

Chapter 2

Neural Networks with Feedback and Self-organization

In this chapter another classes of neural networks are considered as compared with feed-forward NN—NN with back feed and with self-organization.

In Sect. 2.1 recurrent neural network of Hopfield is considered, its structure and properties are described. The method of calculation of Hopfield network weights is presented and its properties considered and analyzed. The results of experimental investigations for application Hopfield network for letters recognition under high level of noise are described and discussed. In the Sect. 2.2 Hamming neural network is presented, its structure and properties are considered, algorithm of weights adjusting is described. The experimental investigations of Hopfield and Hamming networks in the problem of characters recognition under different level of noise are presented. In the Sect. 2.3 so-called self-organizing networks are considered. At the beginning Hebb learning law for neural networks is described. The essence of competitive learning is considered. NN with self-organization by Kohonen are described. The basic competitive algorithm of Kohonen is considered/its properties are analyzed. Modifications of basic Kohonen algorithm are described and analyzed. The modified competitive algorithm with neighborhood function is described. In the Sect. 2.4 different applications of Kohonen neural networks are considered: algorithm of neural gas, self-organizing feature maps (SOMs), algorithms of their construction and applications.

2.1 Neural Network of Hopfield

Revival of interest in neural networks is connected with Hopfield's work (1982) [1]. This work shed light on that circumstance that neuron networks can be used for the computing purposes. Researchers of many scientific fields received incentive for further researches of these networks; pursuing thus the double aim: the best understanding of how the brain works and application of properties of these networks.

2.1.1 Idea of Recurrency

The neural network of Hopfield is an example of a network which can be defined as a dynamic system with feedback at which the output of one completely direct operation serves as an input of the following operation of a network, as shown in Fig. 2.1.

Networks which work as systems with feedback, are called “recurrent networks”. Each direct operation of a network is called an iteration. Recurrent networks, like any other nonlinear dynamic systems, are capable to show the whole variety of different behavior. In particular, one possible pattern of behavior is that the system can be stable, i.e. it can converge to the only fixed (motionless) point.

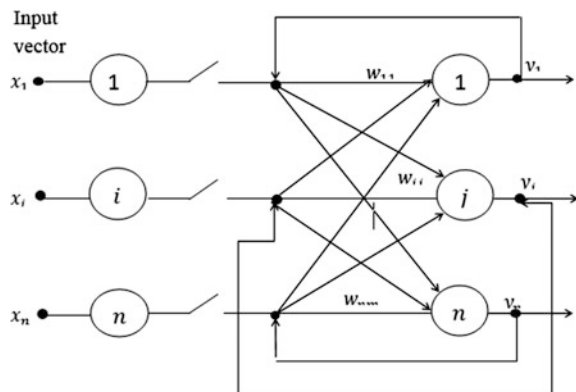
When the motionless point is an input to such dynamic system, at the output we will have the same point. Thus the system remains fixed at the same state. Periodic cycles or chaotic behavior are also possible.

It was shown that Hopfield’s networks are stable. In general case it may be more than one fixed point. That depends on the starting point chosen for initial iteration to which fixed point a network will converge.

Motionless points are called as **attractors**. The set of points (vectors) which are attracted to a certain attractor in the course of iterations of a network, is called as “attraction area” of this attractor. The set of motionless points of Hopfield’s network is its **memory**. In this case the network can work as associative memory. Those input vectors which get to the sphere of an attraction of a separate attractor, are connected (associated) with it.

For example, the attractor can be some desirable image. The area of an attraction can consist of noisy or incomplete versions of this image. There is a hope that images which vaguely remind a desirable image will be remembered by a network as associated with this image.

Fig. 2.1 Binary network of Hopfield



2.1.2 Binary Networks of Hopfield

In Fig. 2.1 the binary network of Hopfield is represented. Input and output vectors consist of “-1” and “+1” (instead of “-1”, “0” can be used). There is a symmetric weight matrix $W = \|w_{ij}\|$ consisting of integers with zeros (or “-1”) on a diagonal. The input vector X is multiplied by a weight matrix, using usual matrix and vector multiplication. The input vector X is fed to corresponding neurons and the output vector is determined. However, only 1 component of an output vector $Y = [y_j]$ is used on each iteration. This procedure is known as **“asynchronous correction”**. This component which can be chosen incidentally or by turn enters to a threshold element, whose output is -1, or +1). Corresponding component of an input vector is replaced with this value and, thus, forms an input vector for the following iteration. Process proceeds until input and output vectors become identical (that is, the motionless point will be reached) This algorithm is described below.

2.1.3 Description of the Algorithm of Asynchronous Correction

At the first moment a key k is closed (see Fig. 2.1) so an input vector x is fed with weight $\|w_{ij}\|$ to input neurons and the total signal at the input of j th neuron $S_j(x)$ is defined. Further, the key k is disconnected and outputs of neurons are fed to their inputs. The following operations are to be made:

- Calculate components of an output vector $y_j, j = 1, 2, \dots, n$, using the formula

$$y_j = T\left(\sum_{i=1}^n w_{ij}x_i\right) \quad (2.1)$$

where

$$T(x) = \begin{cases} -1, & \text{if } x < 0 \\ 1, & \text{if } x > 0 \\ y \text{ is not changed,} & \text{if } x = 0 \end{cases}$$

- To execute asynchronous correction, i.e. [2–4]:

Step 1: start with an input vector (x_1, x_2, \dots, x_n) .

Step 2: find y_j according to the formula (2.1).

Step 3: replace (x_1, x_2, \dots, x_n) with $(y_1, x_1, x_2, \dots, x_n) = Y$ and a feed Y back to input X .

Step 4: repeat process to find y_2, y_3 , etc. and replace the corresponding inputs.

Repeat steps 2–3 until the vector: $Y = (y_1, y_2, \dots, y_n)$ ceases to change. It was proved each such step reduces the value of communications energy E if at least one of outputs has changed:

$$E' = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j, \quad (2.2)$$

so convergence to a motionless point (attractor) is provided.

Asynchronous correction and zeros on a diagonal of a matrix W guarantee that power function (2.2) will decrease with each iteration [2, 5]. Asynchronous correction is especially essential to ensuring convergence to a motionless point. If we allow whole vector to be corrected on each iteration, it is possible to receive a network with periodic cycles as terminal states of an attractor, but not with motionless points.

2.1.4 Patterns of Behavior of Hopfield's Network

Weight matrix distinguishes behavior of one Hopfield's network from another so there is a question: "How to define this weight matrix?"

The answer is it should be given a set of certain weight vectors which are called **etalons**. There is a hope that these etalons will be the fixed points of a resultant Hopfield's network, though it is not always so. In order to ensure these etalons to be attractors, the weight matrix $W = \|w_{ij}\|$ should be calculated so [5]:

$$w_{ij} = \sum_{k=1}^N (x_{ki} - 1)(x_{kj} - 1), \text{ if } i \neq j \quad (2.3)$$

$$0, \text{ if } i = j,$$

where N is a number of the etalons, X_k is the k th etalon.

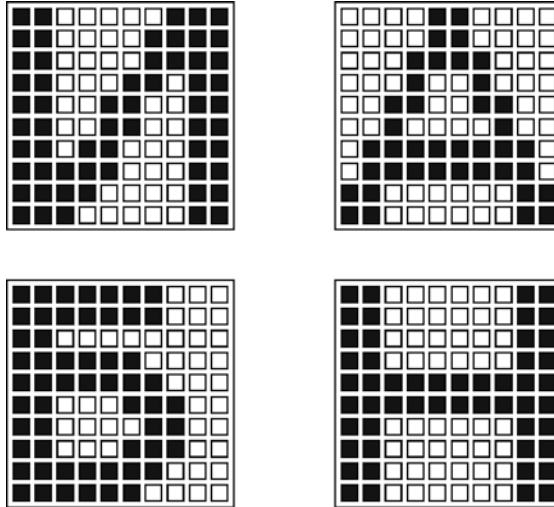
If etalon vectors form a set of orthogonal vectors, it is possible to guarantee that if the weight matrix is determined as shown above in formula (2.3), each etalon vector will be a motionless point. However, generally in order that etalons become motionless points, orthogonality isn't obligatory.

It should be noted that Hopfield network weights aren't trained like BP or RBF networks but are calculated in accordance with formula (2.3).

2.1.5 Application of Hopfield's Network

Hopfield's network can be used in particular for images recognition. But the number of the recognizable images isn't too great owing to limitation of memory of Hopfield's networks. Some results of its work are presented below in the experiment of adjusting network to recognize letters of the Russian alphabet [2].

Initial images are:



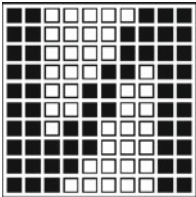
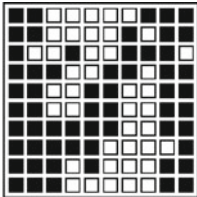
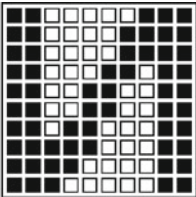
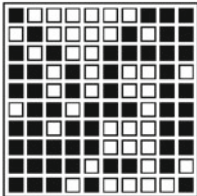
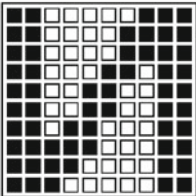
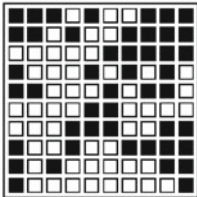
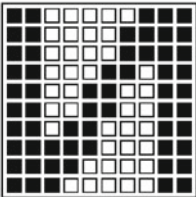
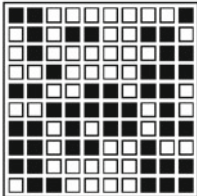
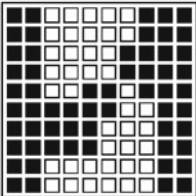
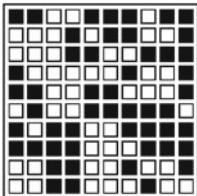
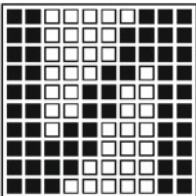
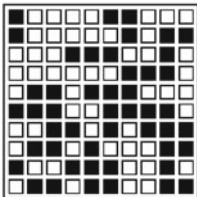
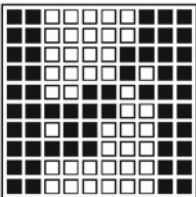
Research was conducted thus: consistently increasing noise level each of 4 images, they are fed to inputs of Hopfield's network. Results of network functioning are given in Table 2.1.

Thus, Hopfield's neural network perfectly copes with a problem of recognition of images(pattern) in experiments with distortion of 0–40 %. In this range all images(pattern) were recognized without mistakes (sometimes there are insignificant distortions for 40 % level of noise).

At 45–60 % level of noise images(patterns) are recognized unstably, often there is “entangling” and at the neural network output appears absolutely other image (pattern) or its negative.

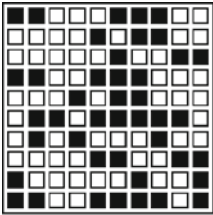
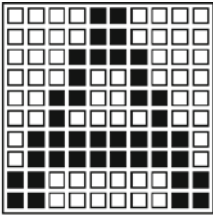
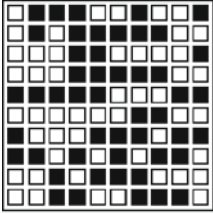
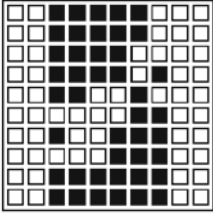
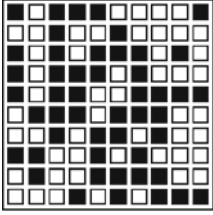
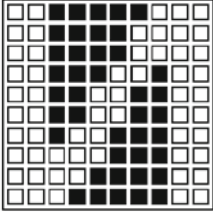
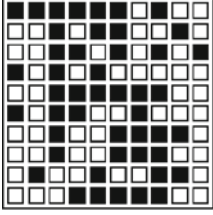
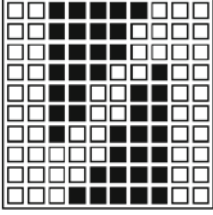
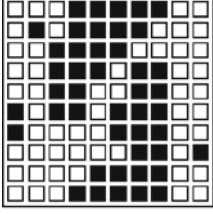
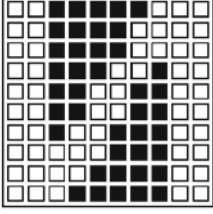
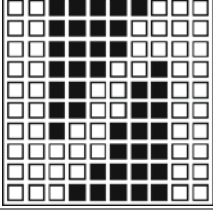
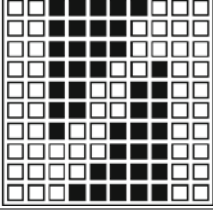
Beginning from 60 % noise level at the system output the negative of the tested image(pattern) appears which sometimes is partially distorted (starts appearing at 60–70 %).

Table 2.1 Experiments with Hopfield network

The tested image	Percent of image dis-	View of the distorted image	Result of recognition
tortion			
	10%		
	20%		
	30%		
	35%		
	40%		
	45%		

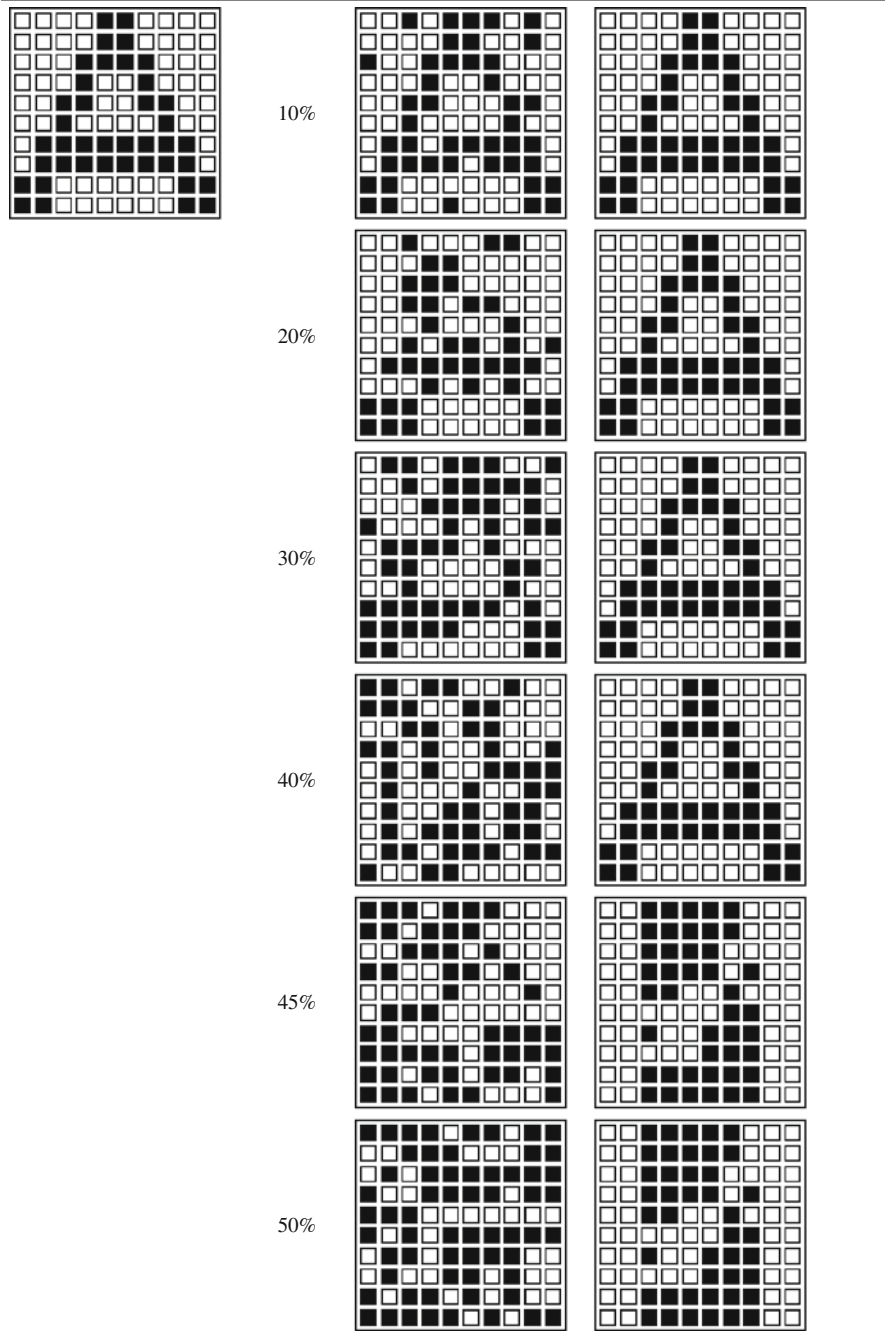
(continued)

Table 2.1 (continued)

50%		
60%		
70%		
80%		
90%		
100%		

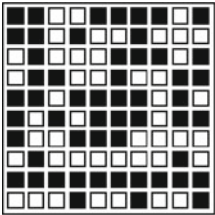
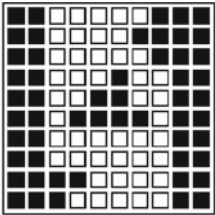
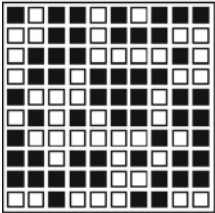
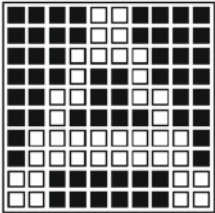
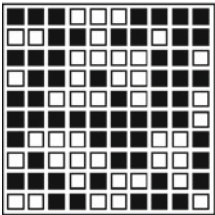
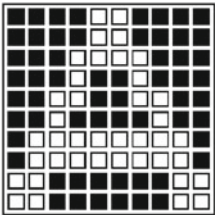
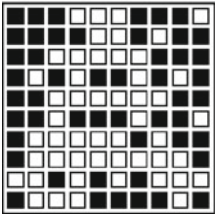
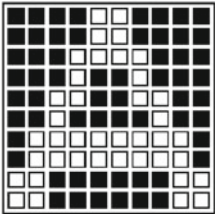
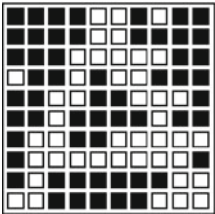
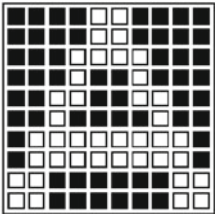
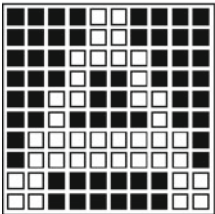
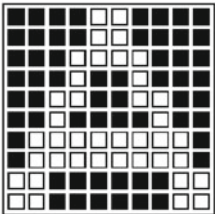
(continued)

Table 2.1 (continued)



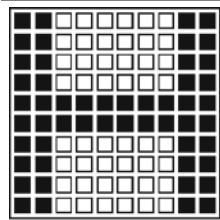
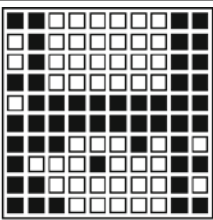
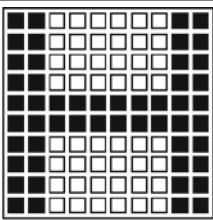
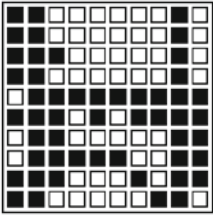
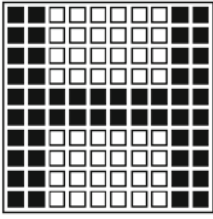
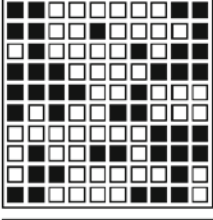
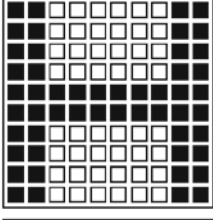
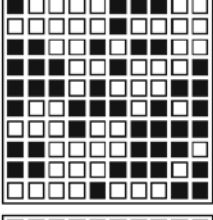
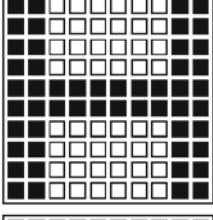
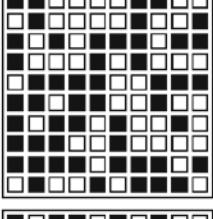
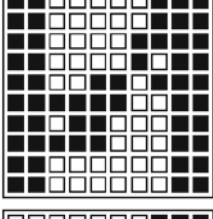
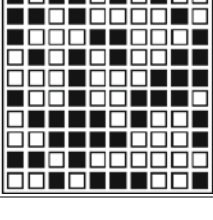
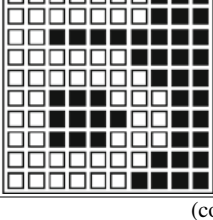


(continued)

Table 2.1 (continued)

60%		
65%		
70%		
80%		
90%		
100%		

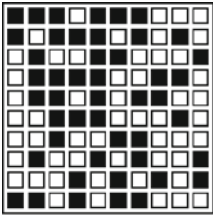
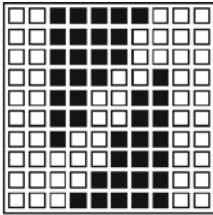
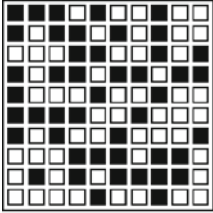
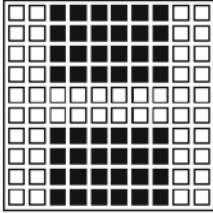
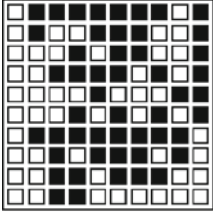
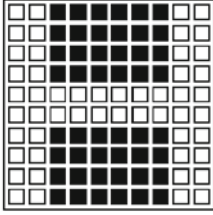
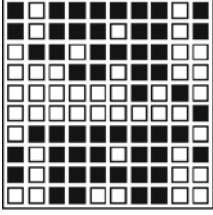
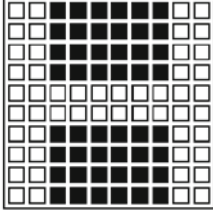
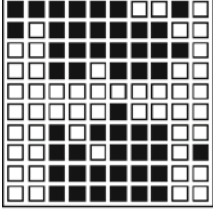
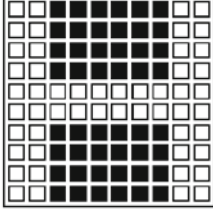
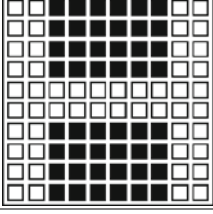
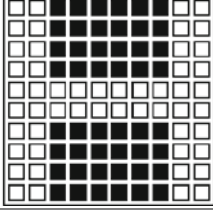
(continued)

Table 2.1 (continued)

			
	10%		
	20%		
	30%		
	40%		
	45%		
	50%		

(continued)

Table 2.1 (continued)

55%		
60%		
70%		
80%		
90%		
100%		

2.1.6 The Effect of “Cross-Associations”

Let's complicate a task and train our neural network with one more pattern:

The letter “Π” is very similar to letters “I” and “H” which already exist in the memory (Fig. 2.2). Now Hopfield's neural network can't distinguish any of these letters even in an undistorted state. Instead of correctly recognizable letter it displays the following image (for distortion of an pattern from 0 to 50 %):

It looks like to each of letters “I”, “H”, “Π” but isn't the correct interpretation of any of them (Fig. 2.3).

From 50 to 60 % noise level at the output of neural network at first appears an image presented above (Fig. 2.3) in slightly distorted form, and then its negative.

Since 65 % of noise level, at the neural network output steadily there is an image negative to shown in Fig. 2.3.

The described behavior of a neural network is known as effect of “cross associations” [2]. Thus “A” and “B” letters are recognized unmistakably at noise level up to 40 %.

At 45–65 % noise level at the network output appear, their slightly noisy interpretations, the image similar to a negative of a letter “B” (but very distorted), or a negative of the tested image (pattern). At distortion level of 70 % and more neural network steadily displays its negative of the tested image.

The experimental investigations had revealed the following shortcomings of Hopfield's neural network:

1. existence of cross-associations when some images(patterns) are similar to each other (like the experiments with letter Π);
2. due to storing capacity restrictions the number of the remembered attractors (patterns) is only $(0, 15-0, 2) n$, where n is dimension of a weight matrix W .

These circumstances significantly limit possibilities of practical use of Hopfield's network.

Fig. 2.2 New test symbol

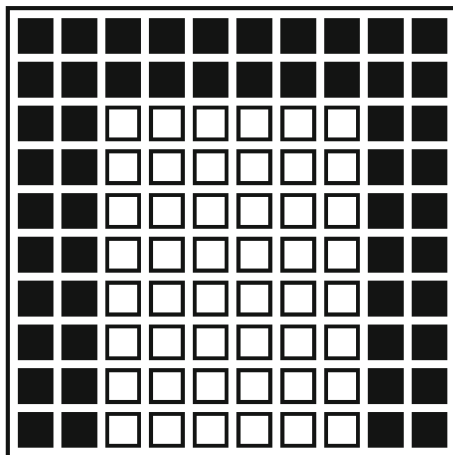
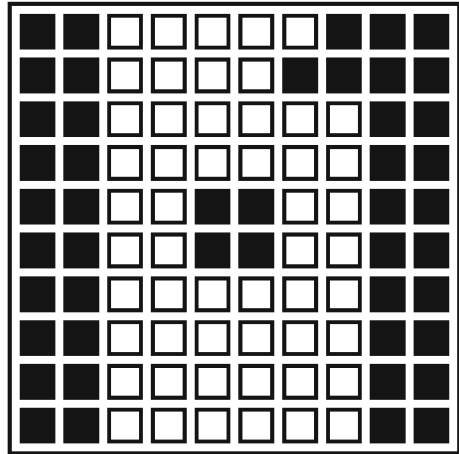


Fig. 2.3 Output symbol for tested symbols И, Н, П



2.2 Neural Network of Hamming. Architecture and Algorithm of Work

When there is no need that the network would display an etalon pattern in an explicit form, that is it is enough to define, say, a class of a pattern, associative memory is realized successfully by Hamming's network. This network is characterized, in comparison with Hopfield's network, by smaller costs of memory and volume of calculations that becomes obvious of its structure and work (Fig. 2.4) [2, 6].

The network consists of two layers. The first and second layers have m neurons, where m is a number of patterns. Neurons of the first layer have n synapses connected to the network inputs (forming a fictitious zero layer). Neurons of the second layer are connected among themselves by synaptic connections. The only synopsis with positive feedback for each neuron is connected to his axon.

The idea of network functioning consists in finding of Hamming distance from the tested pattern to all patterns represented by their weights. The number of different bits in two binary vectors is called as Hamming distance. The network has to choose a pattern with the minimum of Hamming distance to an unknown input signal therefore the only one of a network outputs corresponding to this pattern will be made active.

At an initialization stage the following values are assigned to weight coefficients of the first layer and a threshold of activation function:

$$w = \frac{x_i^k}{2}, i = \overline{1, n}; k = \overline{1, m} \quad (2.5)$$

$$T_k = \frac{n}{2}; \overline{1, m} \quad (2.6)$$

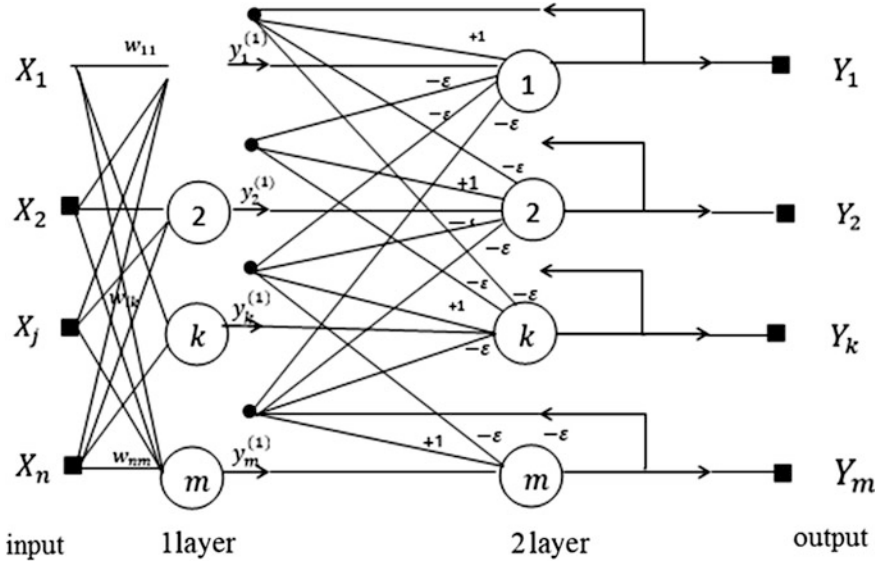


Fig. 2.4 Architecture of Hamming network

Here x_i^k is the i th an element of the k th pattern.

Weight coefficients of the braking synapses in the second layer are set equal to some value $-\varepsilon$, where $0 < \varepsilon < \frac{1}{m}$, where m is a number of classes.

The neuron synapse connected with its own axon has a weight (+1).

2.2.1 Algorithm of a Hamming Network Functioning

1. Enter the unknown vector $X = \{x_i : i = \overline{1, n}\}$ to a network input and determine outputs of the first layer neurons (the top index in brackets in formula (2.7) specifies number of a layer):

$$y_j^{(1)} = f(s_j^{(1)}) = f\left(\sum_{i=1}^n w_{ij}x_i + T_j\right), j = \overline{1, m} \quad (2.7)$$

After that initialize states of axons of the second layer with received values:

$$y_j^{(2)} = y_j^{(1)} \quad j = \overline{1, m} \quad (2.8)$$

2. Calculate new states of the second layer neurons

$$s_j^{(2)}(p+1) = y_j(p) - \varepsilon \sum_{k=1}^m y_k^{(2)}(p), \quad j = \overline{1, m} \quad (2.9)$$

And values of their axons:

$$y_j^{(2)}(p+1) = f[s_j^{(2)}(p+1)], \quad j = \overline{1, m} \quad (2.10)$$

Activation function f has a threshold, thus the size of a threshold should be rather big so that any possible values arguments won't lead to saturation.

3. Check, whether output of the second layer neurons has changed since the last iteration. If yes, then pass to a step 2 of the next iteration, otherwise—the end.

From the description of algorithm it is evident that the role of the first layer is very conditional, having used once on a step 1 value of its weight coefficients, the network doesn't come back to it any more, so that the first layer may in general be excluded from a network and replaced with a matrix of weight coefficients.

Note advantages of neural network of Hamming:

- small costs of memory;
- the network works quickly;
- algorithm of work is extremely simple;
- capacity of a network doesn't depend on dimension of an input signal (as in Hopfield's network) and equals exactly to a number of neurons.

2.2.2 *Experimental Studies of Hopfield's and Hamming's Networks*

Comparative experimental researches of Hopfield's and Hamming's neural networks in a problem of symbols recognition were carried out. For learning of a network input sample of symbols (1, 7, e, q, p) was used. Then generated noisy patterns from this sample were entered and their recognition was performed. Level of noise changed from 0 to 50 %. Results of recognition of the specified symbols are presented in Fig. 2.5. On the screen 4 images (patterns) are presented (from left to right, from top to down): the initial image—a etalon, the noisy image, result of Hamming network, result of Hopfield's network (Figs. 2.6, 2.7, 2.8, 2.9).

2.2.3 *Analysis of Results*

Results of experiments with Hopfield's and Hamming's networks are presented in the Table 2.2. A corresponding table element is a result of recognition by a network

Symbol 1



Fig. 2.5 Recognition of the symbol 1

Symbol 7



Fig. 2.6 Recognition of the symbol 7

Symbol E

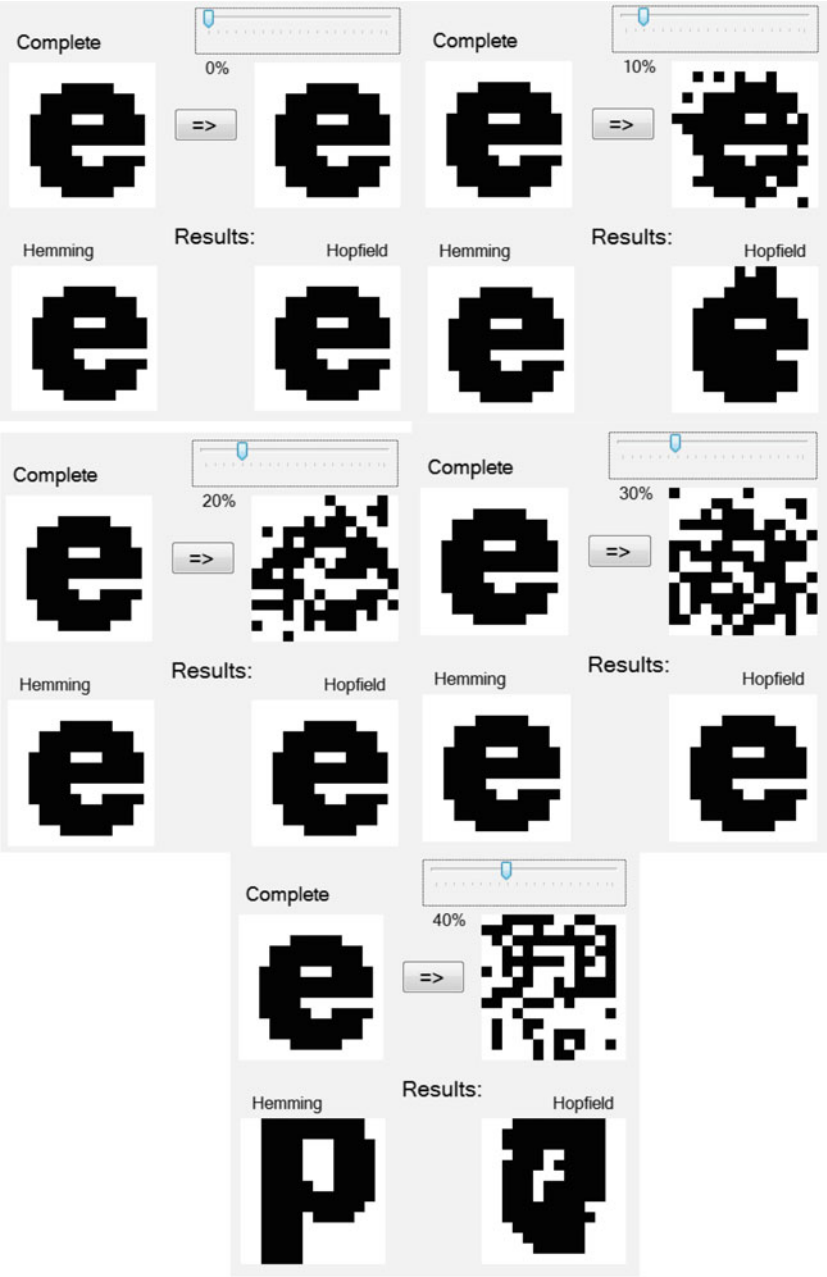


Fig. 2.7 Recognition of the symbol e

Symbol Q

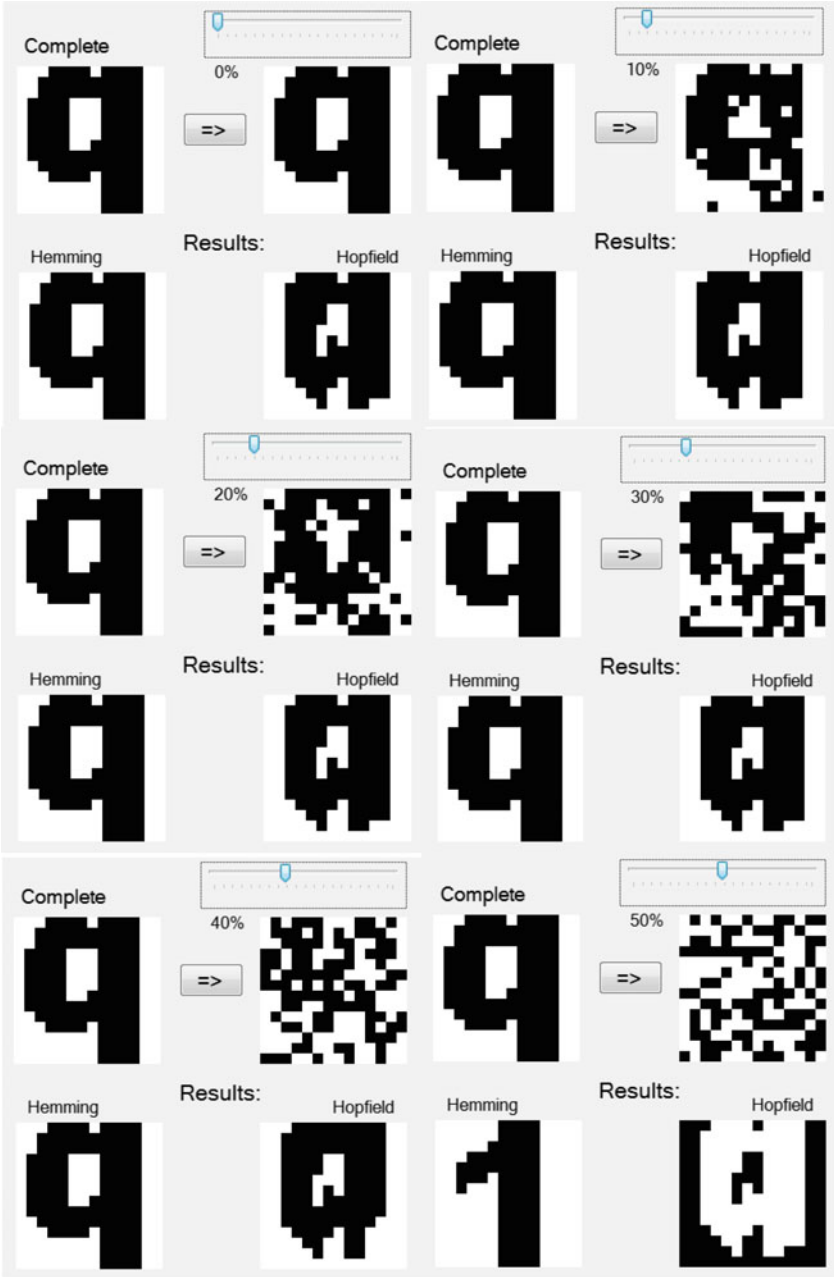


Fig. 2.8 Recognition of the symbol q

Symbol P



Fig. 2.9 Recognition of a p symbol

Table 2.2 Comparative results of experiments with Hopfield and Hamming networks

Recognition (Hamm, Hop)	0 %	10 %	20 %	30 %	40 %	50 %
1	(1; 1)	(1; 0.5)	(1; 1)	(1; 0.5)	(1, 0.5)	(0; 0)
7	(1; 1)	(1; 0.5)	(1; 0.5)	(1; 1)	(1, 0.5)	(0; 0)
E	(1; 1)	(1; 0.5)	(1; 1)	(1; 1)	(0; 0)	(0; 0)
Q	(1; 0)	(1; 0)	(1; 0)	(1; 0)	(1; 0)	(0; 0)
P	(1; 1)	(1; 1)	(1; 0)	(1; 0)	(1; 0)	(0; 0)

(Hamming; Hopfield) of a symbol at the specified noise level, and namely: 0—it isn't recognized; 0.5—it is recognized with defects; 1—it is recognized correctly.

Hamming's network in general performed well (except for "e" symbol) and recognized correctly up to the level of noise 40 %), while Hopfield's network results of recognition are much worse, at recognition of symbols with a similar elements (e, p, q) there were difficulties—recognition level was less than 30 %, it is the effect of cross associations.

2.3 Self-organizing Neural Networks. Algorithms of Kohonen Learning

2.3.1 Learning on the Basis of Coincidence. Law of Hebb Learning

In 1949 the Canadian psychologist D. Hebb published the book "Organization of Behaviour" in which he postulated the plausible mechanism of learning at the cellular level in a brain [2, 7].

The main idea of Hebb consisted therein when the input signal of neuron arriving through synaptic communications causes activation operation of neuron, efficiency of such input in terms of its ability to cause operation of neuron in the future has to increase.

Hebb assumed that change of efficiency has to happen in a synapse which transmits this signal to an neuron input. The latest researches confirmed this guess of Hebb. Though recently other mechanisms of biological learning at the cellular level were detected, but in recognition of merits of Hebb this law of learning was called in his honor [7].

The law of Hebb learning belongs to a class of laws of learning by competition.

Linear Associative Elements

In Fig. 2.10 the architecture of the neural network (NN) consisting of m neurons which are called as "linear associators" is presented.

The input vector in the linear associator is a vector, $X = \{x_i\}$, $i = \overline{1, n}$, which is taken out of space R_n according to some distribution $\rho(x)$.

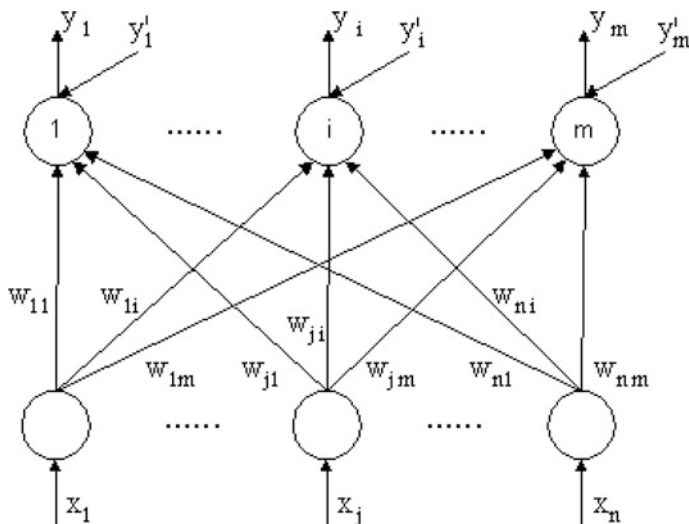


Fig. 2.10 Network with linear associators—neurons

The output vector is obtained from input X by the following formula:

$$Y = WX \quad (2.11)$$

where $W = \|w_{ij}\|$ is a weight matrix $n \times m$; $W = (W_1, W_2, \dots, W_m)$, W_j is W -matrix column, $W_j = (W_{1j}, W_{2j}, \dots, W_{nj})^T$ is a weight vector.

We will designate through $Y' = \{y_j\}$ a desirable output. The main idea of a linear associative neural network consists that the network has to learn on pairs input-output:

$$(X_1, Y_1), (X_2, Y_2), \dots, (X_L, Y_L)$$

When to an neural network input the signal X_k is given, desirable output Y' has to be equal Y_k . If on an network input the vector $X_k + \varepsilon$ is given (where ε —rather small), the output has to be equal $Y_k + \varepsilon$ (i.e. at the output have to receive the vector close to Y_k).

The law of Hebb's learning is as follows [2, 7]:

$$w_{ij}^{new} = w_{ij}^{old} + y_{kj}x_{ki}, \quad (2.12)$$

Where x_{ki} —is the i th vector component X_k ; y_{kj} — j th vector component Y_k .

In a vector form the expression (2.12) is written so:

$$W^{new} = W^{old} + X_k Y_k^T = W^{old} + Y_k X_k^T, \quad (2.13)$$

To realize this law in the course of learning the corresponding components $Y_k = [y_{kj}]$, which are shown by dashed lines (arrows) in Fig. 2.10 are entered.

It is supposed that before learning, all $w_{ij}^{(0)} = 0$. Then as a result of display of the learning sample $(X_1, Y_1), \dots, (X_L, Y_L)$ the final state of a matrix of W is defined so:

$$W = Y_1 X_1^T + Y_2 X_2^T + \dots + Y_L X_L^T. \quad (2.14)$$

This Eq. (2.14) is called “a formula of the sum of external product” for W . This name comes from that fact that $Y_k X_k^T$ —is an external product.

The reformulation of Hebb law in the form of the sum of external product allows to conduct additional researches of opportunities of this law to provide associations of pairs of vectors (X_k, Y_k) .

The first conclusion consists that if vectors $\{X_1, X_2, \dots, X_L\}$ are ortogonal and have unit length, i.e. are orthonormalized, then

$$Y_k = W X_k, \quad (2.15)$$

In other words, the linear associative neural network will produce desirable transformation “input-output”.

This is the consequence of an orthonormalization property:

$$X_i^T X_j = \delta_{ij} = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \quad (2.16)$$

Then

$$W X_k = \sum_{r=1}^L Y_r X_r^T X_k = Y_k. \quad (2.17)$$

But the problem consists that an orthonormalization condition is very rigid (first of all it is necessary, that $L \leq n$).

Further we are restricted by the requirement that $\|X_i\| = 1$. It would be much more useful if it was succeeded to lift this restriction. These goal can be achieved, but not in a linear associative neural network. Here, if vector $s X_k$ aren't orthonormalized, then a reproduction error Y_k appears at the output:

$$W X_k = \sum_{r=1}^L Y_r X_r^T X_k = Y_k + \sum_{r \neq k} Y_r X_r^T X_k = Y_k + \eta. \quad (2.18)$$

It is desirable to achieve that η be minimum. To provide $\eta = \min$ or $\eta = 0$, it is necessary to pass to a nonlinear associative network with nonlinear elements.

2.3.2 Competitive Learning

Competitive learning is used in problems of self-learning, when there is no classification of the teacher.

The laws of learning relating to category competitive, possess that property that there arises a competitive process between some or all processing elements of a neural network. Those elements which appear winners of competition, get the right to change their weights, while the rest of the weights don't change (or change by another rule).

Competitive learning is known as “Kohonen’s learning”. Kohonen’s learning significantly differs from Hebb learning and BP algorithm by therein the principle of self-organization is used (as opposed to the principle of controlled learning with the teacher).

The competitive law of learning has long and remarkable history [2]. In the late sixties—the beginning of the 70th Stephen Grossberg suggested the whole set of competitive learning schemes for neural networks. Another researcher who dealt with problems of competitive learning was van der Malsburg. The learning law of van der Malsburg was based on idea that the sum of the weights of one input element connected with various processing neurons has to remain a constant in the course of learning i.e. if one of weights (or some) increases, the others have to decrease.

After considerable researches and studying works of Grossberg, van der Malsburg and others Toivo Kohonen came to the conclusion that the main goal of competitive learning has to consist in designing of a set of vectors which form a set of equiprobable representatives of some fixed function of distribution density $\rho(x)$ of input vectors. And though learning laws of this type were independently received by many researchers, T. Kohonen was the first who paid attention to a question of equiprobability. Exactly thanks to this idea and the world distribution of T. Kohonen book “Self-organization and associative memory” [8] his name began to associate with this law of learning.

2.3.3 Kohonen’s Learning Law

The basic structure of a layer of Kohonen neurons is given in Fig. 2.11. The layer consists of N processing elements, each of which receives n input signals x_1, x_2, \dots, x_n from a lower layer which is the direct transmitter of signals. To an input x_i and communication (i, j) we will attribute weight w_{ij} .

Each processing element of a layer of Kohonen counts the input intensity I_j in compliance with a formula [2, 8]:

$$I_j = D(W_j, X), \quad (2.19)$$

where $W_j = (w_{1j}, w_{2j}, \dots, w_{nj})^T$ and $X = (x_1, x_2, \dots, x_n)$; $D(W_j, X)$ —some measure (metrics) of distance between W_j and X .

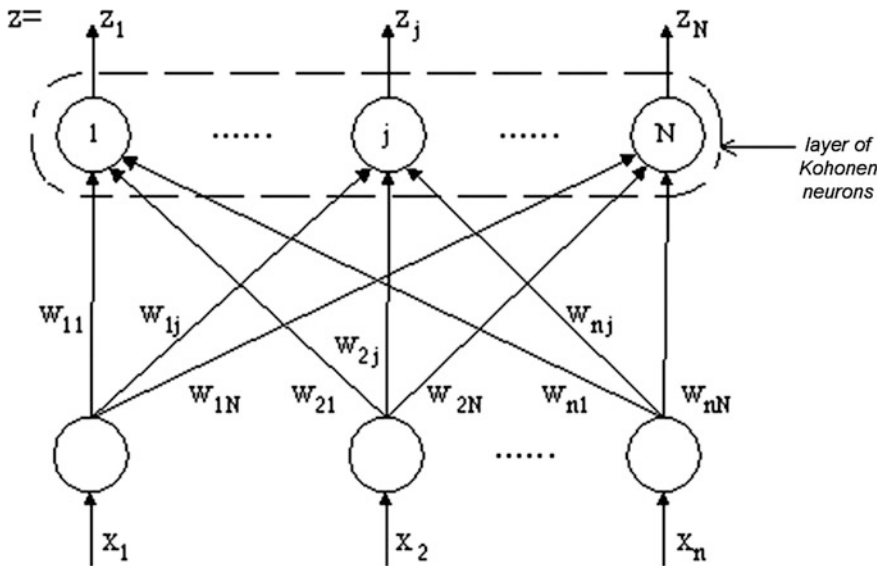


Fig. 2.11 Architecture of Kohonen network

We will define two most general forms of function $D(W_j, X)$:

1. Euclidean distance: $d(W, X) = \|W - X\|$;
2. Spherical arc distance:

$$\Delta(W, X) = 1 - W^T X = 1 - \cos \theta \quad (2.20)$$

where $W^T X$ —the scalar product, and is supposed that $\|W\| = \|X\| = 1$.

In this statement, unless otherwise stated, we'll use Euclidean distance $d(W, X)$. At implementation of the Kohonen law as soon as each processing element (neuron) counted the function I_j , a competition between them takes place, whose purpose is to find an element with the smallest value I_j (i.e. $I_{j_{\min}}$). As soon as the winner of such competition is found, his output z is put equal to 1. Output signals of all other elements remain equal to 0.

At this moment a learning by Kohonen takes place.

The learning data for Kohonen's layer assumed to consist of sequence of input vectors $\{X\}$, which are taken randomly with the fixed density of probabilities distribution $\rho(x)$. As soon as next vector X it is entered into a network, the processing Kohonen's neurons start competing to find the winner for whom $\min_j d(X, W_j)$ is reached. Then for the winner neuron j^* output is established $z_{j^*} = 1$, and for all others $z_j = 0$, $j \neq j^*$.

At this moment a change of weights according to Kohonen learning law is performed:

$$W_j^{new} = W_j^{old} + \alpha(X - W_j^{old})z_j, \quad (2.21)$$

where $0 < \alpha < 1$.

This law can be written in the following form:

$$W_j^{new} = \begin{cases} (1 - \alpha)W_j^{old} + \alpha X, & \text{for winner } j = j^* \\ W_j^{old}, & \forall j \neq j^* \end{cases} \quad (2.22)$$

It is evident that at such learning law the weight vector W_j moves to an input vector X . At the beginning of learning process $\alpha = 1$ and then in process of learning it monotonously decreases up to the value $\alpha = 0, 1$.

This algorithm realizes the principle “The winner takes all” therefore in foreign literature it is called WTA. Further it should be noted similarity of learning by Kohonen and statistical process of finding of “k-means”.

K-means for the fixed set of vectors $\{X_1, X_2, \dots, X_L\}$, which are chosen randomly from some population with the fixed density of probabilities distribution $\rho(x)$, make a set of k vectors $W = (W_1, W_2, \dots, W_k)$ such that the following functional is minimized:

$$\min_{\{w_i\}} \sum_{i=1}^L D^2(X_i, W(X_i)) \quad (2.23)$$

Where $W(X_i)$ is a vector W , closest to X_i .

In summary, it is necessary to emphasize that the learning by Kohonen’s algorithm generally doesn’t generate a set of equiprobable weight vectors, that is a set of such vectors that X which is chosen randomly, according to density of probabilities distribution ρ will have equal probability to be the closest to each of weight vectors W_j .

2.3.4 Modified Competitive Learning Algorithms

As it was already noted above, we seek to get vectors W_j , which would be approximately equally probable in the sense of being the closest to the vectors of X taken from \mathfrak{R}^n with some density of probability distribution. In other words, for any vector of X taken from \mathfrak{R}^n with probability $\rho(x)$, it is desirable that the probability of that X will appear to be the closest to W_i , has to be approximately equal to $\frac{1}{N}$ for all $i = \overline{1, N}$.

There are some approaches for the solution of the problems arising at implementation of a basic learning law of Kohonen [2, 9].

1. The first approach is called as Radial Sprouting. It is the best for Euclidean metrics and metrics (distances) similar to it. All weight vectors $W_i = [w_{ij}]$ are originally set equal to $w_{ij}(0) = 0$. All input vectors X at first are multiplied by some small positive scalar β . Process begins with β , close to 0. It provides proximity of input vectors of X to vectors W_j . In process development β slowly increases, until reaches the value $\beta = 1$. As soon as it occurs, weight vectors “are pushed out” from the initial values and follow input vectors. This scheme works quite well, but usually some weight vectors will lag behind process and as a result will be not involved in the competition process that slows down learning process.
2. Other approach (“noise addition”) consists in adding randomly distributed noise to data vectors X that facilitates effect of achievement $\rho(X) > 0$, in all area Ω_x . Level of noise is chosen at first rather big so that noise vector be much greater, than a data vector X . But in the process of learning noise level gradually decreases. This approach works correctly, but it appears even more slowly, than approach of “Radial Sprouting”. Therefore approaches of “Radial Sprouting” and “noise addition” solve a problem of presentation of badly representable laws with small probability of distribution in some area, but they don’t solve a problem of equiprobable positioning of vectors W_j .

In general, the basic Kohonen’s learning law will bring to a surplus at placement of vectors W_j in those areas where probabilities distribution density $\rho(X)$, is large, and to shortage of vectors W_j in areas where density of probabilities distribution $\rho(X)$ is small.

3. The third approach which was offered by Duane Desieno, is to build-in “consciousness” (or memory) in each element k to carry out monitoring (control) of history of successful results (victories) of each neuron. If the processing Kohonen element wins competition significantly more often than $\frac{1}{N}$ times (time), then his “consciousness” excludes this element from competition for some time, thereby giving the chance to elements from the oversaturated area to move to the next non-saturated areas. Such approach often works very well and is able to generate good set of equiprobable weight vectors.

The main idea of the consciousness mechanism is a tracking of a share of time during which the processing element j wins competition. This value can be calculated locally by each processing element by formula:

$$f_j(t+1) = f_j(t) + \beta (z_j - f_j(t)) \quad (2.24)$$

When competition is finished and the current value z_j (0 or 1) is defined, the constant β takes a small positive value (typical value $\beta = 10^{-4} = 0,0001$) and the share f_j is calculated. Right after it the current shifts value b_j is defined

$$b_j = \gamma \left(\frac{1}{N} - f_j \right) \quad (2.25)$$

where γ —positive constant ($\gamma \approx 10$).

Further the correction of weights is carried out. However, unlike a usual situation in which weight are adjusted only for one processing element—winner with $z_i = 1$, here separate competition is being held for finding of the processing element which has the smallest value of

$$D(W_j, X) - b_j \quad (2.26)$$

The winner element corrects further the of weight according to the usual law of Kohonen's learning.

The role of the shift b_j is as follows. For often winning elements j the value $f_j > \frac{1}{N}$ and $b_j < 0$ therefore for them value $D(W_j, X) - b_j$ increases in comparison with $D(W_j, X)$ for seldom winning elements $f_j \ll \frac{1}{N}$, $b_j > 0$ and $D(W_j, X) - b_j$ decreases that increases their chances to win competition. Such algorithm realizes the consciousness mechanism in work of self-organizing neuron networks and therefore it is called as CWTA (Conscience Winner Takes All) [9].

2.3.5 Development of Kohonen Algorithm

In 1982 T. Kohonen suggested to introduce into the basic rule of competitive learning information on an arrangement of neurons in an output layer [2, 8, 10]. For this purpose neurons of an output layer are ordered, forming a one-dimensional or two-dimensional lattice. The arrangement of neurons in such lattice is marked by a vector index $i = (i_1, i_2)$. Such ordering naturally enters distance between neurons $|i - j|$.

The modified rule of competitive of Kohonen's learning considers distance of neurons from winner neuron [2, 8, 9]:

$$W_j(t+1) = W_j(t) + \alpha (X - W_j) \Lambda(d(i, j^*)) \quad (2.27)$$

where Λ —function of the neighborhood. $\Lambda(d(i, j^*))$ is equal 1 for winner neuron with an index j^* , and gradually decreases in process of increase in distance d , for example, by function

$$\Lambda(d) = e^{-d^2/R^2} \quad (2.28)$$

Both rate of learning α , and radius of interaction R gradually decreases in the course of learning so at a final stage of learning we come back to the basic law of weights adaptation only of winner neurons $\alpha(t) = a_0 e^{-kt}$.

As we can see in this algorithm the principle 2 is realized: winner takes away not all but maximum (income) therefore in foreign literature it is called as WTM (Winner Takes MOST).

Learning by Kohonen modified algorithm WTM reminds a tension of an elastic grid of prototypes on a data file from the learning sample. In process of learning the elasticity of a network gradually decreases and weights changes also decrease except winner neuron.

As a result we receive not only quantization of input's, but also we order input information in the form of a one-dimensional or two-dimensional topographic map of Kohonen. On this grid each multidimensional vector has the coordinate, and the closer are coordinates of two vectors on the grid, the closer they are in the initial space.

2.3.6 Algorithm of Neural Gas

Acceleration of convergence of the modified Kohonen WTM'S algorithm and the best self-organization of a network gas molecules motion can be received with application of the method offered by M. Martinez, S. Berkovich and K. Shulten [9] and called by authors "algorithm of neural gas" owing to similarity of its dynamics. In this algorithm on each iteration all neurons are sorted depending on their distance to a vector X . After sorting neurons are marked in the sequence corresponding to increase in their remoteness:

$$d_0 < d_1 < d_2 < \dots < d_{n-1}, \quad (2.29)$$

where $d_i = \|X - W_{m(i)}\|$ designates remoteness from a vector X of the i th neuron taking as a result of sorting position m in the sequence begun with neuron winner to which remoteness d_0 is put into compliance. Value of the neighborhood function for i th neuron is determined by a formula [9]:

$$\Lambda(i, x) = \exp\left\{-\frac{m(i)}{\sigma(t)}\right\}, \quad (2.30)$$

where $m(i)$ defines the sequence received as a result of sorting of $(m(i) = 0, 1, 2, \dots, n-1)$, and $\sigma(t)$ is the parameter similar to R neighbourhood level in Kohonen WTM's algorithm decreasing eventually with t .

At $\sigma(t) = 0$ adaptation only of the winner neuron occurs and the algorithm turns into usual (basic) Kohonen's algorithm, and at $\sigma(t) \neq 0$ of adaptation are subject the weights of many neurons neighbors, and the level of change of scales depends on size $\Lambda(i, x)$.

For achievement of good results of self-organization, process of learning has to begin with rather great value of σ , but eventually its size decreases to zero. Change of $\sigma(t)$ can be linear or exponential. In work [9] it was offered to change value according to expression

$$\sigma(t) = \sigma_{\max} \left(\frac{\sigma_{\min}}{\sigma_{\max}} \right)^{\frac{t}{T_{\max}}}, \quad (2.31)$$

where $\sigma(t)$ —value on iteration of t ; σ_{\min} and σ_{\max} —the accepted minimum and maximum values of σ . Value T_{\max} defines the maximum number of iterations. The learning coefficient of the i th neuron $\alpha_i(t)$ can also change both linearly, and exponentially, and its exponential dependence is defined by expression

$$\alpha_i(t) = \alpha_i(0) \left(\frac{\alpha_{\min}}{\alpha_i(0)} \right)^{\frac{t}{T_{\max}}}, \quad (2.32)$$

where $\alpha_i(0)$ is an initial value α , and α_{\min} is a priori set minimum value corresponding to $t = T_{\max}$. In practice the best results of self-organization are reached at linear change of $\alpha(t)$ [9]. For reduction of calculations volume at realization of neural gas algorithm it is possible to use the simplification consisting that adaptations of weights happen only for the first k of neurons neighbors in the ordered sequence of $d_0 < d_1 < d_2 < \dots < d_k$.

Algorithm of neural gas together with Kohonen's algorithm (CWTA) with memory considering a share of victories of each neuron of f_j are the most effective tools of neurons self-organization in Kohonen's network.

Comparative Analysis of Algorithms of Self-organization

Above considered algorithms were compared at the solution of a problem of recovery of the two-dimensional learned data of difficult structure which are presented in Fig. 2.12 [9]. For recovery of data 2 sets of the neurons including 40 and 200 elements were used which after ordering positions of neurons will reflect data distribution. They have to locate in areas of the maximum concentration of data.

Results of self-organization of 40 neurons when using three algorithms are presented in Fig. 2.13 algorithm with memory (CWTA) (Fig. 2.13a), neural gas (Fig. 2.13b) and basic algorithm of Kohonen (Fig. 2.13c). For comparison in Fig. 2.15 similar pictures are obtained by Kohonen's network consisting of 200 neurons (Fig. 2.14).

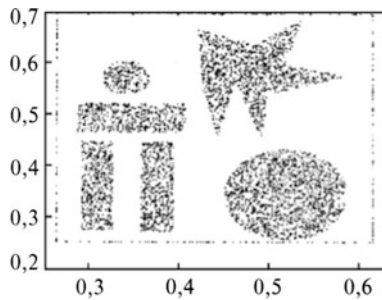


Fig. 2.12 Data structures to be simulated

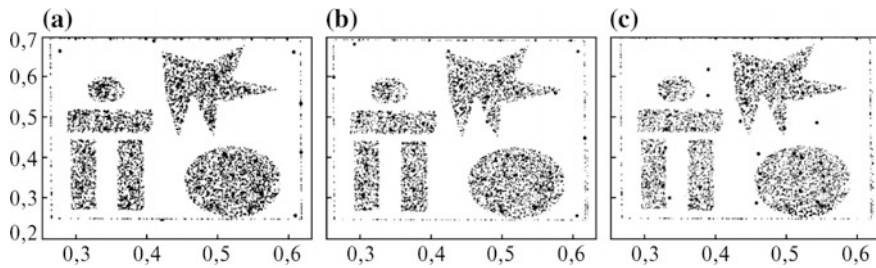


Fig. 2.13 Results of self-organization of different algorithms with 40 neurons

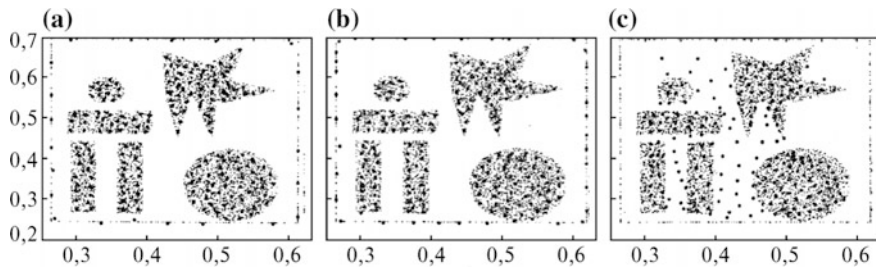


Fig. 2.14 Results of self-organization for 200 neurons

As follows from the given results, irrespective of number of neurons the best results of self-organization were received with use of algorithms of the self-organization with memory and neural gas.

For quantitative comparison of results it is possible to use criterion “a quantization error”:

$$E_q = \sum_{i=1}^n \|X_i - W_i^*\|, \quad (2.33)$$

where W_i^* is winner neuron weight at presentation of a vector X_i .

At 200 neurons the following criterion values were received [9]: $E_q = 0,007139$ for CWTA; $E_q = 0,007050$ for algorithm of neural gas and $E_q = 0,02539$ for basic Kohonen’s algorithm.

2.4 Application of Kohonen Neural Networks

Neural networks with self-organization are used in two main directions.

1. For automatic classification of objects.
2. For visual display of properties of multidimensional space (representation of multidimensional vectors of features).

In the first case the problem of automatic splitting a set of the objects represented by multidimensional vectors in some feature space on similarity—to difference of feature values is solved. Such task sometimes is called as a task of the cluster analysis and it is in detail considered in the 7th chapter.

In the second case the problem of visualization of properties of multidimensional feature vectors on the two-dimensional plane is solved.

So-called “Self-organizing Maps” (SOM) are used for this purpose [10].

For this in multidimensional space the spatial lattice is stretched in which nodes are processing neurons (a layer of Kohonen’s neurons). Further the next points to each of these neurons are defined. They define area of an attraction of this neuron. Average value of each feature of neurons in the attraction area is defined $-X_i$ cp.

Further nodes of a lattice are mapped on the plane in the form of squares or hexagons, the corresponding value of a feature for this neuron is painted in the certain color: from blue (the minimum value of a feature) to red (the maximum value). As a result we receive the self-organizing feature map similar to a geographical map. Such maps are built on all features and we receive a certain atlas.

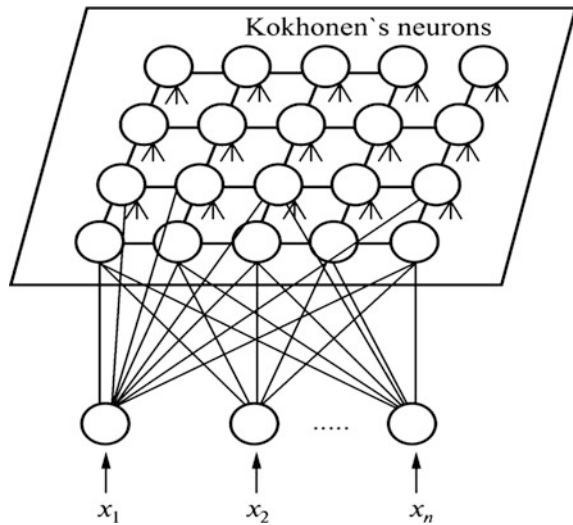
Self-organizing Maps (SOMs) represents a method of design of N-dimensional input space in discrete output space which makes an effective compression of input space in a set of the coded (weight) vectors. The output space usually represents itself a two-dimensional lattice. SOM uses a lattice for approximation of probability density function of an input space, thus keeping its structure i.e. if two vectors are close to each other in the input space, they have to be close and on the map as well. During self-organization process SOM carries out an effective clustering of input vectors, keeping structure of initial space [10].

Stochastic Algorithm of Learning

Learning of SOM is based on strategy of competitive learning. We will consider N-dimensional input vectors X_p where the index p designates one learning pattern. The first step of process of learning is definition of structure of the maps, usually two-dimensional lattice. Map is usually square, but may be rectangular. The number of elements (neurons of an output layer) on the map is less, than the number of the learning patterns (samples). The number of neurons has to be equal in an ideal to the number of the independent learning patterns. The structure of SOM is given in Fig. 2.15. Each neuron on the Map is connected with a N-dimensional weight vector which forms the center of one cluster. Big cluster groups are formed by grouping together of “similar” next neurons.

Initialization of Weight Vectors Can Be Carried Out in Various Ways

1. To each weight $W_{kj} = \{W_{kj1}, W_{kj2}, \dots, W_{kjN}\}$, where k is a number of rows and J is a number of columns, random values are attributed. Initial values are limited to the range of the corresponding input parameter (variable). Though it is simple to realize random initialization of weight vectors, such way of initialization gives a big variation of components into SOM that increases learning time.

Fig. 2.15 Structure of SOM

2. To weight vectors randomly chosen input patterns are attributed, i.e. $W_{kj} = X_p$, where p is a pattern index. Such approach can result in premature convergence until weight aren't disturbed with small casual values.
3. To find the main components of vectors of input space and to initialize weight vectors with these values.
4. Other technology of initialization of scales consists in definition of a hyper cube of rather big size covering all learning patterns. The algorithm begins work with finding of four extreme points by definition of four extreme learning patterns. At first find two patterns with the greatest distance from each other in an Euclidean metrics. The third pattern is placed in the most remote point from these two patterns, and the fourth pattern—with the greatest distance from these three patterns according to Euclid. These four patterns form lattice corners on the Map (SOM: (1, 1); (1, J); (K, 1); (K, J). Weight vectors of other neurons are defined by interpolation of four chosen patterns as follows. The weight of boundary neurons are initialized so [10, 11]:

$$5. W_{1j} = \frac{w_{1J} - w_{11}}{J - 1} (j - 1) + W_{11},$$

$$6. W_{Kj} = \frac{w_{KJ} - w_{K1}}{J - 1} (j - 1) + W_{K1},$$

$$7. W_{kj} = \frac{w_{KJ} - w_{11}}{K - 1} (k - 1) + W_{11}, \quad (2.34)$$

$$8. W_{KJ} = \frac{w_{KJ} - w_{1J}}{K - 1} (k - 1) + W_{1J},$$

9. For all $j = 2, 3, \dots, J - 1$ and $k = 2, 3, \dots, K - 1$.

10. other weight vectors are initialized so:

$$W_{Kj} = \frac{w_{KJ} - w_{K1}}{J - 1}(j - 1) + w_{K1}. \quad (2.35)$$

The standard algorithm of learning for SOM is stochastic in which weight vectors adapt after display of each pattern of a network. For each neuron the related code vector (weight) adapts so:

$$W_{kj}(t+1) = W_{kj}(t) + h_{mn,kj}(t) [X_p - W_{kj}(t)], \quad (2.36)$$

where m, n —index of a line and column of winner neuron correspondingly. The winner neuron is defined, as usual, by calculation of Euclidean distance from each weight vector to an input vector and a choice of neuron, the closest to an input vector, i.e.

$$\|W_{mn} - X_p\|^2 = \min_{(k,j)} \{ \|W_{kj} - X_p\|^2 \}, \quad (2.37)$$

Function $h_{mn,kj}(t)$ in the Eq. (2.36) is considered as function of the neighborhood. Thus, only those neurons which are in the vicinity (in the neighborhood) of (m, n) winner neuron, change the weights. It is necessary for ensuring convergence, that $h_{mn,kj}(t) \rightarrow 0$ at $t \rightarrow \infty$.

Function of the neighborhood usually is function of distance between coordinates of the neurons presented on the map i.e.

$$h_{mn,kj}(t) = h(\|c_{mn} - c_{kj}\|^2, t), \quad (2.38)$$

where $c_{mn}, c_{kj} \in R$, and with increase in distance $\|c_{mn} - c_{kj}\|^2, h_{mn,kj}(t) \rightarrow 0$. The neighbourhood can be determined by a square or a hexagon. However are most often used a smooth Gaussian kernel:

$$h_{mn,kj}(t) = \alpha(t) \exp\left(-\frac{\|c_{mn} - c_{kj}\|^2}{2\sigma^2(t)}\right). \quad (2.39)$$

Here $\alpha(t)$ is the learning speed, and $\sigma(t)$ -kernel width. Both functions $\alpha(t)$ and $\sigma(t)$ are monotonously decreasing functions with increase in t .

Creation of SOM in Batch Mode

Stochastic learning algorithm of SOM is too slow owing to need of weights adaptation of all neurons after each display of a pattern. The version of SOM learning algorithm in batch mode was developed. The first package SOM learning algorithm was developed by Kohonen and is described below [11].

1. To initialize weight vectors by purpose of the first KJ of the learning patterns where KJ—total number of patterns on the map.

Until stop condition won't be satisfied,
 for each neuron of kj do
 make the list of all patterns of X_p which are the closest to a weight vector of this neuron;
 end.

2. For each weight vector of w_{kj} calculate new value of a weight vector as an average of the corresponding list of patterns.

Also the accelerated version of package SOM learning algorithm was developed. One of the design problems arising at creation of SOM—determination of the map sizes. Too many neurons can cause glut when each learning pattern is attributed to various neurons. Or alternately, final SOM can successfully create good clusters from similar patterns, but many neurons will be with zero or close to the zero frequency of use where the frequency of neuron means number of patterns for which the neuron became the winner. Too small number of neurons, on the other hand, leads to clusters with big intra cluster dispersion.

The Accelerated Package Algorithm of Creation of SOM

1. To initialize weight vectors of w_{kj} , using any way of initialization.
2. Yet until stop condition(s) won't be satisfied

for each neuron k_j do
 calculate an average value for all patterns for which this neuron was the winner;
 designate average as \bar{w}_{kj} ;
 end.

3. To adapt weight value for each neuron, using expression

$$W_{kj} = \frac{\sum_n \sum_m N_{nm} h_{nm,kj} \bar{W}_{nm}}{\sum_n \sum_m N_{nm} h_{nm,kj}}, \quad (2.40)$$

where indexes m, n are summarized according to all numbers of rows and columns; N_{nm} is the number of patterns for which the neuron appeared to be the winner, and $h_{nm,kj}$ —function of the neighborhood which specifies, whether neuron of (m, n) gets into area of the neighborhood of (k, j) neuron and in what degree;
 end.

The method of search close to optimum structure of SOM consists in beginning with small structure and to increase the Maps sizes when the increase in number of neurons is required. We will notice that development of the map takes place along with learning process. Consider one of algorithms of SOM structure development for the rectangular Map [11].

Algorithm of SOM Structure Development

To initialize weight vectors for small SOM,
 yet until the stop condition(s) won't be satisfied do;
 until a condition of the Map growth won't be true, do;
 to learn SOM for t displays of patterns, using any method of SOM learning;
 end.

If the condition of the Map growth is satisfied,

1. find (k, j) neuron with the greatest error of quantization (intercluster dispersion);
2. find the most distant direct neighbor of (m, n) in rows of the Map;
3. find the most remote neuron in Map columns;
4. insert a column between neurons of (k, j) and (r, s) , and a line between neurons of (k, j) and (m, n) (this step keeps rectangular structure of the Map);
5. for each neuron (a, b) in a new column initialize the corresponding vector of W_{ab} , using expression

$$W_{ab} = \gamma(W_{a,b-1} + W_{a,b+1}), \quad (2.41)$$

and for each neuron in a new row calculate

$$W_{ab} = \gamma(W_{a-1,b} + W_{a+1,b}) \quad (2.42)$$

where $\gamma \in (0, 1)$;

end.

6. To adjust the weights of the final structure of SOM, using additional learning iterations, until convergence will be reached.

The increase in the sizes of the map needs to be stopped when one of the following criteria is satisfied:

- the maximum size of the Map is reached;
- the greatest error of quantization for neuron will become less than threshold ε determined by the user;
- the error of the map quantization converged to a preset value.

Some aspects of this algorithm demand the explanation.

There are constants ε , γ and the maximum SOM size, and also various stop conditions. For parameter γ a good choice is $\gamma = 0.5$. The idea of an interpolation step consists in assigning a weight vector to new neuron so that it will take patterns from (k, j) neuron with the greatest error of quantization to reduce an error of this neuron. Value $\gamma < 0.5$ will locate neuron (a, b) closer to (k, j) that, perhaps, will lead to that more patterns will be taken by it at kj neuron, value $\gamma > 0.5$ will cause a boomerang effect.

The threshold of an quantization error ε is important to provide the sufficient size of the SOM Map. Small value ε leads to too big SOM size whereas too great value of ε can lead to increase in learning time to reach rather big size of structure.

It is easy to define the upper bound of the Map size, it is equal simply to the number of learning patterns p_T . However it, naturally, is undesirable. The maximum size of the Map can be presented as βp_T , where $\beta \in (0, 1)$. Optimum value β depends on the solved task, and it is necessary to take measures to provide not too small value β if the increase in the SOM sizes isn't provided.

Process of learning of SOM is too slow owing to a large number of iterations of scales correction. For reduction of computing complexity and acceleration of convergence of learning some mechanisms are offered. One of them is batch mode of designing of SOM. Two other mechanisms include control of functions of the neighborhood and learning speed.

If Gaussian neighborhood function is used, all neurons drop to the winner neuron neighborhood area, but with different degree. Therefore introducing a certain threshold θ , it is possible to limit number of neurons which will get to this area and by that to reduce computational costs of correction of their weights. Besides, the width of the neighborhood function σ can be changed dynamically during learning process. For example, it is possible to choose this function so:

$$\sigma(t) = \sigma(0)e^{-\frac{t}{\tau_1}}, \quad (2.43)$$

where $\tau_1 > 0$, is some constant; $\sigma(0)$ is the initial rather big variation. Similarly it is possible to use the learning speed $\alpha(t)$ decreasing with increase in time:

$$\alpha(t) = \alpha(0)e^{-\frac{t}{\tau_2}}, \quad (2.44)$$

where $\tau_2 > 0$, is some constant; $\alpha(0)$ is initial, rather big variation.

Clustering and Visualization. Application of SOM

Implementation of SOM learning process consists in a clustering (grouping) of similar patterns, keeping thus topology of input space. After learning the set of the learned weights is obtained without obvious borders between clusters.

The additional step for finding of borders between clusters is required.

One of the ways to define and visualize these borders between clusters is to calculate the unified matrix of distances (U-matrix [11]) which includes geometrical approximation of distribution of weight vectors on the Map. The U-matrix expresses for each neuron distance to weight vectors of the next neurons. Great values in a matrix of distances of U indicate location of borders between clusters.

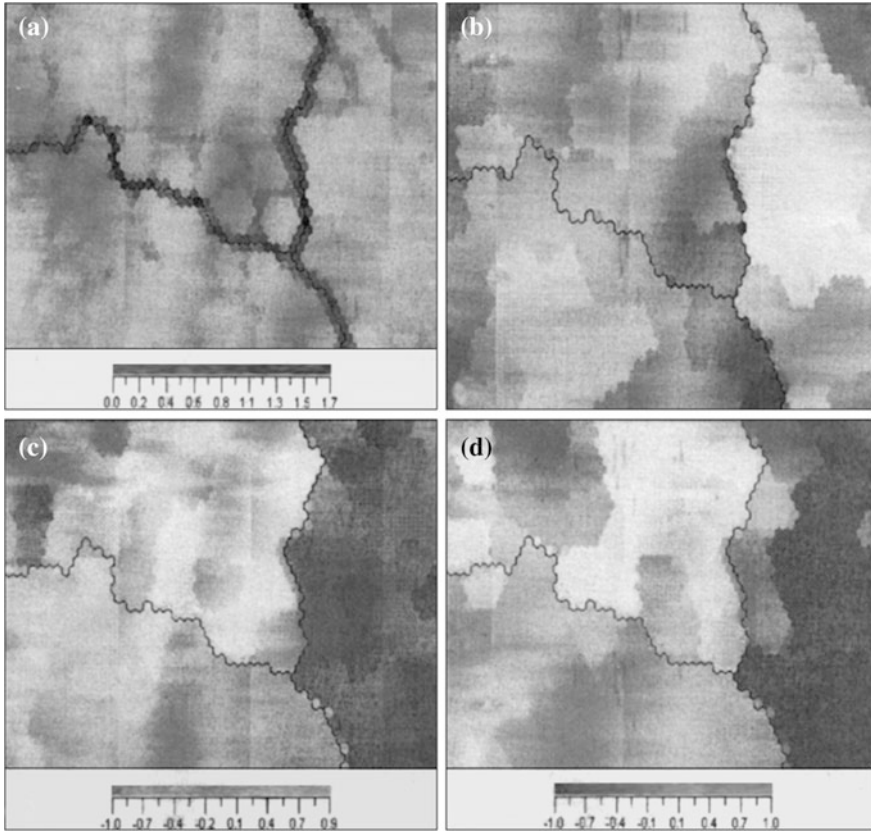


Fig. 2.16 Example of SOM for irises clustering problem

As an example consider a problem of a clustering of irises. Its statement and the description are given in Chap. 4. For example, in Fig. 2.16, the U-matrix for a problem of irises clustering with use of scaling of the SOM Map in shades of the grey is presented. Thus borders between clusters are marked in more dark color.

For the same problem in Fig. 2.16b clusters on a full map are visualized. Borders between them can usually be found by one of clustering methods, for example, Ward's method (see Chap. 7). The clustering by Ward's method uses approach "from down-up" in which each neuron originally forms an own cluster. On the subsequent iterations two next clusters merge in one until the optimum or certain number of clusters is designed. The end result of a clustering is the set of clusters with the minimum intra cluster dispersion and big inter-cluster dispersion.

For definition, what clusters need to be united, the metrics (distances) according to Ward is used (see Chap. 7). The metrics of distance is defined so:

$$d_{rs} = \frac{n_r n_s}{n_r + n_s} \|W_r - W_s\|^2, \quad (2.45)$$

where r and s are indexes of clusters; n_r and n_s are number of patterns in the corresponding clusters; W_s and W_r are vectors of the gravity centers of these clusters. Two clusters s and r unite (merge) if their metrics of d_{rs} is the smallest. For the new created cluster q its weight is defined thus:

$$W_q = \frac{1}{n_r + n_s} (n_r W_r + n_s W_s), \quad (2.45)$$

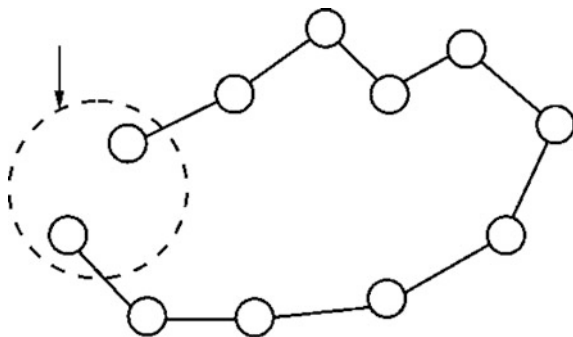
and $n_q = n_r + n_s$.

It's worth to note that for preservation of topological structure two clusters can be integrated only if they are adjacent (neighbors).

The main advantage of SOM consists in easy visualization and interpretation of the clusters created on the Map. In addition to visualization of the full map presented in Fig. 2.17b, separate components of vectors can be visualized, i.e. for each input feature the separate Map for visualization (display) of values distribution of this feature in space can be constructed, using the corresponding color gamut. Such Map and the planes of features can be used for research and the analysis of data. For example, the marked region on the visualized map can be designed on the feature plane to find distribution of values of the corresponding input parameters (features) for this region. In Fig. 2.16c, d SOM's for the third and fourth features for a problem of the irises clustering are presented. The learned SOM can be also used as the classifier [11] as information on clusters is inaccessible during learning process, (SOM) it is necessary to investigate the clusters created on the Map manually and to assign appropriate tags of classes. Further the vector of input data is entered into the map and the winner neuron is defined. The corresponding tag of a cluster to which the entered input vector belongs, is used as a name (a number) of a class.

In the mode of a recall SOM can be used for interpolation of the missed values in a pattern. When entering the input of such pattern, ignoring inputs with the

Fig. 2.17 Area of violation of a continuity of mapping using SOMs



missed values the winner neuron is defined. Further the missed value is determined by the corresponding feature value of a winner neuron, or by interpolation of values of the neighbor neurons.

The described topographic maps give an evident presentation of structure of data in multidimensional input space, which geometry we aren't able to imagine otherwise [10]. Visualization of multidimensional information is the main use of Kohonen's maps.

Note that in consent with the general everyday principle "free lunches don't happen" topographic maps keep the proximity relation only locally, i.e. neighbors on the map of area are close and in the initial space, but not on the contrary (Fig. 2.17). Generally there is no mapping cutting the dimension and keeping the relation of proximity globally.

In Fig. 2.17 the arrow shows the area of violation of a continuity of mapping, neighbors points on the plane are displayed on the opposite ends of the map. The convenient instrument of visualization is the coloring of topographic maps how it used on usual maps. Each feature generates the corresponding coloring of the map by average value of this feature at the data which got to this cell [10].

Having collected maps of all of the interesting features, we'll receive the topographical atlas giving an integrated presentation of the structure of multidimensional data. Self-learning Kohonen's networks are widely used for data preprocessing at pattern recognition in space of very big dimension. In this case, that procedure to be effective, it is required to compress at first input information by one or another way:

1. or to lower dimension, having defined significant features;
2. or to make quantization of data.

SOM are applied to the solution of a wide range of real problems, including the analysis of images, recognition of the speech, the analysis of musical patterns, processing of signals, robotics, telecommunications, data mining of the hidden knowledge and the analysis of time series [9].

References

1. Hopfield, J.J.: Neural Networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, **79**, 2554—2558 (1982)
2. Zaychenko, Y.P.: Fundamentals of intellectual systems design. Kiev. Publishing House, "Slovo", pp. 352 (2004) (rus)
3. Chung, F.L., Lee, T.: Fuzzy competitive learning. *Neural Netw.* **7**, 539–552 (1994)
4. Deb, K., Joshi, D., Anand. A.: Real-coded evolutionary algorithms with parent-centric recombination. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 61–66, (2002)
5. Hopfield, J.J.: Neurons, dynamics and computation. *Phys. Today*, **47**, 40–46 (1994)

6. Heykin, S.: Neural networks. Full course. (2nd edn). Transl. engl. Moscow.-Publishing House "Williams". pp. 1104 (2006). (rus)
7. Hebb, D.O.: The Organization of Behavior: A Neuropsychological Theory. Wiley, New York (1949)
8. Kohonen, T.: Self-Organization and Associative Memory. (3rd edn). Springer, New York (1988)
9. Osovsky, S.: Neural networks for information processing, transl. from pol.—M.: Publishing house Finance and Statistics. pp. 344 (2002). (rus)
10. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biol. Cybern. **43**, 59–69 (1982)
11. Engelbrecht, A.: Computational Intelligence. An Introduction (2nd edn). John Wiley & Sons, Ltd., pp. 630 (2007)

The Fundamentals of Computational Intelligence:
System Approach

Zgurovsky, M.; Zaychenko, Y.P.

2017, XX, 375 p. 143 illus., 70 illus. in color., Hardcover

ISBN: 978-3-319-35160-5