chemom. Intell. Lab. Syst.* **28** (1995) 61 – 72.

7-17. N. Majcen, K. Rajer-Kanduc, M. Novic and J. Zupan, "Modeling of Property Prediction from Multicomponent Analytical Data Using Different Neural Networks", *Anal. Chem.* **67** (1995) 2154 – 2161.

7-18. F. Ehrentreich, M. Novic, S. Bohanec and J. Zupan, "Bewertung von IR-Spektrum-Struktur-Korrelationen mit Counter-propagation-Netzen", in *Software Development in Chemistry* **10**, Ed.: J. Gasteiger, GDCh, Frankfurt am Main, FRG, 1996, pp. 271 – 292.