Chapter 3
MVN Learning

"Since we cannot know all that there is to be known about anything,
we ought to know a little about everything."

Blaise Pascal

In this Chapter, we consider all aspects of the MVN learning. We start in Section 3.1 from the specific theoretical aspects of MVN learning and from the representation of the MVN learning algorithm. Then we describe the MVN learning rules. In Section 3.2, we consider the first learning rule, which is based on the adjustment of the weights depending on the difference (in terms of the angular distance) between the arguments of the current weighted sum and the desired output. In Section 3.3, we present the error-correction learning rule for MVN. For both learning rules presented in Sections 3.2 and 3.3, we prove theorems about the convergence of the learning algorithm based on these rules. In Section 3.4, we discuss the Hebbian learning rule for MVN. Section 3.5 contains some concluding remarks.

3.1 MVN Learning Algorithm

In Chapter 2, we have introduced discrete and continuous MVN. We also have considered the edged separation of an n-dimensional space, which is implemented by MVN using the k-edge generated by the weights that implement a corresponding input/output mapping.

As any other neuron, MVN creates the weights implementing its input/output mapping during the learning process. The ability to learn from its environment is a fundamental property of a neuron. We have already observed fundamentals of learning in Section 1.2. According to Definition 1.2, the MVN learning as well as any other neuron learning is the iterative process of the adjustments of the weights using a learning rule. In other words, it is reduced to the adaptation of the neuron to its input/output mapping thorough the adjustment of the weights using a learning rule every time, when for some learning sample the neuron's actual output does not coincide with the desired output.

In this Chapter, we present the MVN learning algorithm, which can be based on the two learning rules. One of the rules is based on the estimation of the closeness of the actual output to the desired one in terms of angular distance. Another one is the error-correction learning rule. We will also consider the Hebbian learning for MVN.

I. Aizenberg: Complex-Valued Neural Networks with Multi-Valued Neurons, SCI 353, pp. 95–132.
springerlink.com

3.1.1 Mechanism of MVN Learning

Let us now consider what MVN has to do when it learns, what is behind the learning process. It is also important to mention that those fundamentals of MVN learning, which we consider here are important not only for a single MVN, but for MLMVN (multilayer neural network based on multi-valued neurons), whose learning algorithm with the error backpropagation we will consider in Chapter 4.

Let A be a learning set with the cardinality $|A| = N$, thus the learning set contains N learning samples. They are such samples $\left(x_1^i,...,x_n^i\right) \to d_i, i = 1,...,N$ for which the exact desired output $f\left(x_1^i,...,x_n^i\right) = d_i, i = 1,...,N$ is known.

If we return to the threshold neuron, its learning process can be presented in the following way. Its learning set can always be presented as $A = A_1 \bigcup A_{-1}$, where A_1 is a subset of the learning samples where the neuron's output has to be equal to 1, and A_{-1} is a subset of the learning samples where the neuron's output has to be equal to -1. As we have seen, learning in this case is reduced to the search for a hyperplane, which separates the subsets A_1 and A_{-1} of the learning set in that n-dimensional space where a problem to be learned is defined. The coefficients of a hyperplane equation are the weights implementing a corresponding input/output mapping.

Let us consider now how the discrete MVN learns. Taking into account that the discrete MVN implements a k-valued input/output mapping, it is easy to conclude that a learning set should consist of k classes. Let $k > 2$ be some integer. Let us consider $(n+1)$ - dimensional vectors $X = \left(1, x_1,...,x_n\right)$, $\left(x_1,...,x_n\right) \in T \subseteq O^n$, where O is the set of points located on the unit circle. The 0th coordinate (constant 1) can be considered as a pseudo input corresponding to the weight w_0. We introduce it just to be able to consider a weighted sum $w_0 + w_1x_1 + ... + w_nx_n$ as a dot product of two $(n+1)$ - dimensional vectors $X = \left(1, x_1,...,x_n\right)$ and $W = \left(w_0, w_1,...,w_n\right)$.

Let A_j be a learning subset $\left\{X_1^{(j)},...,X_{N_j}^{(j)}\right\}$ of the input neuron states corresponding to the desired output $\varepsilon^j, j = 0,...,k-1$. In such a case we can present the entire learning set A as a union of the learning subsets $A_j, j = 0,1,...,k-1$ as follows $A = \bigcup_{0 \le j \le k-1} A_j$. In general, some of the sets $A_j, j = 0,1,...,k-1$ may be empty if for some $j = 0,...,k-1$ there is no

learning sample whose desired output is ε^j. It is also clear that $A_i \bigcap A_j = \varnothing$ for any $i \neq j$. A practical content behind this mathematical representation is, for example, a *k*-class classification problem, which is presented by the learning set *A* containing learning subsets $A_j, j = 0,1,...,k-1$ such that each of them contains learning samples belonging only to one of *k* classes labeled by the corresponding class membership label $\varepsilon^j, j = 0,...,k-1$.

Definition 3.16. The sets $A_0, A_1, \ldots, A_{k-1}$ are called *k-separable*, if it is possible to find a permutation $R = (\alpha_0, \alpha_1, ..., \alpha_{k-1})$ of the elements of the set $K = \{0,\ 1,\ ...,\ k-1\}$, and a weighting vector $W = (w_0, w_1, ..., w_n)$ such that

$$P\left(X, \overline{W}\right) = \varepsilon^{\alpha_j} \tag{3.73}$$

for each $A_j, j = 0,1,...,k-1$. Here $\overline{W}$ is a vector with the components complex-conjugated to the ones of the vector *W*, $\left(X, \overline{W}\right)$ is a dot product of the (*n*+1)-dimensional vectors within the (*n*+1)-dimensional unitary space, *P* is the MVN activation function (2.50). Without loss of generality we may always supply (2.50) by $P(0,0) = \varepsilon^0 = 1$. This means that the function *P* is now determined on the entire set $\mathbb{C}$ of complex numbers.

Let *A* be a learning set and $A = \bigcup_{0 \le j \le k-1} A_j$ is a union of the *k*-separable disjoint subsets $A_0, A_1, ..., A_{k-1}$. On the one hand, this means that (3.73) holds for any $X \in A$. On the other hand, the MVN input/output mapping presented by such a learning set is described by the *k*-valued function $f(x_1,\ ...,\ x_n): A \to E_k$. It follows from the fact that the learning subsets $A_0, A_1, ..., A_{k-1}$ are *k*-separable and (3.73) holds for any $X \in A$ that (2.51) also holds for the function $f(x_1,\ ...,\ x_n)$ on the its entire domain *A*, which means that according to Definition 2.7 this function is a *k*-valued threshold function. Hence, we proved the following theorem.

Theorem 15. If the domain of a *k*-valued function $f(x_1,\ ...,\ x_n): A \to E_k$ can be represented as a union of *k*-separable disjoint subsets $A_0, A_1, ..., A_{k-1}$ (some of them can be empty in general), then the function $f(x_1,\ ...,\ x_n)$ is a *k*-valued threshold function.

It directly follows from Definition 3.16 and (3.73) that the problem of MVN learning should be reduced to the problem of the *k*-separation of learning subsets. In other words, the learning problem for the given learning subsets $A_0, A_1, ..., A_{k-1}$ can be formulated as a problem, how to find a permutation $(\alpha_0,\ \alpha_1,\ ...,\ \alpha_k)$ and a weighting vector $W = (w_0, w_1, ..., w_n)$ such that (3.73) holds for the entire learning set *A*.

On the other hand, we see that the notion of *k*-separation of the learning subsets is closely related to the notion of edge-like sequence (see Definition 2.14). Evidently, if the sets $A_0, A_1, ..., A_{k-1}$ are *k*-separable, and the permutation $R = (\alpha_0, \alpha_1, ..., \alpha_{k-1})$ is applied to them, then the edge-like sequence results from such a permutation. From the geometrical point of view, the *k*-separation means that elements from only one learning subset (one class) $A_j = \{X \mid f(X) = \varepsilon^j\}$ belong to each edge of the *k*-edge. Moreover, the elements belonging to the same class cannot belong to the different edges.

We will say that any infinite sequence S_u of objects $u_0, u_1, ..., u_{k-1}$ such that $u_j \in S_u \Rightarrow u_j \in A, \quad u \in A \Rightarrow u = u_j$ for some *j* taking its values from the infinite set, forms a learning sequence from the set *A*.

The MVN learning process should be defined as a process of finding such a permutation $R = (\alpha_0, \alpha_1, ..., \alpha_{k-1})$ of the elements of the set $K = \{0,\ 1,\ ...,\ k-1\}$ and such a weighting vector $W = (w_0, w_1, ..., w_n)$ that (3.73) holds for the entire learning set *A*.

Let us suppose that the permutation $R = (\alpha_0, \alpha_1, ..., \alpha_{k-1})$ is already known. Then the learning process is reduced to obtaining the sequence S_w of the weighting vectors $W_0, W_1, ...$ such that starting from the some m_0 $W_{m_0} = W_{m_0+1} = W_{m_0+2} = ...$ and (3.73) holds. Each weighting vector in the sequence S_w corresponds to the next learning sample. The process of finding the sequence S_w of the weighting vectors is iterative. One iteration of the learning process consists of the consecutive checking for all the learning samples whether (3.73) holds for the current learning sample. If so, the next learning sample should be checked. If not, the weights should be adjusted according to a learning rule (we did not consider learning rules yet). One *learning iteration* (*learning epoch*) is a complete pass over all the learning samples $(x_1^i, ..., x_n^i) \to d_i, i = 1, ..., N$.

Stabilization of the sequence S_w in conjunction with the fact that (3.73) holds means that the learning process converges with the zero error.

As well as for any other neuron, there can be situations when MVN learning with the zero error is not reasonable. It depends on the particular problem, which is necessary to learn. If errors for some learning samples are acceptable (which means that (3.73) may not hold for some learning samples), the mean square error (MSE) (1.20) or the root mean square error (RMSE) (1.21) criteria should be used to stop the learning process. It is important to understand that for the discrete MVN both MSE and RMSE should be applied to the errors in terms of numbers of sectors (see Fig. 2.21), thus not to the elements of the set $E_k = \left\{\varepsilon_k^0, \varepsilon_k, ..., \varepsilon_k^{k-1}\right\}$, but to the elements of the set $K = \{0,1,...,k-1\}$ or to their arguments $\left\{\arg\varepsilon_k^0, \arg\varepsilon_k, ..., \arg\varepsilon_k^{k-1}\right\}$. Hence, in this case the error for the *s*th learning sample $\gamma_s, s = 1,...,N$ is either of

$$\gamma_s = \left(\alpha_{j_s} - \alpha_s\right) \bmod k; s = 1,...,N, \tag{3.74}$$

$$\gamma_s = \left(\arg\varepsilon^{\alpha_{j_s}} - \arg\varepsilon^{\alpha_s}\right) \bmod 2\pi; s = 1,...,N, \tag{3.75}$$

where $\varepsilon^{\alpha_{j_s}}$ is the desired neuron's output for the *s*th learning sample, and ε^{α_s} is the actual MVN output.

The learning process continues until either of MSE or RMSE drops below some pre-determined acceptable minimal value λ. For the reader's convenience, let us adapt here the expressions (1.20) and (1.21) for MSE and RMSE over all *N* learning samples for the local errors (3.74) and (3.75). Equations (1.20) and (1.21) are transformed to, respectively

$$MSE = \frac{1}{N}\sum_{s=1}^{N}\gamma_s^2 < \lambda, \tag{3.76}$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{s=1}^{N}\gamma_s^2} < \lambda. \tag{3.77}$$

If either of MSE (3.76) or RMSE (3.77) criteria is used to stop the learning process, we have to take into account that the sequence of weighting vectors S_w may have left not stabilized when $\lambda > 0$.

The learning process for the continuous MVN does not differ from the one for the discrete MVN. Anyway, the learning set A even for the continuous-valued input/output mapping $f(x_1,...,x_n): T \to O, T \subseteq O^n$ is finite, which means that it can be represented as a union $A = \bigcup_{0 \le j \le k-1} A_j$ of the learning subsets $A_j, j = 0,1,...,k-1$, where k is the number of different values of the function $f(x_1,...,x_n)$ corresponding to the elements of the learning set. In other words, k is the cardinality of the range of the function $f(x_1,...,x_n)$ with respect to the learning set A. If the continuous MVN should learn not with the zero error, then either of MSE (3.76) or RMSE (3.77) criteria with respect to the local errors (3.75) should be applied.

3.1.2 Learning Strategy

Let us have the learning set containing N learning samples $(x_1^i,...,x_n^i) \to d_i, i = 1,...,N$. As we told, one iteration of the learning process consists of the consecutive checking for all the learning samples whether (3.73) holds for the current learning sample. If it does not hold, the weights should be adjusted using a learning rule. This process continues either until the zero error is reached or one of (3.76) or (3.77) holds. It should be mentioned that in the latter case criterion (3.73) can be replaced by either of

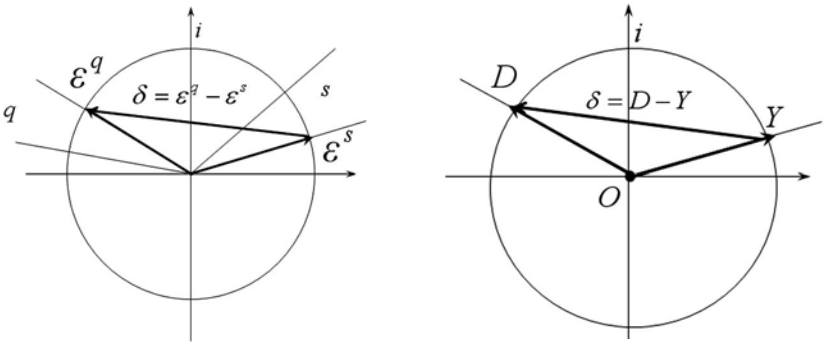
$$\gamma_i = (\alpha_{j_i} - \alpha) \bmod k < \beta; i = 1,...,N, \tag{3.78}$$

$$\gamma_i = (\arg \varepsilon^{\alpha_{ji}} - \arg \varepsilon^{\alpha}) \bmod 2\pi < \beta; i = 1,...,N, \tag{3.79}$$

where β is some acceptable error level for a single learning sample in terms of sectors numbers (3.78) or angular distance (3.79).

If (3.73) does not hold or one of (3.78) or (3.79) does not hold (depending on which error criterion is used), then the MVN weights must be adjusted using a learning rule. Two learning rules will be considered below (Sections 3.2 and 3.3). Geometrically, adjustment of the weights means movement of the weighted sum from the incorrect sector s (discrete MVN, see Fig. 3.33a) to the correct sector q or from the incorrect ray OY (continuous MVN, see Fig. 3.33b) to the correct ray OD.

Thus, the following learning algorithm should be used for MVN learning.



(a) Discrete MVN. ε^q is the desired output, ε^s is the actual output

(b) Continuous MVN. *D* is the desired output, *Y* is the actual output.

Fig. 3.33 Geometrical interpretation of the weights adjustment. The weighted sum has to be moved from the incorrect domain to the correct one.

Let X_j^s be the *s*th element of the learning set *A* belonging to the learning subset A_j. Let *N* be the cardinality of the set *A*, $|A| = N$.

Let *Learning* be a flag, which is 'True' if the weights adjustment is required and "False", if it not required, and *r* be the number of the weighting vector in the sequence S_w.

Step 1. The starting weighting vector W_0 is chosen arbitrarily (e.g., real and imaginary parts of its components can be random numbers); r=0; t=1; *Learning* = 'False';

Step 2. Check (3.73) or one of (3.78) or (3.79) (depending on the error criterion, which is used) for X_j^s:

if (3.73) or one of (3.78) or (3.79) holds
then go to the step 4
else begin Learning = 'True'; *go to* Step 3 *end*;

Step 3. Obtain the vector W_{r+1} from the vector W_r by the learning rule (to be considered);

Step 4. $t = t+1$; *if* $t \le N$
then go to Step 2
else if Learning = 'False'
then the learning process is finished successfully
else begin t=1; *Learning* = 'False'; *go to* Step 2; *end*.

A learning rule, which should be applied on Step 3, is a key point of the learning algorithm. It determines the correction of the weights. It should ensure that after the weights adjustment a weighting vector resulted from this adjustment approaches us closer to the stabilization of the sequence S_w of the weighting vectors. In other words, a learning rule should approach the convergence of the learning algorithm and ensure decreasing of the error after each learning step.

We will consider here two learning rules. Their wonderful property is that they are *derivative-free*. The MVN learning based on this rules should not be considered as the optimization problem. Yes, we have to minimize the neuron error or even to reduce it to zero. But we will see that both learning rules, which we will consider here, generalize the Novikoff's approach to the threshold neuron error-correction learning [12] where the distance from the current weighting vector to the desired weighting vector is decreasing during the learning process without involvement of any optimization technique.

Let us consider the MVN learning rules in detail.

3.2 MVN Learning Rule Based on the Closeness to the Desired Output in Terms of Angular Distance

3.2.1 Basic Fundamentals

Both learning rules (that we consider in this section and in the following section) are based on the compensation of the MVN error by adding the adjusting term to each component of the current weighting vector.

While the second learning rule, which we will consider below in Section 3.3, can be considered as a direct generalization of the Rosenblatt error-correction leaning rule for the threshold neuron, the first learning rule, which we are going to consider now, is based on the compensation of the error between the desired and actual neuron outputs in terms of angular distance between their arguments. This learning rule was initially proposed by N. Aizenberg and his co-authors in [34, 36], then the convergence of the learning algorithm based on this rule was proven in [37], some more adjustments were made in [38] and [60]. Here we will present the most comprehensive description of this algorithm with its deeper analysis (compared to earlier publications).

Let the discrete MVN has to learn some input/output mapping $f(x_1,...,x_n):T \to E_k; T \subseteq O^n$. We have already mentioned that in the case of the continuous MVN and continuous-valued input/output mapping $f(x_1,...,x_n):T \to O; T \subseteq O^n$, all considerations can be reduced to the discrete case because since a learning set with the cardinality N contains exactly N learning samples and the neuron may have at most $k \le N$ different outputs. So we may consider the learning algorithm just for the discrete MVN.

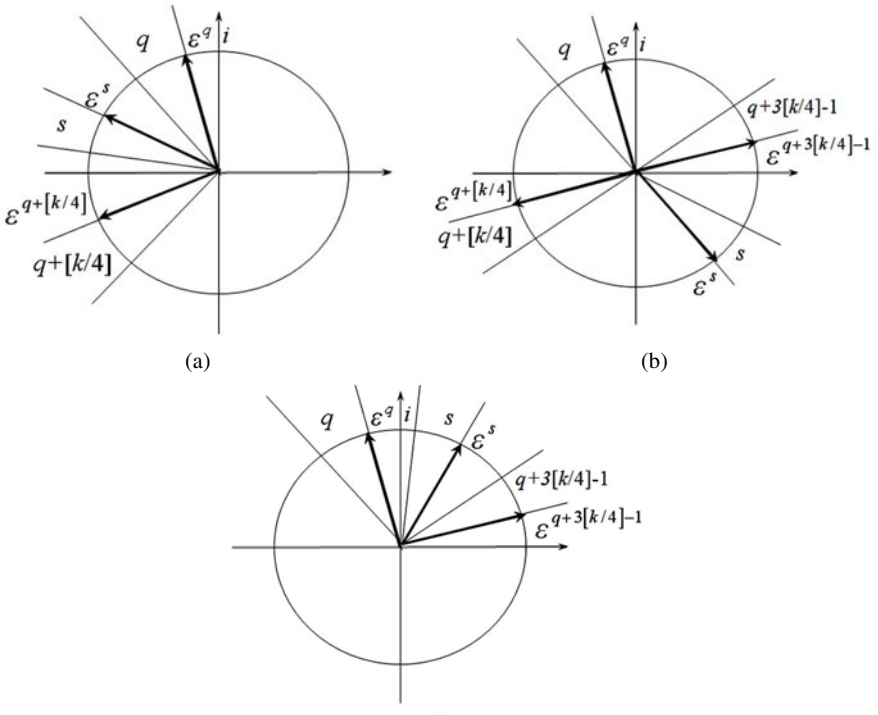


Fig. 3.34 Thee cases of mutual location of the desired (ε^q) and actual (ε^s) MVN outputs

Let us define the following partial order relation on the set $E_k = \left\{\varepsilon_k^0, \varepsilon_k, ..., \varepsilon_k^{k-1}\right\}$. Let us for simplicity use the following notation $\varepsilon = \varepsilon_k$. We will say that ε^α *precedes* to ε^β ($\varepsilon^\alpha \prec \varepsilon^\beta$) if and only if the following condition holds

$$\left(\alpha \bmod k \le \beta \bmod k\right) \wedge \left(0 \le \arg \varepsilon^\beta - \arg \varepsilon^\alpha < \pi\right) \text{ or}$$

$$\left(\alpha \bmod k \ge \beta \bmod k\right) \wedge \left(-\pi \le \arg \varepsilon^\beta - \arg \varepsilon^\alpha < 0\right),$$

where $\alpha, \beta \in K = \left\{0, 1, ..., k-1\right\}$. In other words $\varepsilon^\alpha \prec \varepsilon^\beta$ if and only if ε^α is located "lower" than ε^β in the clockwise direction from ε^β in the "right" half-plane from the line crossing the origin and the point corresponding to ε^α on the unit circle.

Let ε^q be the desired MVN output and ε^s be the actual MVN output (see Fig. 3.33a). Let us discover how far they can be located from each other in terms of

the angular distance accurate within the angle $\pi/2$. Let us consider the case $k \geq 4$ (see Fig. 3.34). There are three possible situations.

1) ε^s is located to the "left" (in the counterclockwise direction) from ε^q such that $\left(\arg\varepsilon^s - \arg\varepsilon^q\right) \bmod 2\pi \leq \pi/2$ and $\varepsilon^{(q+1) \bmod k} \prec \varepsilon^s \prec \varepsilon^{(q+[k/4]) \bmod k}$ (see Fig. 3.34a, [*k*/4] is an integer part of *k*/4).

2) ε^s is located approximately across the unit circle with respect to ε^q, which means that $\pi/2 < \left(\arg\varepsilon^s - \arg\varepsilon^q\right) \bmod 2\pi < 3\pi/2$ and $\varepsilon^{(q+[k/4]+1) \bmod k} \prec \varepsilon^s \prec \varepsilon^{(q+3[k/4]-1) \bmod k}$ (see Fig. 3.34b).

3) ε^s is located to the "right" (in the clockwise direction) from ε^q such that $\left(\arg\varepsilon^s - \arg\varepsilon^q\right) \bmod 2\pi \leq \pi/2$ and $\varepsilon^{(q+3[k/4]) \bmod k} \prec \varepsilon^s \prec \varepsilon^{(q+k-1) \bmod k}$ (see Fig. 3.34c).

The goal of a learning rule is to correct the error, which is approximately equal in terms of angular distance for the three just considered cases $-\pi/2$, π, and $\pi/2$, respectively. Thus, to correct the error, we need to "rotate" the weighted sum, compensating this error. This "rotation" must be done by the adjustment of the weights in such a way, that the adjusted weighting vector moves the weighted sum either exactly where we need or at least closer to the desired sector.

This means that our learning rule has to contain some "rotating" term, which should vary depending on which of the considered above error cases takes place.

The following learning rule was proposed in [34, 36, 37] to correct the weighting vector $W = \left(w_0, w_1, ..., w_n\right)$

$$\tilde{w}_i = w_i + \frac{1}{n+1} C_r \omega_r \varepsilon^q \overline{x}_i; i = 0,1,...,n, \tag{3.80}$$

where n is the number of the neuron inputs, w_i is the ith component of the weighting vector before correction, $\tilde{w}_i$ is the same component after correction, $\overline{x}_i$ is the ith neuron input complex-conjugated ($x_0 \equiv 1$ as we agreed above is a pseudo-input corresponding to the bias w_0), C_r is the learning rate, ε^q is the desired output, and ω_r is a rotating coefficient, which has to compensate the angular error, which we have just considered. To choose ω_r, we have to consider the following cases corresponding to the error cases considered above. Let i be an imaginary unity, and $\varepsilon = \varepsilon_k = e^{i2\pi/k}$.

Case 1. $\omega_m = -i\varepsilon$, if $\varepsilon^s = \varepsilon^{q+1}$ for k=2 and k=3, or $\varepsilon^{(q+1) \bmod k} \prec \varepsilon^s \prec \varepsilon^{(q+[k/4]) \bmod k}$ for k≥4.

Case 2. $\omega_m = 1$, if $\varepsilon^{(q+[k/4]+1) \bmod k} \prec \varepsilon^s \prec \varepsilon^{(q+3[k/4]-1) \bmod k}$ for k≥4 (for k<4 such a case is impossible).

Case 3. $\omega_m = i$, if $\varepsilon^s \prec \varepsilon^{q+2}$ for k=3, or $\varepsilon^{(q+3[k/4]) \bmod k} \prec \varepsilon^s \prec \varepsilon^{(q+k-1) \bmod k}$ for k≥4 (for k=2 such a case is impossible).

Thus, adjusting the weights according to (3.80), we obtain the following equation for the r+1st weighting vector belonging to the sequence S_w from the rth weighting vector belonging to the same sequence

$$W_{r+1} = W_r + \frac{1}{n+1} C_r \omega_r \varepsilon^q \overline{X}, \tag{3.81}$$

where r is the number of the current weighting vector in the sequence S_w, addition is component-wise, and $\overline{X} = \left(1, \overline{x}_1, ..., \overline{x}_n\right)$ is the vector of neuron inputs with the complex-conjugated components.

Before justification of the choice of ω_r let us first clarify a very important role of the multiplier $\frac{1}{n+1}$ in the learning rule (3.80) and let us see how the weighted sum changes after the weights are corrected. Let us find the updated weighted sum after the weights are corrected according to (3.80). The current weighted sum is $z = w_0 + w_1 x_1 + ... + w_n x_n$. Suppose for simplicity, but without loss of generality that $C_m = 1$. For the updated weighted sum, taking into account that $x_i \overline{x}_i = 1; i = 0, 1, ..., n$ (since all x_i are located on the unit circle) we obtain

$$\tilde{z} = \tilde{w}_0 + \tilde{w}_1 x_1 + ... + \tilde{w}_n x_n = \left(w_0 + \frac{1}{n+1} \omega_r \varepsilon^q \right) + \left(w_1 + \frac{1}{n+1} \omega_r \varepsilon^q \overline{x}_1 \right) x_1 +$$

$$+ ... + \left(w_n + \frac{1}{n+1} \omega_r \varepsilon^q \overline{x}_n \right) x_n =$$

$$= \underbrace{w_0 + w_1 x_1 + ... + w_n x_n}_{z} + \underbrace{\frac{1}{n+1} \omega_r \varepsilon^q + ... + \frac{1}{n+1} \omega_r \varepsilon^q}_{n+1 \text{ times}} = z + \omega_r \varepsilon^q.$$

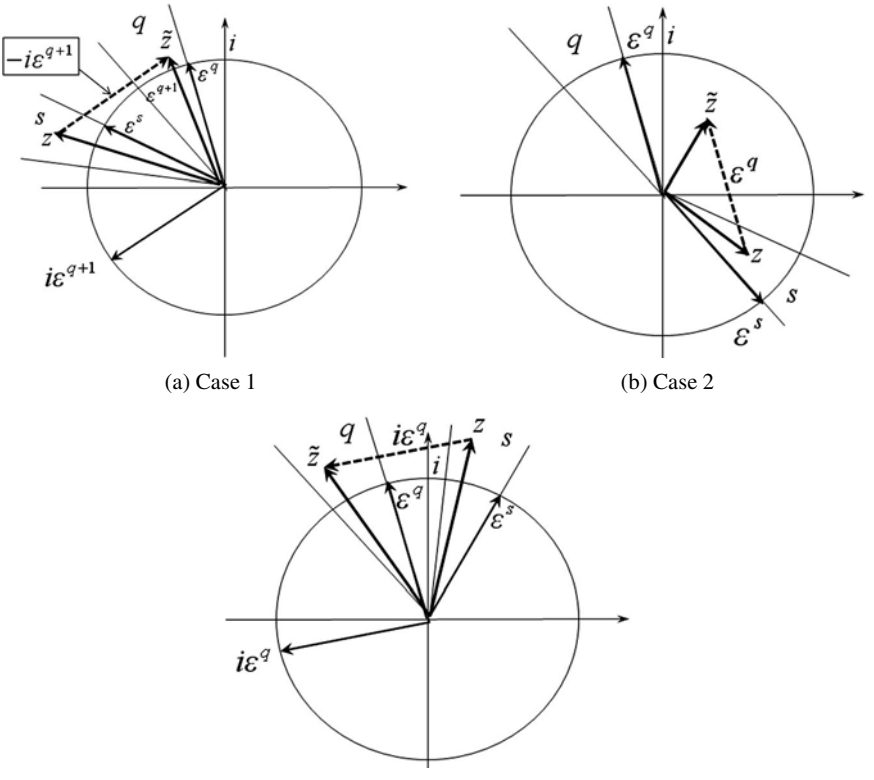


Fig. 3.35 Movement of the weighted sum z after the correction of the weights according to (3.80)

This means that after the weights are corrected, the weighted sum is changed by $\omega_r \varepsilon^q$. The multiplier $\dfrac{1}{n+1}$, which can be considered as a constant learning rate is important to avoid change of the weighted sum by $(n+1)\omega_r \varepsilon^q$, which may lead to the jump over a desired output. Thus, using the multiplier $\dfrac{1}{n+1}$, we share the adjusting term among all the weights. In fact, we do not know which of the weights contributes more to the error. Hence, if we assume that each of them contributes uniformly, this assumption is natural.

We can clarify now the choice of ω_r in (3.80) (see Fig. 3.35).

In the Case 1 (see Fig. 3.35a), the current weighted sum should be rotated clockwise, that is to the right side from its current location, because the actual output ε^s is located to the left from the desired output ε^q and the difference between

the arguments of the actual and desired outputs does not exceed $\pi / 2$. Choosing $\omega_r = -i\varepsilon$, we ensure that after the correction of the weights the updated weighted sum $z + \omega_r \varepsilon^q = z - i\varepsilon\varepsilon^q = z - i\varepsilon^{q+1}$ moves closer or exactly to the desired sector q and the MVN output moves closer or exactly becomes equal to ε^q, respectively.

In the Case 3 (see Fig. 3.35c), situation is similar, the difference between the arguments of the actual and desired outputs does not exceed $\pi / 2$, but we need to rotate the current weighted sum counterclockwise, that is to the left side from its current location, because the actual output ε^s is located to the right from the desired output ε^q. Choosing $\omega_r = i$, we ensure that after the correction of the weights the updated weighted sum $z + \omega_r \varepsilon^q = z + i\varepsilon^q$ moves closer or exactly to the desired sector q and the MVN output moves closer or exactly becomes equal to ε^q, respectively.

In the Case 2 (see Fig. 3.35b), the current weighted sum should be flipped because the actual output ε^s is about the opposite to the desired output ε^q and the difference between the arguments of the actual and desired outputs exceeds $\pi / 2$. Since in this case we cannot have any preference where to rotate the weighted sum (clockwise or counterclockwise), we should simply rotate it such that it will be moved to the desired output. Choosing $\omega_r = 1$, we ensure that after the correction of the weights the updated weighted sum $z + \omega_r \varepsilon^q = z + \varepsilon^q$ moves closer or even exactly to the desired sector q and the MVN output moves closer or may exactly becomes equal to ε^q, respectively.

3.2.2 Convergence Theorem

Now we are ready to formulate and prove the theorem about the convergence of the MVN learning algorithm with the learning rule (3.80). We will provide the reader with a new proof of this theorem compared to [37] and [60]. This new proof is shorter and more elegant.

Suppose that the permutation $R = (\alpha_0, \alpha_1, ..., \alpha_{k-1})$ such that (3.73) holds for the entire learning set A is known.

Theorem 16 (About the convergence of the learning algorithm with the learning rule (3.80))**.** If the learning subsets $A_0, A_1, ..., A_{k-1}$ of the learning set A ($A = \bigcup_{0 \le j \le k-1} A_j$) are k-separable for the given value of k according to Definition 3.16 (which means that the corresponding MVN input/output mapping is a k-valued

threshold function), then the MVN learning algorithm with the rule (3.80) converges after a finite number of steps.

Proof. Suppose that the conclusion of the theorem is false. This means that the sequence S_w of the weighting vectors is infinite. Therefore the weights correction using the rule (3.80) gives the infinite amount of the new weighting vectors, which do not satisfy condition (3.73) at least for one element from some learning subset. According to our assumption the learning subsets $A_0, A_1, ..., A_{k-1}$ are *k*-separable. Therefore the weighting vector *W* exist such that (3.73) holds for the entire learning set *A*.

For simplicity and without loss of generality, let us start the learning process from the zero vector $W_1 = \big((0,0),(0,0),...,(0,0)\big)$, where (a,b) is a complex number $a+bi$, (i is an imaginary unity). Let $S_X = \big(X_1, X_2, ..., X_N\big)$ be a learning sequence of input vectors $X_j = \big(1, x_1^j, ..., x_n^j\big)$, $j = 1, ..., N$, and $S_W = \big(W_1, W_2, ..., W_r, ...\big)$ be a sequence of weighting vectors, which appear during the learning process. We have to prove that this sequence cannot be infinite. Let us remove from the learning sequence those vectors for which $W_{r+1} = W_r$, in other words, those input vectors, for which (3.73) hold without any learning. Let $S_{\tilde{W}}$ be the reduced sequence of the weighting vectors. The Theorem will be proven if we will show that the sequence $S_{\tilde{W}}$ is finite. Let us suppose that the opposite is true: the sequence $S_{\tilde{W}}$ is infinite. So from the assumption that $S_{\tilde{W}}$ is infinite, we have to get the contradiction with the conditions of the theorem.

Without loss of generality we can take $C_r = 1$ in (3.80). This leads to the following transformation of (3.81):

$$\tilde{W}_{r+1} = \tilde{W}_r + \frac{1}{n+1}\omega_r \varepsilon^q \bar{X}_r, \tag{3.82}$$

Thus, our sequence of the weighting vectors $S_{\tilde{W}}$ is obtained according to (3.82). The theorem will be proven, if we can prove that the sequence $S_{\tilde{W}}$ is finite.

Suppose the desired MVN output for the first learning sample does not coincide with the actual MVN output. Thus, we have to adjust the weights according to (3.82):

$\tilde{W}_2 = \frac{1}{n+1}\omega_{m_1}\bar{\bar{X}}_1$, and then for the next correction we obtain

$$\tilde{W}_3 = \tilde{W}_2 + \frac{1}{n+1}\omega_{m_2}\bar{\bar{X}}_2 = \frac{1}{n+1}\left[\omega_{m_1}\bar{\bar{X}}_1 + \omega_{m_2}\bar{\bar{X}}_2\right], ...$$

Applying (3.82) to obtain the r+1[st] vector from the learning sequence, we have the following

$$\tilde{W}_{r+1} = \frac{1}{n+1}\left[\omega_{m_1}\bar{\bar{X}}_1 + ... + \omega_{m_r}\bar{\bar{X}}_r\right]. \tag{3.83}$$

where $m_j \in \{1,2,3\}; j = 1,...,r$, and every time m_j is chosen depending on which of three cases (see above) for the angular error takes place.

Let us find a dot product of both parts of (3.83) with the weighting vector W, which exists according to the condition of the theorem (subsets $A_0, A_1, ..., A_{k-1}$ are k-separable):

$$\left(\tilde{W}_{r+1}, \bar{W}\right) = \frac{1}{n+1}\left[\left(\omega_{m_1}\bar{\bar{X}}_1, \bar{W}\right) + ... + \left(\omega_{m_r}\bar{\bar{X}}_r, \bar{W}\right)\right]. \tag{3.84}$$

Let us now estimate the absolute value $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$ of the dot product $\left(\tilde{W}_{r+1}, \bar{W}\right)$:

$$\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| = \frac{1}{n+1}\left|\left[\left(\omega_{m_1}\bar{\bar{X}}_1, \bar{W}\right) + ... + \left(\omega_{m_r}\bar{\bar{X}}_r, \bar{W}\right)\right]\right|. \tag{3.85}$$

Since for any complex number β $|\beta| \geq |\mathrm{Re}\,\beta|$ and $|\beta| \geq |\mathrm{Im}\,\beta|$, then the absolute value of the sum in the right-hand side of (3.85) is always greater than or equal to the absolute values of the real and imaginary parts of this sum. Let $a = \min_{j=1,...,r}\left|\mathrm{Re}\left(\omega_{m_j}\tilde{X}_j, \bar{W}\right)\right|$. Then it follows from (3.85) that

$$\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| \geq \frac{ra}{n+1}. \tag{3.86}$$

According to the fundamental Schwarz inequality [74] the squared dot product of the two vectors does not exceed the product of the squared norms of these vectors or in other words, the norm of the dot product of the two vectors does not exceed the product of the norms of these vectors $\left\|\left(V_1, V_2\right)\right\| \leq \left\|V_1\right\| \cdot \left\|V_2\right\|$. Thus, according to the Schwartz inequality

$$\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| \leq \left\|\tilde{W}_{r+1}\right\| \cdot \left\|W\right\|. \tag{3.87}$$

Taking into account (3.86), we obtain from (3.87) the following

$$\frac{ra}{n+1} \le \left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| \le \left\|\tilde{W}_{r+1}\right\| \cdot \left\|W\right\|.$$

Then it follows from the last inequality that

$$\left\|\tilde{W}_{r+1}\right\| \ge \frac{ra}{\left\|W\right\|(n+1)}. \tag{3.88}$$

Let for simplicity $\frac{a}{n+1} = \tilde{a}$. Then (3.88) is transformed as follows:

$$\left\|\tilde{W}_{r+1}\right\| \ge \frac{r\tilde{a}}{\left\|W\right\|}. \tag{3.89}$$

As we told, W is a weighting vector, which exist according to the condition of the Theorem. According to our assumption, the sequence $S_{\tilde{W}}$ of the weighting vectors is infinite. Since r is the number of the weighting vector in the sequence $S_{\tilde{W}}$, let us consider (3.89) when $r \to \infty$. $\left\|\tilde{W}_{r+1}\right\|$ is a norm of the vector and therefore it is a non-negative finite real number, $\left\|W\right\|$ is a norm of the vector and it is a finite positive real number ($\left\|W\right\| \ne 0$ because vector W is a weighting vector satisfying (3.73) and therefore at least one of its components is not equal to 0), and $\tilde{a}$ is a finite positive real number. It follows from this analysis that $\frac{r\tilde{a}}{\left\|W\right\|} \underset{r\to\infty}{\to} \infty$.

However, this means that from (3.89) we obtain

$$\left\|\tilde{W}_{r+1}\right\| \ge \frac{r\tilde{a}}{\left\|W\right\|} \to \infty. \tag{3.90}$$

Inequality (3.90) is contradictory. Indeed, the norm of a vector, which is in the left-hand side, is a finite non-negative real number. However, it has to be greater than or equal to the infinity in the right-hand side of (3.90), which is impossible. This means that (3.90) is contradictory. This means in turn that either it is impossible that $r \to \infty$ or the vector W does not exist. The latter contradicts to the condition of the Theorem. Hence, $r \not\to \infty$ and it is always a finite integer number. Thus, our assumption that the sequence $S_{\tilde{W}}$ of the weighting vectors is infinite, is false, which means that it is always finite. Theorem is proven.

So the MVN learning algorithm with the learning rule (3.80) converges after a finite number of learning iterations. As we see, this learning algorithm is derivative-free. It is not considered as the optimization problem of the minimization of the error functional. It is important that a famous local minima problem, which is typical for those learning rules that are based on the optimization technique and which we have considered in Section 1.3 (see Fig. 11), does not exist for the MVN learning algorithm based on the learning rule (3.80). The error in this MVN learning algorithm decreases because each following weighting vector in the sequence $S_{\tilde{W}}$ should be closer to the "ideal" weighting vector W, which exists if the MVN input/output mapping is described by some k-valued threshold function. According to (3.86) the absolute value of the dot product of the vector W and the weighting vector W_{r+1} in the sequence $S_{\tilde{W}}$ must be greater than or equal to the finite number proportional to r, which is the number of the correction. On the one hand, since r increases, this means that $\left|\left(\tilde{W}_{r+1}, \overline{W}\right)\right|$ at least does not decrease. On the other hand, as we have proven, $\left|\left(\tilde{W}_{r+1}, \overline{W}\right)\right|$ cannot increase to infinity. This means that the learning algorithm converges when $\left|\left(\tilde{W}_{r+1}, \overline{W}\right)\right|$ reaches its maximum. This means that vectors W_{r+1} and W are as close to each other as it is possible. Ideally, they are collinear or close to collinearity. It follows from (3.85) and (3.86) that $\left|\left(\tilde{W}_{r+1}, \overline{W}\right)\right|$ cannot decrease during the learning process. It may only increase or remain the same. If it increases, this means that the error decreases. This means that geometrically, the MVN learning algorithm based on the learning rule (3.80) "rotates" the initial weighting vector such that $\left|\left(\tilde{W}_{r+1}, \overline{W}\right)\right|$ should be maximized. It follows from this that the worst starting condition for the learning process is when the vectors W_1 (the starting weighting vector) and W are orthogonal to each other and $\left|\left(\tilde{W}_{r+1}, \overline{W}\right)\right|$=0, while the best starting condition is when the same vectors are about collinear. The closer they are to the collinearity, the smaller is the error and the shorter way is required for the convergence of the learning process. This kind of "non-optimization" learning is based on the same idea, which was developed in [12] by A. Novikoff for the threshold neuron and its error-correction learning.

Let us make one more remark. If the permutation $R = (\alpha_0, \alpha_1, ..., \alpha_{k-1})$ such that (3.73) holds for the entire learning set A is not known, it is possible to find such a permutation by $k!$ means. This follows from the fact that there are exactly $k!$ different permutations from the elements of the set A.

3.3 MVN Error-Correction Learning Rule

3.3.1 Basic Fundamentals

We have considered in Section 1.2 the error-correction learning rule (1.17) for the threshold neuron. For the reader's convenience we repeat it here

$$\tilde{w}_0 = w_0 + \alpha\delta;$$
$$\tilde{w}_i = w_i + \alpha\delta x_i, i = 1,...,n,$$

where $\delta = d - y$ is the error, which is according to (1.5) the difference between the desired neuron output and its actual output, and α is the learning rate. Is it possible to apply the same idea for MVN? Yes, the MVN error-correction learning rule was justified in [60] where the convergence of the MVN learning algorithm with the error-correction learning rule was also proven.

The MVN error-correction learning rule was proposed in 1995 by the author of this book, Naum Aizenberg, and Georgy Krivosheev in [110] (then the convergence of the learning algorithm based on it was proven by the author of this book, N. Aizenberg, and J. Vandewalle in [60]), as follows

$$\tilde{w}_i = w_i + \frac{C_r}{(n+1)}\left(\varepsilon^q - \varepsilon^s\right)\overline{x}_i; i = 0,1,...,n, \tag{3.91}$$

where n is the number of the neuron inputs, w_i is the ith component of the weighting vector before correction, $\tilde{w}_i$ is the same component after correction, $\overline{x}_i$ is the ith neuron input complex-conjugated ($x_0 \equiv 1$ as we agreed above is a pseudo-input corresponding to the bias w_0), C_r is the learning rate, ε^q is the desired output, ε^s is the actual output. $\delta = \varepsilon^q - \varepsilon^s$ is the error. It should be mentioned that throughout this Section we still use the notation $\varepsilon = \varepsilon_k$ for simplicity. As well, as the MVN learning algorithm based on the angular error compensation learning rule, the MVN learning algorithm based on the error-correction learning rule is also reduced to the straightening of the sequence S_w of the weighting vectors. Thus, adjusting the weights according to (3.91), we obtain the following equation for the r+1st weighting vector belonging to the sequence S_w from the rth weighting vector belonging to the same sequence

$$W_{r+1} = W_r + \frac{C_r}{(n+1)}\left(\varepsilon^q - \varepsilon^s\right)\overline{X}, \tag{3.92}$$

where r is the number of the current weighting vector in the sequence S_w, addition is component-wise, and $\overline{X} = (1, \overline{x}_1, ..., \overline{x}_n)$ is the vector of inputs with the complex-conjugated components. As we have done this earlier, we introduce the pseudo-input $x_0 \equiv 1$ corresponding to the weight w_0.

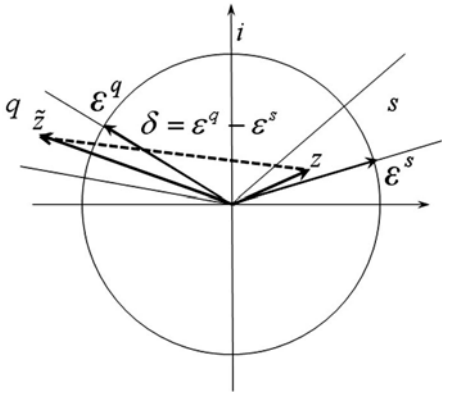


Fig. 3.36 Movement of the weighted sum z after the correction of the weights according to (3.91)

Let us find the updated weighted sum after the weights are corrected according to (3.91). The current weighted sum is $z = w_0 + w_1 x_1 + ... + w_n x_n$. Suppose for simplicity, but without loss of generality that $C_r = 1$.

If $\delta = \varepsilon^q - \varepsilon^s$ is the error, then for the updated weighted sum, taking into account that $x_i \overline{x}_i = 1; i = 0, 1, ..., n$ (since all x_i are located on the unit circle) we obtain

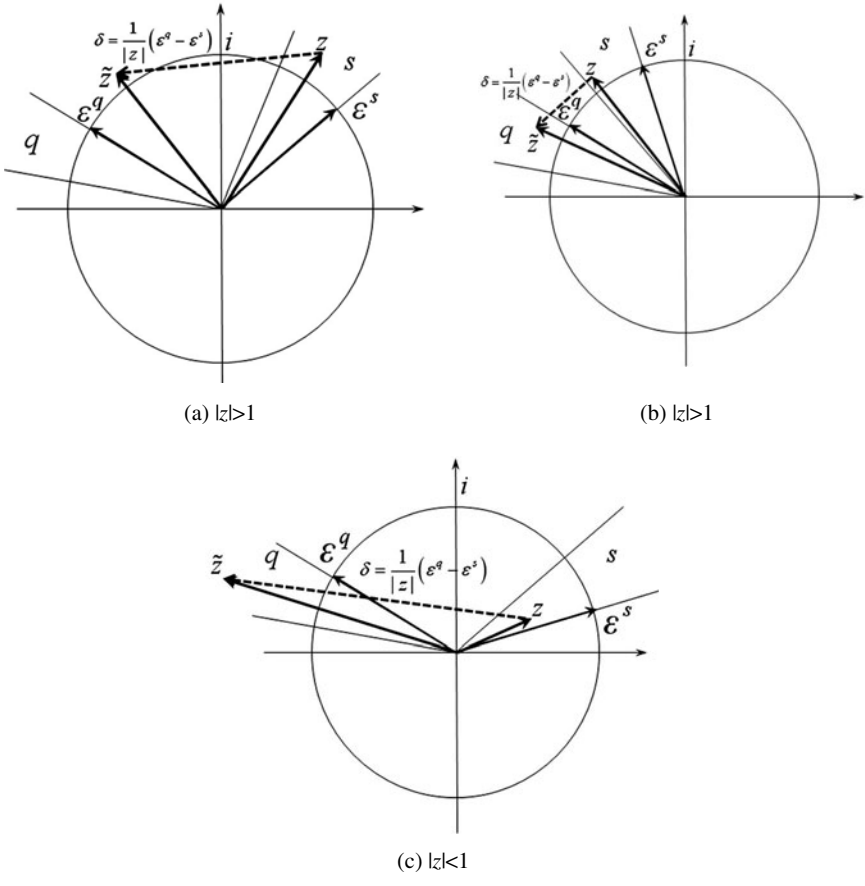
$$\begin{aligned}
&\tilde{z} = \tilde{w}_0 + \tilde{w}_1 x_1 + ... + \tilde{w}_n x_n = \\
&\left(w_0 + \frac{1}{n+1}\delta\right) + \left(w_1 + \frac{1}{n+1}\delta\overline{x}_1\right)x_1 + ... + \left(w_n + \frac{1}{n+1}\delta\overline{x}_n\right)x_n = \\
&= \underbrace{w_0 + w_1 x_1 + ... + w_n x_n}_{z} + \underbrace{\frac{1}{n+1}\delta + ... + \frac{1}{n+1}\delta}_{n+1 \text{ times}} = z + \frac{n+1}{n+1}\delta = \\
&= z + \delta.
\end{aligned} \tag{3.93}$$

This means that after the weights are corrected, the weighted sum is changed exactly by δ that is by the error. This is illustrated in Fig. 3.36. The current weighted sum located in the sector s has to be moved to the sector q. The direction of this movement is determined by the error $\delta = \varepsilon^q - \varepsilon^s$, which is equal to the difference between the desired output ε^q and actual output ε^s. Correcting the weights according to (3.91) (or (3.92), which is the same), we ensure that after the correction of the weights the updated weighted sum $z + \delta$ moves closer or exactly to the desired sector q and the MVN output moves closer or exactly becomes equal to ε^q, respectively.

In [62], it was suggested to modify the learning rule (3.92). A variable learning rate, which is equal to $1/|z_r|$, the inverse absolute value of the current weighted sum, was introduce there. This modification should be reasonable for those input/output mappings that are described by highly nonlinear functions with many irregular jumps. With this modification, the learning rule (3.92) becomes

$$W_{r+1} = W_r + \frac{C_r}{(n+1)|z_r|}\left(\varepsilon^q - \varepsilon^s\right)\bar{X} \,. \tag{3.94}$$

The use of the variable learning rate $1/|z_r|$ makes movements of the weighted sum "softer". This is illustrated in Fig. 3.37. If the absolute value of the current



(a) |z|>1 (b) |z|>1

(c) |z|<1

Fig. 3.37 Movement of the weighted sum z after the correction of the weights according to (3.94)

weighted sum z is greater than 1 (it is located outside of the unit circle, see Fig. 3.37a), then $\frac{1}{|z|}<1$, and the weighted sum moves shorter than by $\varepsilon^q-\varepsilon^s$. It cannot reach in this way its target, the sector q, but it also could not reach it moving by $\varepsilon^q-\varepsilon^s$. However, it moved closer to the sector q, and on the next learning step it moves exactly there (see Fig. 3.37b). Moving by $\varepsilon^q-\varepsilon^s$, the adjusted weighted sum would need even one more step to move to the desired sector, while moving by $\frac{1}{|z|}\left(\varepsilon^q-\varepsilon^s\right)$, it does not need it. In Fig. 3.37c, the absolute value of the current weighted sum z is less than 1 (it is located inside the unit circle). Therefore $\frac{1}{|z|}>1$ and the weighted sum moves further than by $\varepsilon^q-\varepsilon^s$. It not only reaches its target – the desired sector q, but $\tilde{z}$ is located even more distant from the sector borders than z. The use of the learning rule (3.94) is especially reasonable for highly nonlinear input/output mappings, which may have multiple jumps, peaks, etc. Correcting the weights carefully, this learning rule makes the learning process more adaptive, which may lead to faster convergence of the learning algorithm. However, we will see below that the convergence of the learning algorithm with the error-correction learning rule does not depend on the particular form of this rule - (3.92) or (3.94). Moreover, in [61] another modification of the learning rules (3.92) or (3.94) was proposed by the author of this book.

To learn highly nonlinear input/output mappings, it might be reasonable to calculate the error not as a difference between the desired and actual outputs, but as a difference between the desired output and the projection of the current weighted sum on the unit circle $\delta=\varepsilon^q-\frac{z}{|z|}$.

This leads to the following modification of the learning rules (3.92) and (3.94), respectively:

$$W_{r+1}=W_r+\frac{C_r}{(n+1)}\left(\varepsilon^q-\frac{z_r}{|z_r|}\right)\bar{X}, \tag{3.95}$$

$$W_{r+1}=W_r+\frac{C_r}{(n+1)|z_r|}\left(\varepsilon^q-\frac{z_r}{|z_r|}\right)\bar{X}. \tag{3.96}$$

For the continuous MVN, the error-correction learning rule is derived from the same considerations. Just the error δ is not a difference of the kth roots of unity, but it is a difference of the desired output D and actual output Y, which can be arbitrary numbers located on the unit circle. Taking into account the $\delta=D-Y$ and that

$Y = \frac{z_r}{|z_r|}$, we obtain from (3.95) and (3.96) the following error-correction learning rules for the continuous MVN, respectively

$$W_{r+1} = W_r + \frac{C_r}{(n+1)}(D - Y)\bar{X} = W_r + \frac{C_r}{(n+1)}\left(D - \frac{z_r}{|z_r|}\right)\bar{X}, \quad (3.97)$$

$$W_{r+1} = W_r + \frac{C_r}{(n+1)|z_r|}(D - Y)\bar{X} = W_r + \frac{C_r}{(n+1)|z_r|}\left(D - \frac{z_r}{|z_r|}\right)\bar{X}. \quad (3.98)$$

3.3.2 Convergence Theorem

Now we can formulate and prove the theorem about the convergence of the learning algorithm for the discrete MVN with the learning rules (3.92), and (3.94)-(3.98). For the discrete MVN learning rules (3.92), and (3.94)-(3.96), the proof, which will be given here, was done by the author of this book in [61]. For the continuous MVN learning rules (3.97) and (3.98), the proof is based on the same approach and it is very similar. It will be given here for the first time. It is important to mention that for the discrete MVN learning algorithm with the learning rule (3.92) the convergence theorem was proven in [60]. The proof, which was done in [61] and will be presented here, is shorter and more elegant.

So let us have the learning set *A*. Suppose that the permutation $R = (\alpha_0, \alpha_1, ..., \alpha_{k-1})$ such that (3.73) holds for the entire learning set *A* is known. We use the learning algorithm presented in Section 3.1. Let us just make one important remark about the continuous MVN case. We have already mentioned in Section 3.1 that any learning set *A* for the continuous-valued input/output mapping $f(x_1, ..., x_n): T \to O, T \subseteq O^n$ is finite, which means that it can be represented as a union $A = \bigcup_{0 \le j \le k-1} A_j$ of the learning subsets $A_j, j = 0, 1, ..., k-1$, where *k* is the number of different values of the function $f(x_1, ..., x_n)$ representing a continuous input/output mapping.

Theorem 3.17 (About the convergence of the learning algorithm with the error-correction learning rules (3.92), (3.94)-(3.98))**.** If the learning subsets $A_0, A_1, ..., A_{k-1}$ of the learning set *A* ($A = \bigcup_{0 \le j \le k-1} A_j$) are *k*-separable for the given value of *k* according to Definition 3.16 (which means that the corresponding MVN input/output mapping is a *k*-valued threshold function), then the MVN learning

algorithm with either of the learning rules (3.92), (3.94)-(3.98) converges after a finite number of steps.

Proof. This proof is based on the same idea as the one of Theorem 3.16 and it is in major similar. Let us first proof the Theorem for the rule (3.92). Then we will prove it for the learning rules (3.94)-(3.98).

Since we are given a condition that our learning subsets $A_0, A_1, \ldots, A_{k-1}$ are *k*-separable, this means that there exists a weighting vector $W = (w_0, w_1, \ldots, w_n)$ such that (3.73) holds for any $X = (x_1, \ldots, x_n)$ from the domain of *f* and at least one of the weights is non-zero.

Let us now look for a weighting vector applying the learning rule (3.92) according to our learning algorithm. We may set $C_r = 1$ in (3.92) for any *r*. For simplicity and without loss of generality, let us start learning process from the zero vector $W_1 = ((0,0),(0,0),\ldots,(0,0))$, where (a,b) is a complex number $a+bi$, where i is an imaginary unity. Let $S_X = (X_1, X_2, \ldots, X_N)$ be a learning sequence of input vectors $X_j = (x_1^j, \ldots, x_n^j)$, $j = 1, \ldots, N$, and $S_W = (W_1, W_2, \ldots, W_r, \ldots)$ be a sequence of weighting vectors, which appear during the learning process. We have to prove that this sequence cannot be infinite. Let us remove from the learning sequence those vectors for which $W_{r+1} = W_r$, in other words, those input vectors, for which (3.73) hold without any learning. Let $S_{\tilde{W}}$ be the reduced sequence of the weighting vectors. The Theorem will be proven if we will show that the sequence $S_{\tilde{W}}$ is finite. Let us suppose that the opposite is true: the sequence $S_{\tilde{W}}$ is infinite. Let ε^{s_1} be the actual output for the input vector X_1 and the weighting vector W_1 and ε^{q_1} be the desired MVN output for the same input vector X_1. Since the desired and actual outputs do not coincide with each other, we have to apply the learning rule (3.92) to adjust the weights. According to (3.92) we obtain $\tilde{W}_2 = \frac{1}{n+1}\left(\varepsilon^{q_1} - \varepsilon^{s_1}\right)\bar{\bar{X}}_1$,

$$\tilde{W}_3 = \tilde{W}_2 + \frac{1}{n+1}\left(\varepsilon^{q_2} - \varepsilon^{s_2}\right)\bar{\bar{X}}_2 =$$

$$\frac{1}{n+1}\left[\left(\varepsilon^{q_1} - \varepsilon^{s_1}\right)\bar{\bar{X}}_1 + \left(\varepsilon^{q_2} - \varepsilon^{s_2}\right)\bar{\bar{X}}_2\right], \ldots.$$

$$\tilde{W}_{r+1} = \frac{1}{n+1}\left[\left(\varepsilon^{q_1} - \varepsilon^{s_1}\right)\bar{\bar{X}}_1 + \ldots + \left(\varepsilon^{q_r} - \varepsilon^{s_r}\right)\bar{\bar{X}}_r\right]. \tag{3.99}$$

Let us find a dot product of both parts of (3.99) with W:

$$\left(\tilde{W}_{r+1}, \bar{W}\right) = \frac{1}{n+1}\left[\left(\left(\varepsilon^{q_1} - \varepsilon^{s_1}\right)\bar{\bar{X}}_1, \bar{W}\right) + \ldots + \left(\left(\varepsilon^{q_r} - \varepsilon^{s_r}\right)\bar{\bar{X}}_r, \bar{W}\right)\right].$$

Let $\varepsilon^{q_j} - \varepsilon^{s_j} = \omega_j$, $j = 1, \ldots, r$.

Then the last equation may be rewritten as follows:

$$\left(\tilde{W}_{r+1}, \bar{W}\right) = \frac{1}{n+1}\left[\left(\omega_1\bar{\bar{X}}_1, \bar{W}\right) + \ldots + \left(\omega_r\bar{\bar{X}}_r, \bar{W}\right)\right]. \tag{3.100}$$

Let us estimate the absolute value $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$:

$$\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| = \frac{1}{n+1}\left|\left[\left(\omega_1\bar{\bar{X}}_1, \bar{W}\right) + \ldots + \left(\omega_r\bar{\bar{X}}_r, \bar{W}\right)\right]\right|. \tag{3.101}$$

Since for any complex number β $|\beta| \geq |\mathrm{Re}\,\beta|$ and $|\beta| \geq |\mathrm{Im}\,\beta|$, then the absolute value of the sum in the right-hand side of (3.101) is always greater than or equal to the absolute values of the real and imaginary parts of this sum. Let $a = \min\limits_{j=1,\ldots,r}\left|\mathrm{Re}\left(\omega_j\tilde{X}_j, \bar{W}\right)\right|$. Then it follows from (3.101) that

$$\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| \geq \frac{ra}{n+1}. \tag{3.102}$$

According to the fundamental Schwarz inequality [74] the squared dot product of the two vectors does not exceed the product of the squared norms of these vectors or in other words, the norm of the dot product of the two vectors does not exceed the product of the norms of these vectors $\left\|\left(V_1, V_2\right)\right\| \leq \left\|V_1\right\| \cdot \left\|V_2\right\|$. Thus, according to the Schwartz inequality

$$\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| \leq \left\|\tilde{W}_{r+1}\right\| \cdot \left\|W\right\|. \tag{3.103}$$

Taking into account (3.102), we obtain from (3.103) the following

$$\frac{ra}{n+1} \leq \left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right| \leq \left\|\tilde{W}_{r+1}\right\| \cdot \left\|W\right\|.$$

Then it follows from the last inequality that

$$\left\|\tilde{W}_{r+1}\right\| \geq \frac{ra}{\|W\|(n+1)}. \tag{3.104}$$

Let for simplicity $\frac{a}{n+1} = \tilde{a}$. Then (3.104) is transformed as follows:

$$\left\|\tilde{W}_{r+1}\right\| \geq r\tilde{a} / \|W\|. \tag{3.105}$$

As we told, W is some weighting vector, which exists for our input/output mapping. This vector exists according to the condition of the Theorem because the learning subsets $A_0, A_1, ..., A_{k-1}$ are k-separable. According to our assumption, the sequence $S_{\tilde{W}}$ of the weighting vectors is infinite. Since r is the number of the learning step, let us consider (3.105) when $r \to \infty$. $\left\|\tilde{W}_{r+1}\right\|$ is a non-negative finite real number, $\|W\|$ is a finite positive real number ($\|W\| \neq 0$ because vector W is a weighting vector for our input/output mapping, and this means that at least one of the weights is not equal to 0), and $\tilde{a}$ is a finite positive real number. It follows from this analysis that

$$\frac{r\tilde{a}}{\|W\|} \underset{r\to\infty}{\to} \infty.$$

However, this means that from (3.105) we obtain

$$\left\|\tilde{W}_{r+1}\right\| \geq \frac{r\tilde{a}}{\|W\|} \to \infty. \tag{3.106}$$

Inequality (3.106) is contradictory. Indeed, the norm of a vector, which is in the left-hand side, is a finite non-negative real number. However, it has to be greater than or equal to the infinity in the right-hand side of (3.106), which is impossible. This means that (3.106) is contradictory. This means in turn that either it is impossible that $r \to \infty$ or the vector W does not exist. The latter means that the learning subsets $A_0, A_1, ..., A_{k-1}$ are not k-separable. However, this contradicts to the condition of the Theorem. Hence, $r \not\to \infty$ and it is always a finite integer number. Thus, our assumption that the sequence $S_{\tilde{W}}$ of the weighting vectors is infinite, is false, which means that it is always finite. Hence, the learning algorithm with the learning rule (3.92) converges after a finite number of steps.

Let us now prove that the learning algorithm also converges when either of the learning rules (3.94)-(3.98) is used. For these three learning rules the proof of the

convergence of the learning algorithm is almost identical to the proof we have just presented, accurate within specifics of some equations. Let us demonstrate this.

If we apply the learning rule (3.94), we obtain the following equation instead of (3.99)

$$\tilde{W}_{r+1} = \frac{1}{n+1}\left[\frac{1}{|z_1|}\left(\varepsilon^{q_1} - \varepsilon^{s_1}\right)\bar{\bar{X}}_1 + \ldots + \frac{1}{|z_r|}\left(\varepsilon^{q_r} - \varepsilon^{s_r}\right)\bar{\bar{X}}_r\right].$$

Then putting $\frac{1}{|z_j|}\left(\varepsilon^{q_j} - \varepsilon^{s_j}\right) = \omega_j,\ j = 1,\ldots,r$, we obtain (3.100) and from that moment the proof continues with no changes.

If we apply the learning rule (3.95), then (3.99) is substituted by the following expression

$$\tilde{W}_{r+1} = \frac{1}{n+1}\left[\left(\varepsilon^{q_1} - \frac{z_1}{|z_1|}\right)\bar{\bar{X}}_1 + \ldots + \left(\varepsilon^{q_r} - \frac{z_r}{|z_r|}\right)\bar{\bar{X}}_r\right].$$

Then putting $\varepsilon^{q_j} - \frac{z_j}{|z_j|} = \omega_j,\ j = 1,\ldots,r$, we obtain (3.100) and from that moment the proof continues again with no changes.

If we apply the learning rule (3.96), then we again have to substitute (3.99), this time as follows

$$\tilde{W}_{r+1} = \frac{1}{n+1}\left[\frac{1}{|z_1|}\left(\varepsilon^{q_1} - \frac{z_1}{|z_1|}\right)\bar{\bar{X}}_1 + \ldots + \frac{1}{|z_r|}\left(\varepsilon^{q_r} - \frac{z_r}{|z_r|}\right)\bar{\bar{X}}_r\right].$$

Then putting $\frac{1}{|z_j|}\left(\varepsilon^{q_j} - \frac{z_j}{|z_j|}\right) = \omega_j,\ j = 1,\ldots,r$, we obtain (3.100) and from that moment the proof continues again with no changes.

If we apply the learning rule (3.97), then we again have to substitute (3.99), this time as follows

$$\tilde{W}_{r+1} = \frac{1}{n+1}\left[\left(D - \frac{z_1}{|z_1|}\right)\bar{\bar{X}}_1 + \ldots + \left(D - \frac{z_r}{|z_r|}\right)\bar{\bar{X}}_r\right].$$

Then putting $\left(D - \frac{z_j}{|z_j|}\right) = \omega_j,\ j = 1,\ldots,r$, we obtain (3.100) and from that moment the proof continues again with no changes.

If we apply the learning rule (3.98), then (3.99) should be substituted as follows

$$\tilde{W}_{r+1} = \frac{1}{n+1}\left[\frac{1}{|z_1|}\left(D - \frac{z_1}{|z_1|}\right)\bar{\bar{X}}_1 + ... + \frac{1}{|z_r|}\left(D - \frac{z_r}{|z_r|}\right)\bar{\bar{X}}_r\right].$$

Then putting $\frac{1}{|z_j|}\left(D - \frac{z_j}{|z_j|}\right) = \omega_j,\ j = 1,...,r$, we obtain (3.100) and from that moment the proof continues again with no changes.

Theorem is proven. This means that the MVN learning algorithm with either of the learning rules (3.92), (3.94)-(3.98) converges after a finite number of learning iterations. As well as the learning algorithm based on the rule (3.80), the learning algorithm based on the rules (3.92), (3.94)-(3.98) is derivative-free. As well as the algorithm based on the rule (3.80), it is not considered as the optimization problem of the minimization of the error functional. Therefore, a local minima problem (see Section 1.3, Fig. 1.11), which is typical for those learning rules that are based on the optimization technique, does not exist for the MVN learning algorithm based on the learning rules (3.92), (3.94)-(3.98), as well as for the learning algorithm based on the rule (3.80).

The error in the MVN learning algorithm based on the learning rules (3.92), (3.94)-(3.98) naturally decreases because of the same reasons that for the learning algorithm based on the learning rule (3.80). Each following weighting vector in the sequence $S_{\tilde{W}}$ should be closer to the "ideal" weighting vector *W*, which exists if the MVN input/output mapping is described by some *k*-valued threshold function and the learning subsets $A_0, A_1, ..., A_{k-1}$ are *k*-separable. According to (3.99), for the learning rule (3.92), and according to corresponding equations for the learning rules (3.94)-(3.98), the absolute value of the dot product of the vector *W* and the weighting vector W_{r+1} in the sequence $S_{\tilde{W}}$ should not decrease and moreover, as it follows from (3.103) and (3.104), it must be greater than or equal to the finite number proportional to *r*, which is the number of the correction. On the one hand, since *r* increases, this means that $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$ should not decrease. On the other hand, as we have proven, $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$ cannot increase to infinity. This means that the learning algorithm converges when $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$ reaches its maximum. This means that vectors W_{r+1} and W are as close to each other as it is possible. Ideally, they are collinear or close to collinearity. It follows from (3.99) and (3.100) that $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$ cannot decrease during the learning process. It may only increase or remain the same. If it increases, this means that the error decreases. This means that geometrically, the MVN learning algorithm based on

either of the learning rules (3.92), (3.94)-(3.98) "rotates" the initial weighting vector and the intermediate weighting vectors W_r such that $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$ should be maximized. It follows from this that the worst starting condition for the learning process is when the vectors W_1 (the starting weighting vector) and W are orthogonal to each other and $\left|\left(\tilde{W}_{r+1}, \bar{W}\right)\right|$=0, while the best starting condition is when the same vectors are about collinear. The closer they are to the collinearity, the smaller is the error and the shorter way is required for the convergence of the learning process.

Hence, the MVN learning algorithm based on the error-correction learning rule is another example of the "non-optimization" learning, which is based on the same idea that was developed in [12] by A. Novikoff for the threshold neuron and its error-correction learning.

It is important that a beautiful approach to the learning through the direct error-correction, which was proposed by F. Rosenblatt and developed by A. Novikoff for the threshold neuron about 50 years ago, was generalized for MVN. Unlike the threshold neuron, MVN employs this learning rule for multiple-valued and even continuous-valued input/output mappings.

We should recall that if the permutation $R = \left(\alpha_0, \alpha_1, ..., \alpha_{k-1}\right)$ such that (3.73) holds for the entire learning set *A* is not known, it is possible to find such a permutation by *k*! means.

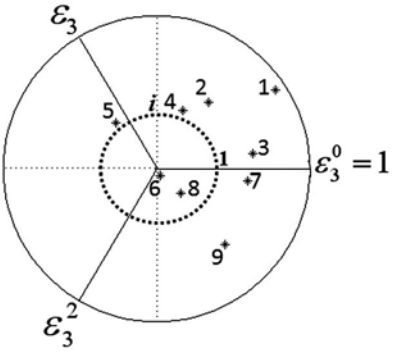
It is worth to mention that the MVN learning algorithm does not depend on the learning rate, unlike any learning algorithm for a real-valued neuron. As we saw, the learning rate C_r in the learning rules (3.80), and (3.92), (3.94)-(3.98) can always be equal to 1. Evidently, that a variable learning rate $\frac{1}{|z_r|}$ in (3.94), (3.96), and (3.98) is self-adaptive.

3.3.3 Example of Error-Correction Learning

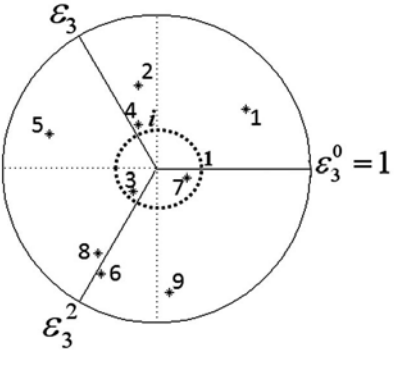
Let us consider how MVN learns. We will use the learning algorithm with the error-correction rule (3.92). We have already considered above (see Table 2.8 and Fig. 2.22a) how the discrete MVN implements in 3-valued logic the Post function $\max\left(y_1, y_2\right); y_i \in K; i = 1, 2$, which becomes in *k*-valued logic over the field of complex numbers $f_{\max}\left(x_1, x_2\right) = \max\left(x_1, x_2\right); x_i \in E_k; i = 1, 2$ Let us consider it again for $k = 3$. We will obtain the weighting vector for this function using the learning algorithm with the error-correction learning rule (3.92).

The results are summarized in Table 3.10 and Fig. 3.38. The learning process starts from the random weighting vector $W_0 = \left(0.96 + 0.32i, 0.79 + 0.73i, 0.59 + 0.5i\right)$ (all real and imaginary parts of

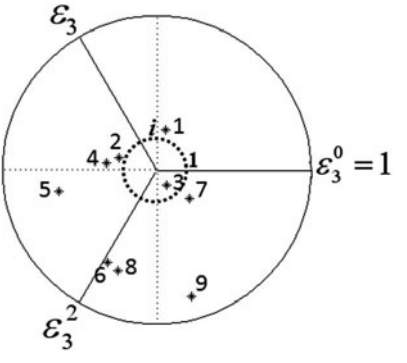
the weights are random numbers in the interval [0, 1]). With the initial weighting vector, the actual outputs for the learning samples 1, 5, 6, 7, 8, 9 coincide with the desired outputs, while for the learning samples 2, 3, 4 the desired outputs are incorrect (see Fig. 3.38a and the column "Initial *W*" in Table 3.10). We have already shown in Section 2.1 that $f_{\max}(x_1, x_2)$ is a threshold function, which can be implemented using MVN with the weighting vector $W = (-2-4\varepsilon_3, 4+5\varepsilon_3, 4+5\varepsilon_3)$.



(a) with the initial random vector



(b) after the 1st iteration



(c) after the 2nd iteration

Fig. 3.38 Learning of the $f_{\max}(x_1, x_2) = \max(x_1, x_2)$ for *k*=3, using the MVN learning algorithm with the rule (3.92).

Locations of the weighted sums corresponding to the learning samples 1-9 are shown

Thus, we may consider this weighting vector as the "ideal" one. According to Theorem 3.16 and Theorem 3.17, if the learning process starts from some arbitrary weighting vector whose components are chosen randomly, then this process should lead to the weighting vector, whose absolute dot product with the "ideal" weighting vector reaches its maximum. Moreover, the absolute value of this dot product should not decrease during the learning process.

For the starting weighting vector W_0 we obtain $\left|\left(W_0, W\right)\right| = 6.34$. After the first learning iteration the actual outputs for the learning samples 1, 3, 5, 6, 7, 9 coincide with the desired outputs, while for the learning samples 2, 4, 8 the desired outputs are incorrect (see Fig. 3.38b and the column "Iteration 1" in Table 3.10).

Table 3.10 MVN learns the Post function $f_{\max}(x_1, x_2)$ in 3-valued logic using the learning algorithm with the error-correction learning rule (3.92)

#	x_1	x_2	Initial W		Iteration 1		Iteration 2		$f_{\max}(x_1, x_2)$
			arg(z)	P(z)	arg(z)	P(z)	arg(z)	P(z)	
1	ε_3^0	ε_3^0	0.585	ε_3^0	0.585	ε_3^0	1.356	ε_3^0	ε_3^0
2	ε_3^0	ε_3^1	0.909	ε_3^0	1.788	ε_3^0	2.850	ε_3^1	ε_3^1
3	ε_3^0	ε_3^2	0.152	ε_3^0	3.896	ε_3^1	5.300	ε_3^2	ε_3^2
4	ε_3^1	ε_3^0	0.138	ε_3^0	1.968	ε_3^0	3.007	ε_3^1	ε_3^1
5	ε_3^1	ε_3^1	2.296	ε_3^1	2.832	ε_3^1	3.359	ε_3^1	ε_3^1
6	ε_3^1	ε_3^2	5.231	ε_3^2	4.223	ε_3^2	4.220	ε_3^2	ε_3^2
7	ε_3^2	ε_3^0	6.156	ε_3^2	6.001	ε_3^2	5.580	ε_3^2	ε_3^2
8	ε_3^2	ε_3^1	5.502	ε_3^2	4.105	ε_3^1	4.342	ε_3^2	ε_3^2
9	ε_3^2	ε_3^2	5.444	ε_3^2	4.817	ε_3^2	4.976	ε_3^2	ε_3^2

It should be mentioned that the weighted sums for the learning samples 2 and 4 have moved much closer to the desired sector 1 compared to the initial state, while the weighted sum for the learning sample 8 has moved a little bit in the incorrect direction from the correct sector 2 to the incorrect sector 1. After the first learning iteration, the updated weighting vector is $W_1 = \left(-1.04 - 0.84i, 0.79 + 1.31i, 0.59 + 1.08i\right)$, and $\left|\left(W_1, W\right)\right| = 16.46$.

After the second learning iteration, all actual outputs coincide with the desired outputs (see Fig. 3.38c and the column "Iteration 2" in Table 3.10) and therefore, the learning algorithm converges. The resulting weighting vector is

$W_2 = (-1.04 - 1.99i, 0.79 + 1.89i, 0.59 + 1.65i)$, and $|(W_2, W)| = 24.66$. We see that vectors W and W_2 are not collinear, but they are close to collinearity (at least, all the real and imaginary parts of their components have the same sign) and this closeness is enough to ensure that both these vectors implement the same input/output mapping.

In Chapter 4, we will use the error-correction learning in the backpropagation learning algorithm for a feedforward neural network based on multi-valued neurons.

A level of growing of $|(W_r, W)|$ where *r* is the number of the learning iteration should be used as a measure of the *learning energy*, which the learning algorithm spends correcting the weights. We will see in Section 3.4 that the best choice for the starting weighting vector in the learning algorithm is the Hebbian weighting vector that is the vector obtained using the Hebb rule. The learning process, which starts from the Hebbian vector leads to fewer corrections of the weights than the learning process starting from the random vector.

3.4 Hebbian Learning and MVN

We have started consideration of different neural learning techniques from the Hebbian learning (see Section 1.2). Let us consider how this important learning technique works for MVN.

The mechanism of the Hebbian learning for MVN is the same as the one for the threshold neuron and as it was described by D. Hebb in his seminal book [8]. This is the mechanism of the association. The weight should pass the input signal or to enhance it or to weaken it depending on the correlation between the corresponding input and the output. As well as for the binary threshold neuron, the associations between the desired outputs and the given inputs should be developed through the dot product of the vector of all the desired outputs with the corresponding vectors of all the given inputs.

The Hebbian learning rule does not change for MVN, and equations (1.3) and (1.4) that describe this rule for the threshold neuron also work for MVN.

Just for the reader's convenience we will repeat these equations here.

Let us have *N* *n*-dimensional learning samples $(x_1^j, ..., x_n^j)$, $j = 1, ..., N$. Let $\mathrm{f} = (f_1, ..., f_N)^T$ be an *N*-dimensional vector-column of the desired outputs. Let $\mathbf{x}_1, ..., \mathbf{x}_n$ be *N*-dimensional vectors of all the inputs ($\mathbf{x}_1 = (x_1^1, x_1^2, ..., x_1^N)^T$, $\mathbf{x}_2 = (x_2^1, x_2^2, ..., x_2^N)^T$, …, $\mathbf{x}_n = (x_n^1, x_n^2, ..., x_n^N)^T$).

Then according to the Hebbian learning rule (see (1.3)) the weights $w_1, ..., w_n$ are calculated as dot products of vector f and vectors $\mathbf{x}_1, ..., \mathbf{x}_n$, respectively.

Weight w_0 is calculated as a dot product of vector $\mathbf{f}$ and the *N*-dimensional vector-constant $\mathbf{x}_0 = (1,1,...,1)^T$:

$$w_i = (\mathbf{f}, \mathbf{x}_i), i = 0,...,n,$$

where $(\mathbf{a},\mathbf{b}) = a_1\overline{b}_1 + ... + a_n\overline{b}_n$ is the dot product of vector-columns $\mathbf{a} = (a_1,...,a_n)^T$ and $\mathbf{b} = (b_1,...,b_n)^T$ in the unitary space ("bar" is a symbol of complex conjugation), thus

$$w_i = (\mathbf{f}, \mathbf{x}_i) = f_1\overline{x}_i^1 + f_2\overline{x}_i^2 + ... + f_N\overline{x}_i^N, i = 0,1,...,n. \quad (3.107)$$

Equation (1.4) determines the normalized version of the Hebbian learning rule

$$w_i = \frac{1}{N}(\mathbf{f}, \mathbf{x}_i), i = 0,...,n.$$

Let us now consider the following examples. Let *k*=4 in the MVN activation function (2.50). Thus, our MVN works in 4-valued logic whose values are encoded by the elements of the set $E_4 = \{1, i, -1, -i\}$ ($i = \varepsilon_4 = e^{i2\pi/4}$ is an imaginary unity and a primitive 4th root of a unity).

Let us first consider four examples illustrated in Fig. 3.39. In all these examples we calculate a weight for one of the MVN inputs and for a single learning sample.

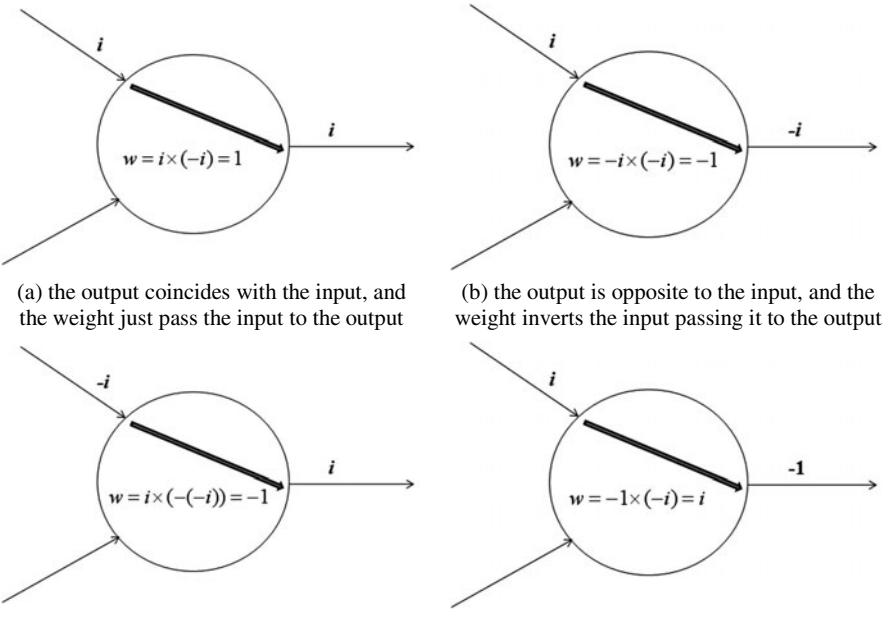
In Fig. 3.39a, the desired MVN output is *i* and the corresponding input is also *i*. According to (3.107) $w_0 = i, w_1 = f_1\overline{x}_1 = i \cdot (-i) = 1$, the weighted sum is $i + 1 \cdot i = 2i$, and according to (2.50) the neuron output is $P(2i) = i$. Thus, if the desired output coincides with the input, the weight just "passes" the input to the output.

In Fig. 3.39b, the desired MVN output is *-i* and the corresponding input is *i*. According to (3.107) $w_0 = -i, w_1 = f_1\overline{x}_1 = -i \cdot (-i) = -1$, the weighted sum is $-i + (-1) \cdot i = -2i$, and according to (3.50) the neuron output is $P(-2i) = -i$. Thus, if the desired output is opposite to the input, the weight inverts the input passing it to the output.

The same situation is illustrated in Fig. 3.39c. Just the desired MVN output here is *i*, while the corresponding input is *-i*. According to (3.107) $w_0 = i, w_1 = f_1\overline{x}_1 = i \cdot i = -1$, the weighted sum is $i + (-1) \cdot (-i) = 2i$, and according to (3.50) $P(2i) = i$.

In Fig. 3.39d, the desired output and the input are neither the same nor opposite to each other. The desired MVN output is -1 and the corresponding input is *i*.

According to (3.107) $w_0 = -1, w_1 = f_1\overline{x}_1 = -1 \cdot (-i) = i$, the weighted sum is $-1 + i \cdot i = -1 - 1 = -2$, and according to (2.50) the neuron output is $P(-2) = -1$. Thus, the weight "rotates" the input such that this input contributes to the desired output.



(a) the output coincides with the input, and the weight just pass the input to the output

(b) the output is opposite to the input, and the weight inverts the input passing it to the output

(c) the output is opposite to the input, and the weight inverts the input passing it to the output

(d) the weight "rotates" the input such that this input contributes to the desired output

Fig. 3.39 Calculation of the MVN weight using the Hebb rule for one of the neuron inputs and for a single learning sample: the weight is equal to the product of the desired output and the complex-conjugated input

Let us now consider calculation of the weights for the two MVN inputs using the Hebbian learning rule.

In Fig. 3.40a, the desired MVN output is i, while its two inputs are i and -1, respectively. According to (3.107) $w_0 = i, w_1 = f_1\overline{x}_1 = i \cdot (-i) = 1$, and $w_2 = f_1\overline{x}_2 = i \cdot (-1) = -i$, the weighted sum is $i + 1 \cdot i + (-1) \cdot (-i) = 3i$ and according to (2.50) the neuron output is $P(3i) = i$. Thus, the weight w_1 passes the input x_1 to the output, while the weight w_2 "rotates" the input x_2 passing it to the output.

In Fig. 3.40b, the desired MVN output is -*i*, while its two inputs are *i* and 1, respectively. According to (3.107) $w_0 = -i, w_1 = f_1\overline{x}_1 = -i \cdot (-i) = -1$ and $w_2 = f_1\overline{x}_2 = -i \cdot 1 = -i$, the weighted sum is $-i + (-1) \cdot i + (-i) \cdot 1 = -3i$, and according to (2.50) the neuron output is $P(-3i) = -i$. Thus, the weight w_1 inverts the input x_1 passing it to the output, while the weight w_2 "rotates" the input x_2 passing it to the output.

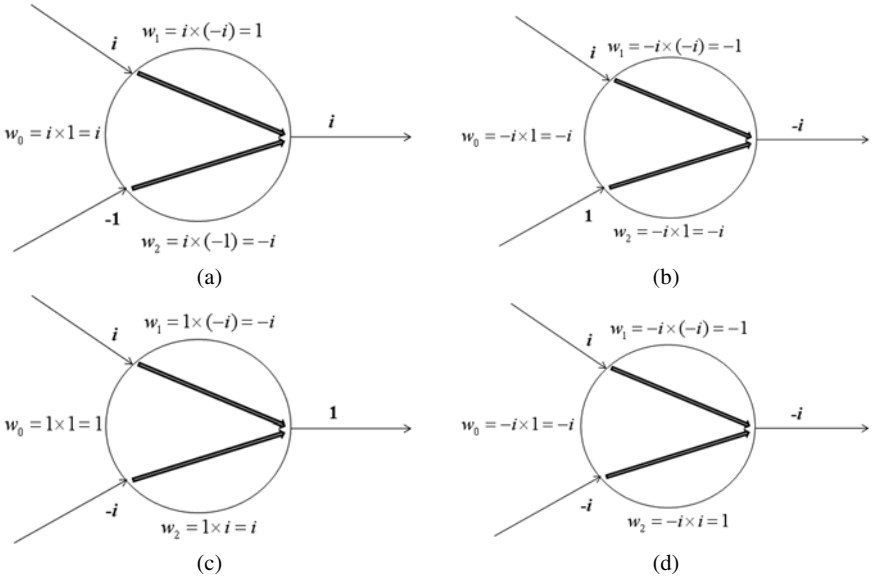


Fig. 3.40 Calculation of the MVN weights using the Hebb rule for the two neuron inputs and for a single learning sample: the weight is equal to the product of the desired output and the complex-conjugated input

In Fig. 3.40c, the desired MVN output is 1, while its two inputs are *i* and -*i*, respectively. According to (3.107) $w_0 = 1, w_1 = f_1\overline{x}_1 = 1 \cdot (-i) = -i$, and $w_2 = f_1\overline{x}_2 = 1 \cdot i = i$, the weighted sum is $1 + (-i) \cdot i + (i) \cdot (-i) = 3$, and according to (2.50) the neuron output is $P(3) = 1$. Thus, both weights w_1 and w_2 "rotate" the inputs x_1 and x_2 passing them to the output.

In Fig. 3.40d, the desired MVN output is -*i*, while its two inputs are *i* and -*i*, respectively. According to (3.107) $w_0 = -i, w_1 = f_1\overline{x}_1 = -i \cdot (-i) = -1$ and $w_2 = f_1\overline{x}_2 = -i \cdot i = 1$, the weighted sum is $-i + (-1) \cdot i + 1 \cdot (-i) = -3i$, and

according to (2.50) the neuron output is $P(-3i) = -i$. Thus, the weight w_1 inverts the input x_1 passing it to the output, while the weight w_2 passes the input x_2 to the output.

Evidently, the Hebbian learning can also be considered for the continuous MVN. In this case, nothing changes, and the Hebb rule still is still described by (1.3), (1.4), and (3.107).

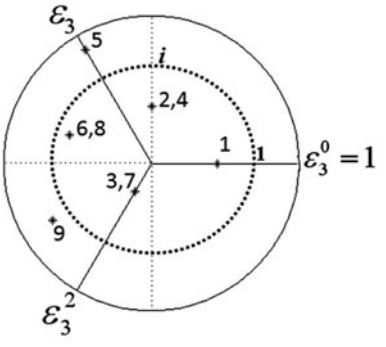
It is important that the MVN Hebbian learning can be used for simulation of the associations that take place in biological neurons when they learn. We have already discussed above (Section 2.4) that the information transmitted by biological neurons to each other is completely contained in the frequency of the generated spikes. The phase, which determines the MVN state, is proportional to the frequency. Thus, the larger is phase, the higher is frequency. The reader may consider examples shown in Fig. 3.39 and Fig. 3.40 from the point of view of simulation of the biological neuron learning. In this case, the neuron states should be interpreted as follows: 1 – "inhibition" (phase 0), i – slight excitation (phase $\pi/2$), -1 – moderate excitation (phase π), and $-i$ – maximal excitation (phase $3\pi/2$).

In all examples of the Hebbian learning, which we have considered above, the learning set has contained a single learning sample. When there are more learning samples in the learning set, the Hebbian learning rule usually does not lead to a weighting vector, which implements the corresponding input/output mapping. However, there is a wonderful property of the weighting vector obtained using the Hebbian learning rule. Although this vector usually does not implement the corresponding input/output mapping, the MVN learning algorithm based on the learning rules (3.80) and (3.92), (3.94)-(3.98) converges much faster when the learning process starts from this (Hebbian) vector than from a random vector.

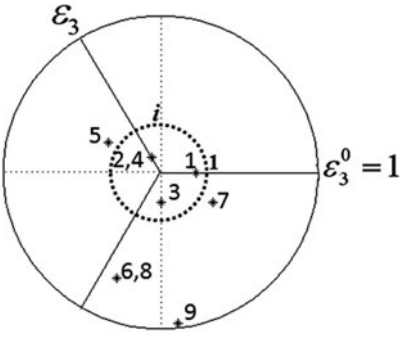
We can illustrate this property using the example of learning the input/output mapping presented by the function $f_{\max}(x_1, x_2) = \max(x_1, x_2)$ for k=3, which we have already used several times. As it was shown in Section 2.1, this is a 3-valued threshold function, which can be implemented using MVN with the weighting vector $W = (-2 - 4\varepsilon_3, 4 + 5\varepsilon_3, 4 + 5\varepsilon_3)$. Thus, we may consider this weighting vector as the "ideal" one. According to Theorem 3.16 and Theorem 3.17, if the learning process starts from some arbitrary weighting vector whose components are chosen randomly, then this process should lead us to the weighting vector, whose absolute dot product with the "ideal" weighting vector reaches its maximum.

Let us find the Hebbian weights for $f_{\max}(x_1, x_2)$. According to (1.4) and taking into account (3.107), we obtain the following Hebbian weighting vector $W_H = (-0.33 + 0.19i, 0.5 - 0.096i, 0.5 - 0.096i)$. This weighting vector does not implement the function $f_{\max}(x_1, x_2)$. Distribution of the weighted sums with the weighting vector W_H is shown in Fig. 3.41a. The outputs for five

learning samples out of nine (samples 2, 4, 6, 8, 9) are incorrect (see the column "Hebbian Weights " in Table 3.11). However, they can easily be corrected using the learning algorithm, for example, with the error-correction rule (3.92). Moreover, the number of corrections of the weights is fewer than for the same learning algorithm when it starts from the random weighting vector.



(a) Distribution of the weighted sums with the Hebbian weighting vector



(b) Iteration 1

Fig. 3.41 Movement of the weighted sum z after the correction of the weights according to (3.92) starting from the Hebbian weighting vector for the function

$$f_{\max}(x_1, x_2) = \max(x_1, x_2); k = 3$$

After a single learning iteration the actual outputs for all the learning samples coincide with the desired outputs (see Fig. 3.41b and the column "Iteration 1" in Table 3.11).

Let us evaluate the energy, which we have to spend for the learning, which starts from the Hebbian weights in terms of growing of the absolute value of the dot product $\left|(W_r, W)\right|$ of the current weighting vector W_r and the "ideal" weighting vector W.

For the Hebbian weighting vector W_H we obtain $\left|(W_H, W)\right| = 5.77$. After a single learning iteration, for the weighting vector $W_1 = (-0.33 - 1.35i, 0.5 + 0.67i, 0.5 + 0.67i)$ resulted from this iteration we obtain $\left|(W_1, W)\right| = 12.47$. Comparing this result to the one considered in Section 3.3 for the learning process started for the same input/output mapping from the random weighting vector, we see that not only a single iteration was enough for the convergence of the learning algorithm, but significantly smaller amount of the corrections of the weights is required for the learning process, which starts from the Hebbian weights. In fact, in the example with the learning

algorithm started from the random weights (see Section 3.3) the absolute value of the dot product $\left|(W_r, W)\right|$ was 6.34 for the initial weights, 16.46 after the first iteration, and 24.66 after the second iteration. Hebbian weights ensure that correcting the weights for some learning sample, we do not corrupt the weights for other learning samples. Moreover, we may simultaneously improve the result for some other learning samples, which require correction of the weights.

Table 3.11 MVN learns the Post function $f_{\max}(x_1, x_2)$ in 3-valued logic using the learning algorithm with the error-correction learning rule (3.92) and starting from the Hebbian weighting vector W_H

#	x_1	x_2	Hebbian Weights		Iteration 1		$f_{\max}(x_1, x_2)$
			$\arg(z)$	$P(z)$	$\arg(z)$	$P(z)$	
1	ε_3^0	ε_3^0	0.0	ε_3^0	0.0	ε_3^0	ε_3^0
2	ε_3^0	ε_3^1	1.571	ε_3^0	2.095	ε_3^1	ε_3^1
3	ε_3^0	ε_3^2	4.188	ε_3^2	4.712	ε_3^2	ε_3^2
4	ε_3^1	ε_3^0	1.571	ε_3^0	2.095	ε_3^1	ε_3^1
5	ε_3^1	ε_3^1	2.094	ε_3^1	2.618	ε_3^1	ε_3^1
6	ε_3^1	ε_3^2	2.808	ε_3^1	4.321	ε_3^2	ε_3^2
7	ε_3^2	ε_3^0	4.188	ε_3^2	5.759	ε_3^2	ε_3^2
8	ε_3^2	ε_3^1	2.808	ε_3^1	4.321	ε_3^2	ε_3^2
9	ε_3^2	ε_3^2	3.665	ε_3^1	4.827	ε_3^2	ε_3^2

As we see comparing the vectors W_H and W_1, only imaginary parts of the weights required correction.

Comparing the vectors W and W_1, we see that they are not collinear, but close to collinearity. At least, the real and imaginary parts of their components have the same sign.

3.5 Concluding Remarks to Chapter 3

In this Chapter, we have considered fundamentals of MVN learning. If the input/output mapping is a *k*-valued threshold function, it can be learned by MVN. It

was shown that in this case, a learning set consists of k learning subsets, which are k-separable. The MVN learning algorithm is based on the sequential iterative examination of the learning samples and correction of the weights using a learning rule wherever it is necessary.

The learning process may continue until the zero error is reached or until the mean square error (or the root mean square error) drops below some reasonable predetermined value.

We have considered two learning rules. The first rule is based on the closeness of the actual output to the desired one in terms of angular distance. The second learning rule is the error-correction learning rule. The convergence theorems for the MVN learning algorithm based on both learning rules were proven. If the MVN input/output mapping is described by some k-valued threshold function, which means that a learning set corresponding to this mapping consists of k disjoint k-separable subsets, then the MVN learning algorithm based on either of the considered learning rules converges.

It is fundamental that the MVN learning is based on the same principles as the perceptron learning in A. Novikoff's interpretation. It is not considered as the optimization problem of the error functional minimization. It is shown that each step of the learning process decreases the distance between the current weighting vector and the "ideal" weighting vector, which exists because the input/output mapping is a k-valued threshold function. The more this distance decreases, the more the error decreases too, and the iterative learning process always converges after a finite number of iterations.

We have also considered the Hebbian learning for MVN. It was shown that the Hebbian learning rule works for MVN in the same manner as for the threshold neuron. It builds associations between the inputs and desired outputs. We have also shown that Hebbian weights, even when they cannot implement the input/output mapping, should be optimal starting weights for the MVN learning algorithm, leading to fewer corrections of the weights rather than starting from the arbitrary random vector.

So we have considered all the MVN fundamentals, its mathematical background, its organization, and its learning rules.

Now we are ready to consider how MVN works in networks and first of all in the feedforward neural network.