Chapter 1
Artificial Neural Network

Multivariate time series analysis in climate and environmental research always requires to process huge amount of data. Inspired by human nervous system, the *artificial neural network* methodology is a powerful tool to handle this kind of difficult and challenge problems and has been widely used to investigate mechanism of climate change and predict the climate change trend. The main advantage is that artificial neural networks make full use of some unknown information hidden in climate data although they cannot extract it. In this chapter, we will introduce various neural networks, including linear networks, radial basis function networks, generalized regression networks, Kohonen self-organizing networks, learning vector quantization networks, and Hopfield networks.

1.1 Network Architectures

Artificial neural network is a structure of interconnected units of large number of *neurons*. Each neuron in the network is able to receive input signals, to process them, and to send an output signal. It consists of a set of the weighted synapses, an adder for summing the input data weighted by the respective synaptic strength, and an activation function for limiting the amplitude of the output of the neuron. Network architectures have two fundamentally different classes including multilayer feedforward networks and recurrent networks.

1.1.1 Multilayer Feedforward Networks

Feedforward networks are currently being used in a variety of climate and environment applications with great success. It consists of a number of neurons organized in

Z. Zhang, *Multivariate Time Series Analysis in Climate and Environmental Research*, https://doi.org/10.1007/978-3-319-67340-0_1

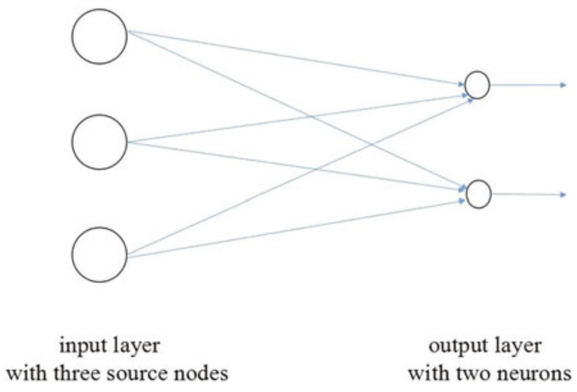


Fig. 1.1 Single-layer feedforward networks

layers. Every neuron in a layer is connected with all the neurons in the previous layer. These connections are not all equal; each connection may have a different strength or weight.

The oldest and simplest artificial neural networks is a single-layer feedforward neural network (see Fig. 1.1). It consists of an input layer of source nodes and an output layer of neurons, where source nodes are projected directly onto the output layer of neurons. The word "single-layer" means that the neural network has only a layer. The layer of source nodes is not counted because no computation is performed.

A multilayer feedforward network consists of an input layer of source nodes, one or more *hidden layers*, and an output layer of neurons (see Fig. 1.2). The hidden layers in the network are not seen directly from either the input or output layer of the network. These hidden layers enable the neural network to extract the higher-order statistical features from its input. Neurons in hidden layers are correspondingly called *hidden neurons*. These hidden neurons have a function to intervene between external input and the network output in some useful manner.

The source nodes in the input layer supply respective elements of the activation pattern to constitute input signals applied to neurons in the first hidden layer. The output signals of neurons in the first hidden layer only are used as input signals to neurons in the second hidden layer. Generally, the output signals of neurons in each hidden layer only are used as input signals to neurons in the adjacent forward hidden layer. There is no connection among neurons in the same layer. Finally, the output signals of neurons in the last hidden layer only are used as input signals to neurons in the output layer. The set of output signals of the neurons in the output layer of the network constitute the overall response of the network to the activation pattern supplied by the source nodes in the input layer of the network.

If every neuron in each layer of the multilayer feedforward network is connected to every neuron in the next layer, then this kind of neural network is called *fully connected*. The simplest fully connected multilayer feedforward network is the network with one hidden layer and one output layer. If such network has m source nodes, h hidden neurons, and n output neurons, for the sake of brevity, this fully connected

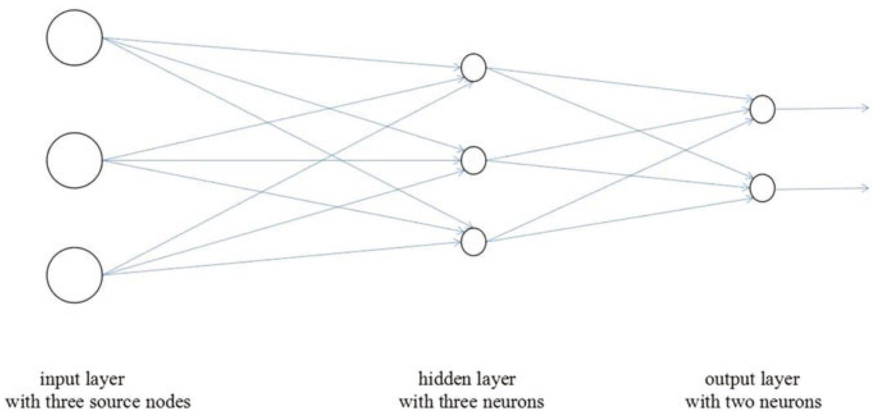


Fig. 1.2 Multilayer feedforward network with a single hidden layer (3-3-2 Network)

multilayer feedforward network is referred to as an $m - h - n$ network. In general, for a fully connected multilayer network with k hidden layers and one output layer, if it has m source nodes in the input layer, h_1 neurons in the first hidden layer, h_2 neurons in the second hidden layer, ..., h_k neurons in the kth hidden layer, and n output neurons, then it is referred to as an $m - h_1 - h_2 - \cdots - h_k - n$ network. If some synaptic connections are missing from the multilayer feedforward network, then the network is called *partially connected*.

1.1.2 Recurrent Networks

Recurrent networks have self-feedback loops or not, the feedback loops involve the use of particular branches composed of unit-delay elements, and recurrent networks also have hidden neurons or not. A recurrent neural network distinguishes from feedforward networks in that it has at least one feedback loop. This offers a lot of flexibility and can approximate arbitrary dynamical systems with arbitrary precision. The network with a single-loop feedback is called a *single-loop feedback network*.

Consider a single-loop feedback network. Denote its input signal by $x(n)$, internal signal by $\widetilde{x}(n)$, and output signal by $y(n)$, where $x(n)$, $\widetilde{x}(n)$, $y(n)$ are dependent on the discrete-time variable n. Assume that the network consists of a forward path $\mathbf{A}$ and a feedback path $\mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ are operators (see Fig. 1.3).

Assume that the output of the forward channel determines in part its own output through the feedback channel. Then, the input and output satisfy the following relationship:

$$\begin{aligned} y(n) &= \mathbf{A}[\widetilde{x}(n)], \\ \widetilde{x}(n) &= x(n) + \mathbf{B}[y(n)], \end{aligned}$$

Fig. 1.3 Single-loop feedback system

$$\begin{tikzcd}
x(n) \arrow[r] & \widetilde{x}(n) \arrow[r, "\mathbf{A}"'] & y(n) \arrow[l, bend right=70, "\mathbf{B}"']
\end{tikzcd}$$

where $\mathbf{A}[\widetilde{x}(n)]$ and $\mathbf{B}[y(n)]$ mean that the operators $\mathbf{A}$ and $\mathbf{B}$ act on $\widetilde{x}(n)$ and $y(n)$, respectively. Eliminating $\widetilde{x}(n)$ from both equations, we get

$$y(n) = \frac{\mathbf{A}}{1 - \mathbf{AB}}[x(n)], \tag{1.1.1}$$

where $\frac{\mathbf{A}}{1-\mathbf{AB}}$ is called a *closed-loop operator* and $\mathbf{AB}$ is called an *open-loop operator*. In general, $\mathbf{AB} \neq \mathbf{BA}$.

If the operator $\mathbf{A}$ is a fixed weight w and the operator $\mathbf{B}$ is a unit-delay operator $\mathbf{z}^{-1}$ whose output is delayed with respect to the input by one time unit, then the closed-loop operator becomes

$$\frac{\mathbf{A}}{1 - \mathbf{AB}} = \frac{w}{1 - w\mathbf{z}^{-1}} = \sum_{l=0}^{\infty} w^{l+1}\mathbf{z}^{-l}.$$

Substituting it into (1.1.1), we get

$$y(n) = \sum_{l=0}^{\infty} w^{l+1}\mathbf{z}^{-l}[x(n)],$$

where $\mathbf{z}^{-l}[x(n)]$ means that the operator $\mathbf{z}^{-l}$ acts on $x(n)$. Since $\mathbf{z}^{-1}$ is a unit-delay operator,

$$\mathbf{z}^{-l}[x(n)] = x(n-l),$$

Furthermore,

$$y(n) = \sum_{l=0}^{\infty} w^{l+1}x(n-l).$$

From this, the dynamic behavior of the single-loop feedback network with the fixed weight w and the unit-delay operator $\mathbf{z}^{-1}$ is determined by the weight w. When $|w| < 1$, the output signal $y(n)$ is *convergent* exponentially. In this case, the system is *stable*. When $|w| \geq 1$, the output signal $y(n)$ is *divergent*. In this case, the system is *unstable*. If $|w| = 1$, the divergence is linear. If $|w| > 1$, the divergence is exponential.

1.2 Perceptrons

The perceptron is a kind of neural networks that can decide whether an input belongs to some specific class. It was the first algorithmically described neural network. Perceptrons can be classified into Rosenblatt's perceptron and multilayer perceptrons.

1.2.1 Rosenblatt's Perceptron

Rosenblatt's perceptron is a network with m input source nodes and a single output neuron, a more general computational model than McCulloch–Pitts model. The perceptron belongs to basically a single-layer neural network and consists of a single neuron with adjustable synaptic weight and bias.

Rosenblatt's perceptron can be described mathematically by the pair of equations:

$$\begin{cases} v = \sum\limits_{i=1}^{m} w_i x_i + b, \\ y = \varphi(v) \end{cases} \tag{1.2.1}$$

or by an equivalent equation:

$$y = \varphi\left(\sum_{i=1}^{m} w_i x_i + b \right),$$

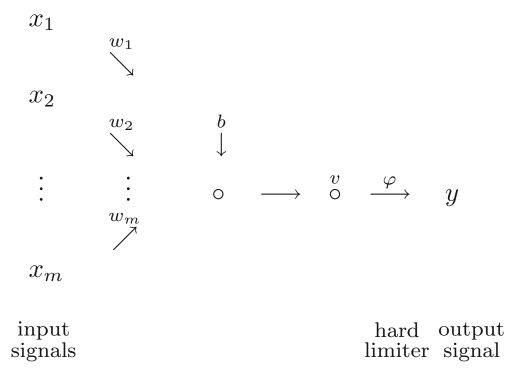
where x_i $(i = 1, \dots, m)$ are the input signals applied to the perceptron, w_i $(i = 1, \dots, m)$ are the synaptic weights of the perceptron, b is the externally applied bias, v is called the *induced local field* (or linear combiner), φ is the activation function (or hard limiter), and y is the output signal of the perceptron (see Fig. 1.4). The Rosenblatt's perceptron consists of a linear combiner followed by a hard limiter, and the hard limiter $\varphi(v)$ determines the output of the perceptron in terms of the induced local field v.

Let $x_0 = 1$ and $w_0 = b$ be the input and the weight of a new synapse. An equivalent form of (1.2.1) is

$$\begin{cases} v = \sum\limits_{i=0}^{m} w_i x_i, \\ y = \varphi(v) \end{cases} \quad \text{or} \quad y = \varphi\left(\sum_{i=0}^{m} w_i x_i \right).$$

The activation function in Rosenblatt's perceptron is a threshold function as follows:

Fig. 1.4 Rosenblatt's perceptron



(a) The Heaviside function is defined as

$$\varphi(v) = \begin{cases} 1 \text{ if } v > 0, \\ 0 \text{ if } v \le 0. \end{cases}$$

If v is the induced local field and $v = \sum_{i=1}^{m} w_i x_i + b$, then the corresponding output is expressed as

$$y = \begin{cases} 1 \text{ if } v > 0, \\ 0 \text{ if } v \le 0. \end{cases}$$

(b) The signum function is defined as

$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0, \\ 0 & \text{if } v = 0, \\ -1 & \text{if } v < 0. \end{cases}$$

If v is the induced local field and $v = \sum_{i=1}^{m} w_i x_i + b$, then the corresponding output is expressed as

$$y = \begin{cases} 1 & \text{if } v > 0, \\ 0 & \text{if } v = 0, \\ -1 & \text{if } v < 0. \end{cases}$$

Based on the feature of the harder limiter, Rosenblatt's perceptron can classify correctly the set of externally applied stimuli $x_1, \ldots, x_m$ into one of two classes. The decision rule for the classification is as follows:

- If the input to the hard limiter $v > 0$, when the activation function is the Heaviside function (or the signum function), the points represented by stimuli $x_1, \ldots, x_m$ are assigned to Class 1;
- If the input to the hard limiter $v < 0$, when the activation function is Heaviside function (or the signum function), the points represented by stimuli $x_1, \ldots, x_m$ are assigned to Class 0 (or Class -1).

Here, Class k is the set consisting of points represented by the stimuli $x_1, \ldots, x_m$ having the output $y = k$.

The input to the hard limiter $v = 0$, i.e., the hyperplane

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

is referred to as the *decision boundary* of two classes. The simplest Rosenblatt's perceptron with two source nodes and a single output neuron can classify the set of externally applied stimuli x_1, x_2 into one of two classes, and the decision boundary of these two classes is a straight line:

$$w_1 x_1 + w_2 x_2 + b = 0, \qquad (1.2.2)$$

where $w_i (i = 1, 2)$ and b are the synaptic weights and bias of the perceptron, respectively. For example, assume that $w_1 = 1$, $w_2 = 0.5$, $b = -0.5$, and the sampling set of externally applied stimuli x_1, x_2 consists of seven points:

$$\begin{array}{llll} \mathbf{x}^{(1)} = (-1, 2), & \mathbf{x}^{(2)} = (1, 2), & \mathbf{x}^{(3)} = (2, -1), & \\ \mathbf{x}^{(4)} = (1, 1), & \mathbf{x}^{(5)} = (-2, 1), & \mathbf{x}^{(6)} = (-1, -1), & \mathbf{x}^{(7)} = (2, 0). \end{array}$$

By (1.2.2), the decision boundary of two classes is $x_1 + 0.5x_2 - 0.5 = 0$. It is seen that four points $\mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}, \mathbf{x}^{(7)}$ lie above the straight line and the rest $\mathbf{x}^{(1)}, \mathbf{x}^{(5)}, \mathbf{x}^{(6)}$ lie below the straight line. The decision rule for the classification shows that these seven points can be classified into two classes.

1.2.2 Multilayer Perceptron

The architecture of multilayer perceptrons is very different from the single-layer perceptron. It is a perceptron consisting of an input layer of m source nodes, i hidden layers of h_i neurons, and an output layer of n neurons. Each neuron of the network has a differentiable nonlinear activation function. The architecture of a fully connected multilayer perceptron is that a neuron node in any layer of the network is connected to all neuron nodes in the previous layer and the signal flow through the network progresses in a forward direction from left to right and on a layer-by-layer basis.

The activation function used commonly in the multilayer perceptron is the sigmoid function. The sigmoid function is mathematically convenient and is close to linear near the origin while saturating rather quickly when getting away from the origin. This allows multilayer perceptrons to model well both strongly and mildly nonlinear relations. The logistic function and the hyperbolic tangent function are two popular sigmoid functions. The logistic function is defined as

$$\varphi(v) = \frac{1}{1+e^{-av}} \qquad (a>0),$$

where a is its slop parameter. Logistic functions of different slopes can be obtained by varying this parameter. The logistic function is differentiable, and its derivative is

$$\varphi'(v) = \frac{ae^{-av}}{(1+e^{-av})^2} = \frac{a}{1+e^{-av}}\left(1 - \frac{1}{1+e^{-av}}\right) = a\varphi(v)(1-\varphi(v)).$$

The hyperbolic tangent function is defined as

$$\varphi(v) = a\tanh(bv) \qquad (a>0,\ b>0).$$

The hyperbolic tangent function is also differentiable, and its derivative is

$$\begin{aligned}\varphi'(v) &= ab\,\mathrm{sech}^2(bv) = ab(1-\tanh^2(bv)) = \tfrac{b}{a}(a^2-\varphi^2(v))\\ &= \tfrac{b}{a}(a-\varphi(v))(a+\varphi(v)).\end{aligned}$$

The process of the multilayer perceptron includes forward propagation of function signals and backward propagation of error signals which are identified. *Forward propagation of function signals* is such a process that an input signal comes in at the input end of the network, propagates forward neuron by neuron through the network, and emerges at the output end of the network as an output signal. *Backward propagation of error signals* is such a process that the error signal originates at an output neuron of the network and propagates backward layer-by-layer through the network.

Due to one or more hidden layers, the multilayer perceptron can classify nonlinearly separable patterns, while the single-layer perceptron cannot.

Example 1.2.1 Assume that a sampling set consists of four points:

$$\begin{aligned}\mathbf{x}^{(1)} &= (0,0), & \mathbf{x}^{(2)} &= (0,1),\\ \mathbf{x}^{(3)} &= (1,0), & \mathbf{x}^{(4)} &= (1,1).\end{aligned}$$

The XOR problem is to use a perceptron to classify these four points into two classes such that points $\mathbf{x}^{(1)}, \mathbf{x}^{(4)}$ are assigned to a class, and points $\mathbf{x}^{(2)}, \mathbf{x}^{(3)}$ are assigned to another class. There is no possibility to solve the XOR problem using any single-layer perceptron. In fact, assume that a following single-layer perceptron can solve it:

$$\begin{cases} v = w_1x_1 + w_2x_2 + b,\\ y = \varphi(v),\end{cases}$$

where $w_i (i=1,2)$ and b are synaptic weights and bias, respectively, and the activation function φ is the Heaviside function. Since $\mathbf{x}^{(1)} = (0,0)$ and $\mathbf{x}^{(4)} = (1,1)$ are assigned to Class 0, it is clear that $b<0$ and $w_1+w_2+b<0$. Adding them together gives

$$w_1 + w_2 + 2b < 0. \qquad (1.2.3)$$

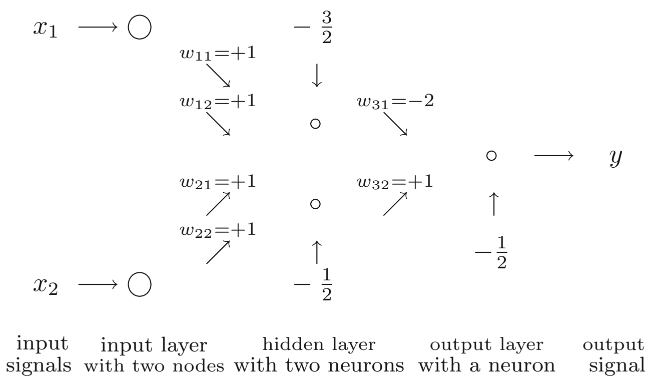


Fig. 1.5 Touretzky–Pomerleau perceptron

Since $\mathbf{x}^{(2)} = (0, 1)$ and $\mathbf{x}^{(3)} = (1, 0)$ are assigned to Class 1, then

$$\begin{aligned} w_2 + b &> 0, \\ w_1 + b &> 0. \end{aligned}$$

Adding them together, we get

$$w_1 + w_2 + 2b > 0.$$

This is contrary to (1.2.3). Thus, there is no possibility to solve the XOR problem using a single-layer perceptron.

Touretzky–Pomerleau perceptron is a multilayer perceptron (see Fig. 1.5), where each "$\circ$" represents a neuron and each neuron has Heaviside function as the activation function. Touretzky–Pomerleau perceptron can solve the XOR problem given by Example 1.2.1.

There are three neurons in Touretzky–Pomerleau perceptron, two of them are hidden neurons in the hidden layer and the remainder one is the output neuron in the output layer. For the top hidden neuron, the synaptic weights and bias are, respectively,

$$\begin{aligned} w_{11} &= w_{12} = 1, \\ b_1 &= -\tfrac{3}{2}. \end{aligned}$$

The straight line

$$l_1: \quad x_1 + x_2 - \frac{3}{2} = 0.$$

is the decision boundary formed by the top hidden neuron. It is clear that the point $\mathbf{x}^{(4)}$ lying above the line l_1 is assigned to Class 1, and points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}$ lying below the line l_1 are assigned to Class 0.

For the bottom hidden neuron, the synaptic weights and bias are, respectively,

$$\begin{aligned} &w_{21} = w_{22} = 1, \\ &b_2 = -\tfrac{1}{2}. \end{aligned}$$

The straight line

$$l_2 : \quad x_1 + x_2 - \frac{1}{2} = 0$$

is the decision boundary formed by the bottom hidden neuron. The points $\mathbf{x}^{(2)}$, $\mathbf{x}^{(3)}$, $\mathbf{x}^{(4)}$ lying above the line l_2 are assigned to Class 1, and the point $\mathbf{x}^{(1)}$ lying below the line l_2 is assigned to Class 0.

For the output neuron, the synaptic weights and bias are, respectively,

$$\begin{aligned} &w_{31} = -2, \qquad w_{32} = 1, \\ &b_3 = -\tfrac{1}{2}. \end{aligned}$$

The output neuron constructs a linear combination of decision boundaries formed by two hidden neurons. The decision boundary formed by the output neuron is a straight line:

$$-2y_1 + y_2 - \frac{1}{2} = 0$$

where y_1 is the output from the top hidden neuron and y_2 is the output from the bottom neuron. The decision rule for the classification is that a point that lies above the line l_1 or below the line l_2 is assigned to Class 0 and a point that lies both below the line l_1 and above the line l_2 is assigned to Class 1. Therefore, according to the decision rule for the classification, the points $\mathbf{x}^{(1)}$, $\mathbf{x}^{(4)}$ are assigned to Class 0 and the points $\mathbf{x}^{(2)}$, $\mathbf{x}^{(3)}$ are assigned to Class 1. So the XOR problem for four points $\mathbf{x}^{(i)}$ is solved using the Touretzky–Pomerleau perceptron.

1.3 Linear Network and Bayes Classifier

Linear neural network distinguishes from Rosenblatt's perceptron in that activation function is a linear function. So the output of the linear network may be any value. A linear network of a neuron can be described by a pair of equations:

$$\begin{cases} v = \sum\limits_{i=1}^{m} w_i x_i + b, \\ y = \varphi(v) = v \end{cases}$$

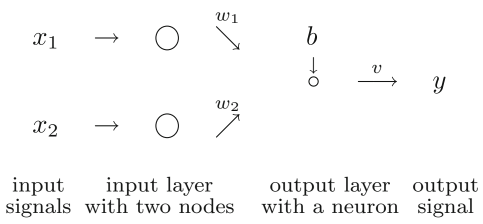


Fig. 1.6 Linear network with double input signals

or simply by an equation:

$$y = \sum_{i=1}^{m} w_i x_i + b = \sum_{i=0}^{m} w_i x_i,$$

where $x_0 = 1$, x_i $(l = 1, \dots, m)$ are the input signals, w_i $(l = 1, \dots, m)$ are the synaptic weights, $w_0 = b$ is the bias, and y is the output signal of the neuron. Similar to Rosenblatt's perceptrons, linear neural network is used for classifying linearly separable patterns. For example, a linear network of a neuron with two source nodes is shown as in Fig. 1.6, where x_1, x_2 are two input signals, y is the output signal, w_1, w_2 are two synaptic weights, b is the bias, and $\circ$ represents the neuron. This network can be described by a pair of equations:

$$\begin{cases} v = w_1 x_1 + w_2 x_2 + b, \\ y = v \end{cases}$$

or simply by an equation: $y = w_1 x_1 + w_2 x_2 + b$.
The corresponding decision boundary is a straight line $w_1 x_1 + w_2 x_2 + b = 0$, and the decision rule for classification is that all points that lie above the straight line are assigned to one class and all points that lie below the straight line are assigned to another class.

Below, we discuss *the Bayes classifier*. Consider a two-class problem represented by classes $\mathcal{B}_i$ in the subspace $\mathbb{R}_i$ $(i = 1, 2)$, where $\widetilde{\mathcal{R}} = \mathcal{R}_1 + \mathcal{R}_2$. Denote by $p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_i)$ $(i = 1, 2)$ the conditional probability density function of a random vector $\mathbf{X}$, given that the observation vector $\mathbf{x}$ is drawn from subspace $\mathbb{R}_i$. In the Bayes classifier, Van Trees defined an *average risk* (AR) of the two-class problem by

$$\begin{aligned} \text{AR} = {} & c_{11} p_1 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1) \mathrm{d}\mathbf{X} + c_{22} p_2 \int_{\mathcal{R}_2} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2) \mathrm{d}\mathbf{X} \\ & + c_{21} p_1 \int_{\mathcal{R}_2} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1) \mathrm{d}\mathbf{X} + c_{12} p_2 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2) \mathrm{d}\mathbf{X}, \end{aligned}$$

where c_{ij} is the cost of deciding in favor of class $\mathcal{B}_i$ and p_i is the prior probability of the observation vector $\mathbf{x}$, and $p_1 + p_2 = 1$. Note that

$$\widetilde{\mathcal{R}} = \mathcal{R}_1 + \mathcal{R}_2.$$

The equivalent form of the average risk is

$$\begin{aligned}\mathrm{AR} &= c_{11}p_1 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)\mathrm{d}\mathbf{X} + c_{22}p_2 \int_{\tilde{\mathcal{R}}-\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2)\mathrm{d}\mathbf{X} \\ &\quad + c_{21}p_1 \int_{\tilde{\mathcal{R}}-\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)\mathrm{d}\mathbf{X} + c_{12}p_2 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2))\mathrm{d}\mathbf{X} \\ &= c_{11}p_1 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)\mathrm{d}\mathbf{X} + c_{22}p_2 \int_{\tilde{\mathcal{R}}} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2)\mathrm{d}\mathbf{X} - c_{22}p_2 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2)\mathrm{d}\mathbf{X} \\ &\quad + c_{21}p_1 \int_{\tilde{\mathcal{R}}} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)\mathrm{d}\mathbf{X} - c_{21}p_1 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)\mathrm{d}\mathbf{X} + c_{12}p_2 \int_{\mathcal{R}_1} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2))\mathrm{d}\mathbf{X}.\end{aligned}$$

From this and

$$\int_{\tilde{\mathcal{R}}} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)\mathrm{d}\mathbf{X} = 1,$$
$$\int_{\tilde{\mathcal{R}}} p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2)\mathrm{d}\mathbf{X} = 1,$$

it follows that the average risk is

$$\mathrm{AR} = c_{21}p_1 + c_{22}p_2 + \int_{\mathcal{R}_1} [p_2(c_{12} - c_{22})p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2) - p_1(c_{21} - c_{11})p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)]\mathrm{d}\mathbf{X},$$

where $c_{21}p_1 + c_{22}p_2$ represents a fixed cost. The Bayes classifier requires that the integrand of the last integral is greater than zero, i.e., $p_2(c_{12} - c_{22})p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2) - p_1(c_{21} - c_{11})p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1) > 0$ or

$$\frac{p_2(c_{12} - c_{22})}{p_1(c_{21} - c_{11})} > \frac{p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)}{p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2)}.$$

The quantities

$$\Lambda(\mathbf{X}) = \frac{p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1)}{p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2)}, \qquad \xi = \frac{p_2(c_{12} - c_{22})}{p_1(c_{21} - c_{11})}.$$

are called *likelihood ratio* and *threshold* of the test, respectively. The likelihood ratio $\Lambda(\mathbf{x})$ and the threshold ξ are both positive. Taking the logarithm of $\Lambda(\mathbf{x})$,

$$\log \Lambda(\mathbf{X}) = \log p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1) - \log p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2). \tag{1.3.1}$$

The quantity $\log \Lambda(\mathbf{X})$ is called log *−likelihood ratio*. Since the logarithm function is a monotonic function, it is more convenient to compute the log −likelihood ratio than the likelihood ratio.

Consider a special two-class problem:

$$\begin{aligned} \text{Class}\,\mathcal{B}_1\colon\ & E[\mathbf{X}] = \mu_1, \\ & E[(\mathbf{X} - \mu_1)(\mathbf{X} - \mu_1)^T] = C; \\ \text{Class}\,\mathcal{B}_2\colon\ & E[\mathbf{X}] = \mu_2, \\ & E[(\mathbf{X} - \mu_2)(\mathbf{X} - \mu_2)^T] = C, \end{aligned} \tag{1.3.2}$$

where **X** is a random vector. In the two-class problem, the mean values of the random vector are different but their covariance matrix of the random vector is the same. Denote by C^{-1} the inverse matrix of the covariance matrix C. Then, the conditional probability density function may be represented as the multivariate Gaussian distribution:

$$p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_i) = \frac{1}{(2\pi)^{\frac{m}{2}}(\det(C))^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_i)^T C^{-1}(\mathbf{x}-\mu_i)\right) \qquad (i=1,2),$$

where m is the dimensionality of the observation vector $\mathbf{x}$ and $\det(C)$ represents the determinant of the matrix C. Assume further in the average risk that

$$p_1 = p_2 = \frac{1}{2}, \quad c_{12} = c_{21}, \quad c_{11} = c_{22} = 0.$$

Note that

$$\begin{aligned}\log p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1) &= -\tfrac{1}{2}(\mathbf{x}-\mu_1)^T C^{-1}(\mathbf{x}-\mu_1) - \log((2\pi)^{\frac{m}{2}}(\det(C))^{\frac{1}{2}}),\\ \log p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2) &= -\tfrac{1}{2}(\mathbf{x}-\mu_2)^T C^{-1}(\mathbf{x}-\mu_2) - \log((2\pi)^{\frac{m}{2}}(\det(C))^{\frac{1}{2}}).\end{aligned}$$

By (1.3.1), the log −likelihood ratio is

$$\begin{aligned}\log\Lambda(\mathbf{X}) &= \log p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_1) - \log p_{\mathbf{X}}(\mathbf{X}|\mathcal{B}_2)\\ &= \left(-\tfrac{1}{2}(\mathbf{x}-\mu_1)^T C^{-1}(\mathbf{x}-\mu_1) - \log((2\pi)^{\frac{m}{2}}(\det(C))^{\frac{1}{2}})\right)\\ &\quad - \left(-\tfrac{1}{2}(\mathbf{x}-\mu_2)^T C^{-1}(\mathbf{x}-\mu_2) - \log((2\pi)^{\frac{m}{2}}(\det(C))^{\frac{1}{2}})\right).\end{aligned}$$

A direct computation shows that

$$\begin{aligned}\log\Lambda(\mathbf{X}) &= -\tfrac{1}{2}(\mathbf{x}^T-\mu_1^T)C^{-1}(\mathbf{x}-\mu_1) + \tfrac{1}{2}(\mathbf{x}^T-\mu_2^T)C^{-1}(\mathbf{x}-\mu_2)\\ &= -\tfrac{1}{2}\mathbf{x}^T C^{-1}\mathbf{x} + \tfrac{1}{2}\mu_1^T C^{-1}\mathbf{x} + \tfrac{1}{2}\mathbf{x}^T C^{-1}\mu_1 - \tfrac{1}{2}\mu_1^T C^{-1}\mu_1\\ &\quad +\tfrac{1}{2}\mathbf{x}^T C^{-1}\mathbf{x} - \tfrac{1}{2}\mu_2^T C^{-1}\mathbf{x} - \tfrac{1}{2}\mathbf{x}^T C^{-1}\mu_2 + \tfrac{1}{2}\mu_2^T C^{-1}\mu_2\\ &= \tfrac{1}{2}(\mu_1-\mu_2)^T C^{-1}\mathbf{x} + \tfrac{1}{2}\mathbf{x}^T C^{-1}(\mu_1-\mu_2) + \tfrac{1}{2}(\mu_2^T C^{-1}\mu_2 - \mu_1^T C^{-1}\mu_1)\\ &= (\mu_1-\mu_2)^T C^{-1}\mathbf{x} + \tfrac{1}{2}(\mu_2^T C^{-1}\mu_2 - \mu_1^T C^{-1}\mu_1).\end{aligned} \tag{1.3.3}$$

By $c_{12}=c_{21}$, $c_{11}=c_{22}=0$, and $p_1=p_2=\frac{1}{2}$, the threshold of the test is

$$\xi = \frac{p_2(c_{12}-c_{22})}{p_1(c_{21}-c_{11})} = \frac{\frac{1}{2}(c_{12}-0)}{\frac{1}{2}(c_{21}-0)} = 1.$$

In (1.3.3), let

$$\begin{aligned}&y = \log\Lambda(\mathbf{X}),\\ &W^T = (\mu_1-\mu_2)^T C^{-1},\\ &b = \tfrac{1}{2}(\mu_2^T C^{-1}\mu_2 - \mu_1^T C^{-1}\mu_1).\end{aligned} \tag{1.3.4}$$

Then, Bayes classifier for multivariate Gaussian distribution, or simply, *Gaussian-distribution classifier*, is

$$y = W^T\mathbf{x} + b. \tag{1.3.5}$$

This equation shows that the Gaussian-distribution classifier is a linear network with the synaptic weight W and the bias b. It is seen from (1.3.5) that the decision boundary of the special two classes $\mathcal{B}_1, \mathcal{B}_2$ is the hyperplane:

$$W^T\mathbf{x} + b = 0.$$

In the Gaussian-distribution classifier, assume that the covariance matrix C is given by $C = aI$, where $a > 0$ is a positive constant and I is the identity matrix. We will find the synaptic-weight vector and bias as well as the representation of the Gaussian-distribution classifier.

Note that $C^{-1} = (aI)^{-1} = \frac{1}{a}I^{-1} = \frac{1}{a}I$. By (1.3.4), the weight vector is equal to

$$W = C^{-1}(\mu_1 - \mu_2) = \frac{1}{a}I(\mu_1 - \mu_2) = \frac{1}{a}(\mu_1 - \mu_2)$$

and the bias is equal to

$$b = \frac{1}{2}(\mu_2^T C^{-1}\mu_2 - \mu_1^T C^{-1}\mu_1) = \frac{1}{2a}(\mu_2^T\mu_2 - \mu_1^T\mu_1).$$

By (1.3.5), the Gaussian-distribution classifier becomes

$$y = W^T\mathbf{x} + b = \frac{1}{a}\left(\mu_1^T - \mu_2^T\right)\mathbf{x} + \frac{1}{2a}(\mu_2^T\mu_2 - \mu_1^T\mu_1).$$

Example 1.3.1 In the one-dimensional case, consider the following two-class problem:

$$\begin{aligned} \text{Class } \mathcal{B}_1\text{: } & E[X] = \mu_1, \\ & E[(X-\mu_1)^2] = C, \\ \text{Class } \mathcal{B}_2\text{: } & E[X] = \mu_2, \\ & E[(X-\mu_2)^2] = C, \end{aligned} \tag{1.3.6}$$

where X is a random variable and μ_1, μ_2, C are all real numbers. So $\mu_1^T = \mu_1$, $\mu_2^T = \mu_2$, and $C^{-1} = \frac{1}{C}$. By (1.3.4), the synaptic weight and the bias are all real numbers and

$$\begin{aligned} W &= C^{-1}(\mu_1 - \mu_2) = \tfrac{\mu_1-\mu_2}{C}, \\ b &= \tfrac{1}{2}(\mu_2^T C^{-1}\mu_2 - \mu_1^T C^{-1}\mu_1) = \tfrac{1}{2C}(\mu_2^2 - \mu_1^2). \end{aligned}$$

By (1.3.5), the univariate Gaussian-distribution classifier is

$$y = W^T x + b = \frac{\mu_1 - \mu_2}{C} x + \frac{1}{2C}(\mu_2^2 - \mu_1^2)$$

and the decision boundary of the two-class problem (1.3.6) is a point:

$$x^* = -\frac{\frac{1}{2C}(\mu_2^2 - \mu_1^2)}{\frac{\mu_1 - \mu_2}{C}} = \frac{\mu_1 + \mu_2}{2}.$$

Assume that $\mu_1 = -10$, $\mu_2 = 10$, and $C = 1$. For the two-class problem (1.3.6), the synaptic weight $W = -20$ and the bias $b = 0$, the univariate Gaussian-distribution classifier is $y = -20x$, and the decision boundary is $x = 0$. The point $x(x > 0)$ and the point $x(x < 0)$ are assigned to Class $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively.

Example 1.3.2 In the two-dimensional case, the means $\mu_1,\ \mu_2$, and the inverse matrix of the covariance matrix are denoted by

$$\mu_1 = (\mu_{11},\ \mu_{12})^T, \qquad \mu_2 = (\mu_{21},\ \mu_{22})^T,$$
$$C^{-1} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}.$$

By (1.3.4), the synaptic weight $W = (W_1, W_2)^T$ is

$$\begin{aligned} W = C^{-1}(\mu_1 - \mu_2) &= \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} \begin{pmatrix} \mu_{11} - \mu_{21} \\ \mu_{12} - \mu_{22} \end{pmatrix} \\ &= \begin{pmatrix} c_{11}(\mu_{11} - \mu_{21}) + c_{12}(\mu_{12} - \mu_{22}) \\ c_{21}(\mu_{11} - \mu_{21}) + c_{22}(\mu_{12} - \mu_{22}) \end{pmatrix}. \end{aligned}$$

Let $\mathbf{x} = (x_1, x_2)^T$. Then,

$$\begin{aligned} W^T\mathbf{x} &= (c_{11}(\mu_{11} - \mu_{21}) + c_{12}(\mu_{12} - \mu_{22}),\ c_{21}(\mu_{11} - \mu_{21}) + c_{22}(\mu_{12} - \mu_{22})) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= c_{11}(\mu_{11} - \mu_{21})x_1 + c_{12}(\mu_{12} - \mu_{22})x_1 + c_{21}(\mu_{11} - \mu_{21})x_2 + c_{22}(\mu_{12} - \mu_{22})x_2 \\ &= (\mu_{11} - \mu_{21})(c_{11}x_1 + c_{21}x_2) + (\mu_{12} - \mu_{22})(c_{12}x_1 + c_{22}x_2). \end{aligned}$$

By (1.3.4), the bias is

$$b = \frac{1}{2}(\mu_2^T C^{-1}\mu_2 - \mu_1^T C^{-1}\mu_1).$$

Two terms $\mu_2^T C^{-1}\mu_2$ and $\mu_1^T C^{-1}\mu_1$ are computed, respectively, as follows:

$$\begin{aligned} \mu_2^T C^{-1}\mu_2 &= (\mu_{21}, \mu_{22}) \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} \begin{pmatrix} \mu_{21} \\ \mu_{22} \end{pmatrix} \\ &= (\mu_{21}c_{11} + \mu_{22}c_{21},\ \mu_{21}c_{12} + \mu_{22}c_{22}) \begin{pmatrix} \mu_{21} \\ \mu_{22} \end{pmatrix} \\ &= (\mu_{21}c_{11} + \mu_{22}c_{21})\mu_{21} + (\mu_{21}c_{12} + \mu_{22}c_{22})\mu_{22} \\ &= \mu_{21}^2 c_{11} + \mu_{21}\mu_{22}c_{21} + \mu_{21}\mu_{22}c_{12} + \mu_{22}^2 c_{22}. \end{aligned}$$

$$\begin{aligned}\mu_1^T C^{-1}\mu_1 &= (\mu_{11},\mu_{12})\begin{pmatrix} c_{11} & c_{12}\\ c_{21} & c_{22}\end{pmatrix}\begin{pmatrix}\mu_{11}\\ \mu_{12}\end{pmatrix}\\ &= (\mu_{11}c_{11}+\mu_{12}c_{21},\ \mu_{11}c_{12}+\mu_{12}c_{22})\begin{pmatrix}\mu_{11}\\ \mu_{12}\end{pmatrix}\\ &= (\mu_{11}c_{11}+\mu_{12}c_{21})\mu_{11}+(\mu_{11}c_{12}+\mu_{12}c_{22})\mu_{12}\\ &= \mu_{11}^2c_{11}+\mu_{11}\mu_{22}c_{11}+\mu_{11}\mu_{12}c_{12}+\mu_{12}^2c_{22}.\end{aligned}$$

So the bias is

$$\begin{aligned}b &= \tfrac{1}{2}(\mu_{21}^2c_{11}+\mu_{21}\mu_{22}c_{21}+\mu_{21}\mu_{22}c_{12}+\mu_{22}^2c_{22})\\ &\quad-\tfrac{1}{2}(\mu_{11}^2c_{11}+\mu_{11}\mu_{12}c_{21}+\mu_{11}\mu_{12}c_{12}+\mu_{12}^2c_{22})\\ &= \tfrac{1}{2}((\mu_{21}^2-\mu_{11}^2)c_{11}+(\mu_{21}\mu_{22}-\mu_{11}\mu_{12})(c_{21}+c_{12})+(\mu_{22}^2-\mu_{12}^2)c_{22})\end{aligned}$$

By (1.3.5), the bivariate Gaussian-distribution classifier is

$$\begin{aligned}\mathbf{y} &= W^T\mathbf{x}+b\\ &= (\mu_{11}-\mu_{21})(c_{11}x_1+c_{21}x_2)+(\mu_{12}-\mu_{22})(c_{12}x_1+c_{22}x_2)\\ &\quad+\tfrac{1}{2}(\mu_{21}^2-\mu_{11}^2)c_{11}+\tfrac{1}{2}(\mu_{21}\mu_{22}-\mu_{11}\mu_{12})(c_{21}+c_{12})+\tfrac{1}{2}(\mu_{22}^2-\mu_{12}^2)c_{22}.\end{aligned}$$

1.4 Radial Basis Function Network

Radial basis function network is derived from the theory of function approximation and interpolation. It uses radial basis functions as activation functions.

1.4.1 Radial Basis Function

For a given set of distinct points $\mathbf{x}_1,\ldots,\mathbf{x}_N\in\mathbb{R}^m$, the radial basis function technique is to find a function $F(\mathbf{x})$ that has the form:

$$F(\mathbf{x})=\sum_{j=1}^N w_j\varphi(\|\mathbf{x}-\mathbf{x}_j\|)=\sum_{j=1}^N w_j\varphi(\mathbf{x},\mathbf{x}_j), \tag{1.4.1}$$

where $w_j\,(j=1,\ldots,N)$ are components of the weight vector $\mathbf{w}$, $\{\varphi(\mathbf{x},\mathbf{x}_j)\}_{j=1,\ldots,N}$ is a set of *radial basis functions*, and $\mathbf{x}_j$ is the *center* of the radial basis function $\varphi(\mathbf{x},\mathbf{x}_j)=\varphi(\|\mathbf{x}-\mathbf{x}_j\|)$; here, $\|\cdot\|$ is the Euclidean distance.

The radial basis functions used widely are as follows.

(a) Gaussian function:

$$\varphi(x)=\exp\left(-\frac{x^2}{2\sigma^2}\right)\quad\text{for some }\sigma>0\text{ and }x\in\mathbb{R},$$

where σ is the width of the Gaussian function.

(b) Multiquadric function: $\varphi(x) = (x^2 + c^2)^{\frac{1}{2}}$ for some $c > 0$ and $x \in \mathbb{R}$.

(c) Inverse multiquadric function: $\varphi(x) = (x^2 + c^2)^{-\frac{1}{2}}$ for some $c > 0$ and $x \in \mathbb{R}$.

1.4.2 Interpolation

The interpolation technique is used for finding the weight vector $\mathbf{w}$.

Given a set of distinct points $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^m$ and a corresponding set of real numbers $d_1, \ldots, d_N \in \mathbb{R}$. The interpolation problem is to seek a function $F : \mathbb{R}^m \to \mathbb{R}$ satisfying the interpolation condition:

$$F(\mathbf{x}_i) = d_i \qquad (i = 1, \ldots, N), \tag{1.4.2}$$

where $F(\mathbf{x})$ is called the *interpolation surface* (or the *interpolation function*). The interpolation surface $F(\mathbf{x})$ is constrained to pass through all the training data points $\{\mathbf{x}_i, d_i\}_{i=1,\ldots,N}$, where N is called the *size of the training sample*. The combination of (1.4.1) and (1.4.2) gives

$$d_i = \sum_{j=1}^{N} w_j \varphi(\mathbf{x}_i, \mathbf{x}_j) \qquad (i = 1, \ldots, N).$$

In more detail,

$$\begin{aligned} d_1 &= \sum_{j=1}^{N} w_j \varphi(\mathbf{x}_1, \mathbf{x}_j), \\ d_2 &= \sum_{j=1}^{N} w_j \varphi(\mathbf{x}_2, \mathbf{x}_j), \\ &\vdots \\ d_N &= \sum_{j=1}^{N} w_j \varphi(\mathbf{x}_N, \mathbf{x}_j). \end{aligned} \tag{1.4.3}$$

This is a system of N equations with N unknown weights $w_j (j = 1, \ldots, N)$.

Let $\varphi_{ij} = \varphi(\mathbf{x}_i, \mathbf{x}_j)$ $(i, j = 1, \ldots, N)$. Then, the system (1.4.3) with N unknown w_j can be rewritten in the matrix form

$$\begin{pmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{pmatrix},$$

where N is the size of the training sample.

Let

$$\Phi = \begin{pmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{pmatrix}, \qquad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}, \qquad \mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{pmatrix}.$$

Then, the system (1.4.3) has the compact form

$$\Phi \mathbf{w} = \mathbf{d}, \tag{1.4.4}$$

where the matrix Φ is called the *interpolation matrix* and the vectors $\mathbf{w}$ and $\mathbf{d}$ are the *linear weight vector* and *desired response vector*, respectively. Micchlli Interpolation Theorem (see Chap. 2) shows that the interpolation matrix Φ is non-singular when $\{\mathbf{x}_i\}_{i=1,\dots,N}$ is a set of distinct points in $\mathbb{R}^{m_0}$. Therefore, its inverse matrix Φ^{-1} exists. So the weight vector $\mathbf{w}$ is given by $\mathbf{w} = \Phi^{-1}\mathbf{d}$.

The *architecture* of radial basis function network is a feedforward network with layered structure. For example, the architecture of a radial basis function network with an input layer, a single hidden layer, and an output layer consisting of a single unit is described as follows.

Given a set of N distinct points $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^m$, i.e., the size of the input layer is m. The single hidden layer consists of N computation units. Each computational unit is described by a radial basis function:

$$\varphi(\mathbf{x}, \mathbf{x}_j) = \varphi(\| \mathbf{x} - \mathbf{x}_j \|) \qquad (j = 1, \dots, N),$$

where the jth input data point $\mathbf{x}_j$ is the center of the radial basis function and $\mathbf{x}$ is the signal applied to the input layer. The connections between the source nodes and hidden units are direct connections with no weights. The output layer consists of a single computational unit. The size of the output layer is 1. It is characterized by the weight vector $\mathbf{w} = (w_1, \dots, w_N)$, and the approximating function is

$$F(\mathbf{x}) = \sum_{j=1}^{N} w_j \varphi(\mathbf{x}, \mathbf{x}_j).$$

However, in practice, since the training sample $\{\mathbf{x}_i, d_i\}_{i=1,\dots,N}$ is often noisy, it could be wasteful of computational resources to have a hidden layer of the same size as the input layer. The size K of the hidden layer is required to be less than N, and then the corresponding approximating function is the sum of K weighted radial basis functions.

1.4.3 Receptive Field

The *receptive field* of a computational unit (e.g., the hidden unit) in a neural network is that region of the sensory field (e.g., the input layer of source nodes) from which an adequate sensory stimulus (e.g., pattern) will elicit a response. The receptive field of a computational unit is defined as

$$\psi(\mathbf{x}) = \varphi(\mathbf{x}, \mathbf{x}_j) - \alpha,$$

where α is some positive constant and $\varphi(\mathbf{x}, \mathbf{x}_j)$ is a radial basis function with center $\mathbf{x}_j$.

When Gaussian function is used as a radial basis function, Gaussian hidden unit (i.e., each computational unit in the hidden layer) is given by

$$\varphi(\mathbf{x}, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma_j^2} \parallel \mathbf{x} - \mathbf{x}_j \parallel^2\right) \qquad (j = 1, \ldots, N),$$

where σ_j is the width of the jth Gaussian function with center $\mathbf{x}_j$.

If all the Gaussian hidden units are assigned a common width σ, then the parameter that distinguishes one hidden unit from another is the center $\mathbf{x}_j$. The receptive field of Gaussian hidden unit is

$$\psi(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2} \parallel \mathbf{x} - \mathbf{x}_j \parallel^2\right) - \alpha,$$

where $0 < \alpha < 1$. Since the minimum permissible value of $\psi(\mathbf{x})$ is zero, it gives

$$\parallel \mathbf{x} - \mathbf{x}_j \parallel = \sigma\sqrt{2\log(1/\alpha)}.$$

It is seen that the receptive field of Gaussian hidden unit is a multidimensional surface centered symmetrically around the point $\mathbf{x}_j$.

The one-dimensional receptive field of Gaussian hidden unit is the closed interval:

$$|x - x_j| \le \sigma\sqrt{2\log(1/\alpha)}.$$

where the center and the radius of the closed interval are x_j and $r = \sigma\sqrt{2\log(1/\alpha)}$, respectively.

The two-dimensional receptive field of Gaussian hidden unit is a circular disk:

$$|x_1 - x_{j1}|^2 + |x_2 - x_{j2}|^2 \le 2\sigma^2 \log(1/\alpha),$$

where the center and the radius of the circular disk are $\mathbf{x}_j = (x_{j1}, x_{j2})$ and $r = \sigma\sqrt{2\log(1/\alpha)}$, respectively.

1.5 Generalized Regression Network

Generalized regression networks are based on the nonlinear kernel regression analysis. They belong to radial basis function networks.

The *kernel regression* builds on the notion of density estimation. Let the random vector $\mathbf{x}$ represent a regressor and the random variable y represent an observation. Then, the regression function $f(\mathbf{x})$ equals the conditional mean of the observation y given the regressor $\mathbf{x}$, i.e.,

$$f(\mathbf{x}) = E[Y|\mathbf{X}] = \int_{\mathbb{R}} y p_{Y|\mathbf{X}}(y|\mathbf{x})\mathrm{d}y, \tag{1.5.1}$$

where $p_{Y|\mathbf{X}}(y|\mathbf{x})$ is the conditional probability density function (pdf) of the random variable Y given the random vector $\mathbf{X}$. It is showed by probability theory that

$$p_{Y|\mathbf{X}}(y|\mathbf{x}) = \frac{p_{\mathbf{X},Y}(\mathbf{x}, y)}{p_{\mathbf{X}}(\mathbf{x})}, \tag{1.5.2}$$

where generally $p_{\mathbf{X}}(\mathbf{x})$ is the pdf of the random vector $\mathbf{X}$ and $p_{\mathbf{X},Y}(\mathbf{x}, y)$ is the joint pdf of the random vector $\mathbf{X}$ and the variable Y. The combination of (1.5.1) and (1.5.2) gives

$$f(\mathbf{x}) = \int_{\mathbb{R}} y \frac{p_{\mathbf{X},Y}(\mathbf{x}, y)}{p_{\mathbf{X}}(\mathbf{x})}\mathrm{d}y = \frac{\int_{\mathbb{R}} y p_{\mathbf{X},Y}(\mathbf{x}, y)\mathrm{d}y}{p_{\mathbf{X}}(\mathbf{x})}, \tag{1.5.3}$$

where the joint pdf $p_{\mathbf{X},Y}(\mathbf{x}, y)$ and the pdf $p_{\mathbf{X}}(\mathbf{x})$ are both unknown.

Assume that the training sample $\{\mathbf{x}_i, y_i\}_{i=1,\ldots,N}$ is given, where $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$, and that $\{\mathbf{x}_i\}_{i=1,\ldots,N}$ is statistical independent and identically distributed. Define the *Parzen–Rosenblatt density estimator* of the pdf $p_{\mathbf{X}}(\mathbf{x})$ as

$$\hat{p}_{\mathbf{X}}(\mathbf{x}) = \frac{1}{Nh^m} \sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \qquad (\mathbf{x} \in \mathbb{R}^m), \tag{1.5.4}$$

where the smoothing parameter h, called *bandwidth*, is a positive number and controls the size of the kernel k. If h is a function of N such that $\lim_{N\to\infty} h(N) = 0$, then

$$\lim_{N\to\infty} E[\hat{p}_{\mathbf{X}}(\mathbf{x})] = p_{\mathbf{X}}(\mathbf{x}).$$

Similarly, define the *Parzen–Rosenblatt density estimator* of the joint pdf $p_{\mathbf{X},Y}(\mathbf{x}, y)$ as

$$\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y) = \frac{1}{Nh^{m+1}} \sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) k\left(\frac{y - y_i}{h}\right) \qquad (\mathbf{x} \in \mathbb{R}^m,\ y \in \mathbb{R}).$$

If h is a function of N such that $\lim_{N\to\infty} h(N) = 0$, then

$$\lim_{N\to\infty} E[\,\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y)\,] = p_{\mathbf{X},Y}(\mathbf{x}, y).$$

Integrating $y\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y)$ with respect to y and using $\int_{\mathbb{R}} k(\xi)\mathrm{d}\xi = 1$ and $\int_{\mathbb{R}} \xi k(\xi)\mathrm{d}\xi = 0$, it follows that

$$\int_{\mathbb{R}} y\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y)\mathrm{d}y = \frac{1}{Nh^{m+1}} \sum_{i=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \int_{\mathbb{R}} yk\left(\frac{y-y_i}{h}\right)\mathrm{d}y.$$

Let $\xi = \frac{y-y_i}{h}$. Then,

$$\begin{aligned}
&\textstyle\int_{\mathbb{R}} y\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y)\mathrm{d}y = \frac{1}{Nh^{m+1}} \sum_{i=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \int_{\mathbb{R}}(y_i + h\xi)k(\xi)h\mathrm{d}\xi \\
&= \textstyle\frac{1}{Nh^{m}} \sum_{i=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \left(y_i \int_{\mathbb{R}} k(\xi)\mathrm{d}\xi + h\int_{\mathbb{R}} \xi k(\xi)\mathrm{d}\xi\right) \\
&= \textstyle\frac{1}{Nh^{m}} \sum_{i=1}^{N} y_i k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right).
\end{aligned} \tag{1.5.5}$$

By (1.5.3), define the *Parzen–Rosenblatt density estimator* of the pdf $f_{\mathbf{X}}(\mathbf{x})$ as

$$\hat{f}(\mathbf{x}) = \frac{\int_{\mathbb{R}} y\hat{p}_{\mathbf{X},Y}(\mathbf{x}, y)\mathrm{d}y}{\hat{p}_{\mathbf{X}}(\mathbf{x})}$$

which is also called *kernel regression estimator*. By (1.5.4) and (1.5.5), the *kernel regression estimator* becomes

$$\hat{f}(\mathbf{x}) = \frac{\frac{1}{Nh^m}\sum_{i=1}^{N} y_i k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\frac{1}{Nh^m}\sum_{i=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)} = \frac{\sum_{i=1}^{N} y_i k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum_{j=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_j}{h}\right)}. \tag{1.5.6}$$

The kernel $k(\mathbf{x})$ is often required to be spherical symmetry, i.e.,

$$k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = k\left(\frac{\|\,\mathbf{x}-\mathbf{x}_i\,\|}{h}\right) \qquad (i = 1, \ldots, N),$$

where $\|\cdot\|$ is the Euclidean distance, and defined the normalized radial basis function as

$$\psi_N(\mathbf{x}, \mathbf{x}_i) = \frac{k\left(\frac{\|\mathbf{x}-\mathbf{x}_i\|}{h}\right)}{\sum_{j=1}^{N} k\left(\frac{\|\mathbf{x}-\mathbf{x}_j\|}{h}\right)} \qquad (i = 1, \ldots, N)$$

with $0 \leq \psi_i(\mathbf{x}) \leq 1$. The function $\psi_N(\mathbf{x}, \mathbf{x}_i)$ represents the probability of an event described by the input vector $\mathbf{x}$, conditional on $\mathbf{x}_i$. Let $w_i = y_i$ $(i = 1, \ldots, N)$. By (1.5.6), the kernel regression estimator becomes

$$\hat{f}(\mathbf{x}) = \frac{\sum\limits_{i=1}^{N} w_i k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum\limits_{j=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_j}{h}\right)} = \sum_{i=1}^{N} w_i \left(\frac{k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum\limits_{j=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_j}{h}\right)} \right) = \sum_{i=1}^{N} w_i \psi_N(\mathbf{x}, \mathbf{x}_i).$$

This equation represents the input–output mapping of a normalized radial basis function network.

The architecture of generalized regression network is a feedforward network with layered structure. For example, the architecture of a generalized regression network with an input layer, a single hidden regression layer, and output layer consisting of two computational units is described as follows

The input layer consists of m source nodes, where m is the dimensionality of the input vector $\mathbf{x} \in \mathbb{R}^m$. The single hidden regression layer consists of N computational units. Each computational unit is described by the normalized radial basis function:

$$\psi_N(\mathbf{x}, \mathbf{x}_i) = \frac{k\left(\frac{\|\mathbf{x}-\mathbf{x}_i\|}{\sigma}\right)}{\sum\limits_{j=1}^{N} k\left(\frac{\|\mathbf{x}-\mathbf{x}_j\|}{\sigma}\right)} \qquad (i = 1, \ldots, N),$$

where the kernel function k is the Gaussian distribution:

$$k(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{m}{2}}} \exp\left(-\frac{\| \mathbf{x} \|}{2}\right).$$

So the corresponding kernel function with the center $\mathbf{x}_i \in \mathbb{R}^{m_0}$ and a common bandwidth σ is

$$k\left(\frac{\| \mathbf{x} - \mathbf{x}_i \|}{\sigma}\right) = \frac{1}{(2\pi\sigma^2)^{\frac{m}{2}}} \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}_i \|}{2\sigma^2}\right) \qquad (i = 1, \ldots, N),$$

Each computational unit is

$$\psi_N(\mathbf{x}, \mathbf{x}_i) = \frac{\frac{1}{(2\pi\sigma^2)^{\frac{m}{2}}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|}{2\sigma^2}\right)}{\sum\limits_{j=1}^{N} \frac{1}{(2\pi\sigma^2)^{\frac{m}{2}}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_j\|}{2\sigma^2}\right)} = \frac{\exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|}{2\sigma^2}\right)}{\sum\limits_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_j\|}{2\sigma^2}\right)} \qquad (i = 1, \ldots, N).$$

The links between source nodes and hidden units are direct connections with no weights. The output layer consists of two computational units:

$$\sum_{i=1}^{N} w_i \exp(-\tfrac{\|\mathbf{x}-\mathbf{x}_i\|}{2\sigma^2}),$$
$$\sum_{j=1}^{N} \exp(-\tfrac{\|\mathbf{x}-\mathbf{x}_j\|}{2\sigma^2}).$$

The size of the output layer is 2. The approximating function is

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{N} w_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|}{2\sigma^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_j\|}{2\sigma^2}\right)}$$

1.6 Self-organizing Network

The aims of the self-organizing network are to simplify and speed up the planning, configuration, management, and optimization of huge amount of data. It consists of an input layer and a competitive layer. Each neuron node in the competitive layer connects all source nodes in the input layer, and the neuron nodes in the competitive layer connect each other. The main advantages of self-organizing networks lie in that they can vary parameters and structure of networks self-organizationally and adaptively and find out the inherent rule of the sample data automatically. The task of the input layer is to receive external information and transfer the input patterns to the competitive layer, while the task of the competitive layer is to find out inherent rules of patterns and then classify them. The self-organizing networks can be divided into the Kohonen self-organizing map network, the learning vector quantization network, and so on.

1.6.1 Kohonen Self-organizing Map Network

The self-organizing map (SOM) network was introduced by Kohonen, so one always calls it Kohonen self-organizing map network. A SOM network consists of an input layer and a competitive layer, where the source nodes of the input layer transfer external information to every neuron in the competitive layer through synaptic-weight vector. When the signals enter the input layer, the network transforms adaptively the incoming signals of arbitrary dimension into a one-dimensional or two-dimensional discrete map in a topologically ordered fashion. The output neurons of the network compete among themselves to be activated. Only one output neuron is activated at any one time. This neuron is called a *winning neuron*. Such competition can be implemented by having lateral inhibition connections (negative feedback) between the neurons. The final result is outputted from the winning neuron of the competitive layer. The SOM network is often viewed as a nonlinear generalization of principal component analysis (see Chap. 4).

Competitive process is an essential process in the self-organizing network. This is based on the best match of the input vector with the synaptic-weight vector as follows:

Let $\mathbf{x} = (x_1, \dots, x_m)^T$ be an input signal, here m is the dimension of the input space and $\| \mathbf{x} \| = \sqrt{\sum_{j=1}^m x_j^2}$, and let $\mathbf{w}_i = (w_{i1}, \dots, w_{im})^T$ $(i = 1, \dots, k)$ be the synaptic-weight vector, where k is the total number of neurons in the network. Correspondingly, the unit vectors are

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\| \mathbf{x} \|} = \left(\frac{x_1}{\sqrt{\sum\limits_{j=1}^m x_j^2}}, \dots, \frac{x_m}{\sqrt{\sum\limits_{j=1}^m x_j^2}} \right)^T =: (\hat{x}_1, \dots, \hat{x}_m)^T,$$

$$\hat{\mathbf{w}}_i = \frac{\mathbf{w}_i}{\| \mathbf{w}_i \|} = \left(\frac{w_{i1}}{\sqrt{\sum\limits_{j=1}^m w_{ij}^2}}, \dots, \frac{w_{im}}{\sqrt{\sum\limits_{j=1}^m w_{ij}^2}} \right)^T =: (\hat{w}_{i1}, \dots, \hat{w}_{im})^T \quad (i = 1, \dots, k).$$

It is clear that

$$\| \hat{\mathbf{x}} \| = \sum_{i=1}^m \hat{x}_i^2 = \sum_{i=1}^m \left(\frac{x_i}{\sqrt{\sum\limits_{j=1}^m x_j^2}} \right)^2 = \frac{\sum\limits_{i=1}^m x_i^2}{\sum\limits_{j=1}^m x_j^2} = 1,$$

$$\| \hat{\mathbf{w}}_i \| = \sum_{l=1}^m \hat{w}_{il}^2 = \sum_{l=1}^m \left(\frac{w_{il}}{\sqrt{\sum\limits_{j=1}^m w_{ij}^2}} \right)^2 = \frac{\sum\limits_{l=1}^m w_{il}^2}{\sum\limits_{j=1}^m w_{ij}^2} = 1 \qquad (i = 1, \dots, k).$$

A commonly used criterion is the Euclidean distance between $\hat{\mathbf{x}}$ and $\hat{\mathbf{w}}_i$:

$$\| \hat{\mathbf{x}} - \hat{\mathbf{w}}_i \| = \sqrt{(\hat{\mathbf{x}} - \hat{\mathbf{w}}_i)^T (\hat{\mathbf{x}} - \hat{\mathbf{w}}_i)} \qquad (i = 1, \dots, k). \tag{1.6.1}$$

The neuron $i_{\mathbf{x}}^*$ satisfying the condition

$$\| \hat{\mathbf{x}} - \hat{\mathbf{w}}_{i_{\mathbf{x}}^*} \| = \min_{i=1,\dots,k} \| \hat{\mathbf{x}} - \hat{\mathbf{w}}_i \| \tag{1.6.2}$$

is the *winning neuron* for the input vector $\mathbf{x}$. Note that

$$\begin{aligned}
&\hat{\mathbf{x}}^T\hat{\mathbf{x}} = \hat{x}_1^2 + \cdots + \hat{x}_m^2 = \| \hat{\mathbf{x}} \|^2 = 1,\\
&\hat{\mathbf{w}}_i^T\hat{\mathbf{w}}_i = \hat{w}_{i1}^2 + \cdots + \hat{w}_{im}^2 = \| \hat{\mathbf{w}}_i \|^2 = 1 \ (i = 1, \ldots, k),\\
&\hat{\mathbf{x}}^T\hat{\mathbf{w}}_i = \hat{x}_1\hat{w}_{i1} + \cdots + \hat{x}_m\hat{w}_{im} = \hat{\mathbf{w}}_i^T\hat{\mathbf{x}} \quad (i = 1, \ldots, k).
\end{aligned}$$

Then, the product $(\hat{\mathbf{x}} - \hat{\mathbf{w}}_i)^T(\hat{\mathbf{x}} - \hat{\mathbf{w}}_i)$ in (1.6.1) becomes

$$\begin{aligned}
&(\hat{\mathbf{x}} - \hat{\mathbf{w}}_i)^T(\hat{\mathbf{x}} - \hat{\mathbf{w}}_i) = (\hat{\mathbf{x}}^T - \hat{\mathbf{w}}_i^T)(\hat{\mathbf{x}} - \hat{\mathbf{w}}_i)\\
&= \hat{\mathbf{x}}^T\hat{\mathbf{x}} - \hat{\mathbf{w}}_i^T\hat{\mathbf{x}} - \hat{\mathbf{x}}^T\hat{\mathbf{w}}_i + \hat{\mathbf{w}}_i^T\hat{\mathbf{w}}_i = 2(1 - \hat{\mathbf{w}}_i^T\hat{\mathbf{x}}) \qquad (i = 1, \ldots, k).
\end{aligned}$$

From this and (1.6.1), the Euclidean distance between $\hat{\mathbf{x}}$ and $\hat{\mathbf{w}}_i$ becomes

$$\| \hat{\mathbf{x}} - \hat{\mathbf{w}}_i \| = \sqrt{2(1 - \hat{\mathbf{w}}_i^T\hat{\mathbf{x}})} \qquad (i = 1, \ldots, k),$$

where $\hat{\mathbf{w}}_i^T\hat{\mathbf{x}}$ is the inner product of $\hat{\mathbf{w}}_i$ and $\hat{\mathbf{x}}$. From this, it is seen that the minimization of the Euclidean distance $d(\hat{\mathbf{x}}, \hat{\mathbf{w}}_i)$ corresponds to the maximization of the inner product $\hat{\mathbf{w}}_i^T\hat{\mathbf{x}}$. Therefore, by (1.6.2), the neuron $i_{\mathbf{x}}^*$ satisfying the condition

$$\hat{\mathbf{w}}_{i_{\mathbf{x}}^*}^T\hat{\mathbf{x}} = \max_{i=1,\ldots,k}\left(\hat{\mathbf{w}}_i^T\hat{\mathbf{x}}\right) \tag{1.6.3}$$

is the winning neuron for the input vector $\mathbf{x}$.

Let ψ_i be the angle between the unit synaptic-weight vector $\hat{\mathbf{w}}_i^T$ and the unit input vector $\hat{\mathbf{x}}$. Note that $\| \hat{\mathbf{w}}_i^T \| = 1$ and $\| \hat{\mathbf{x}} \| = 1$. So the inner product $\hat{\mathbf{w}}_i^T\hat{\mathbf{x}}$ is represented as

$$\hat{\mathbf{w}}_i^T\hat{\mathbf{x}} = \| \hat{\mathbf{w}}_i^T \| \| \hat{\mathbf{x}} \| \cos\psi_i = \cos\psi_i \qquad (i = 1, \ldots, k).$$

Therefore, the maximization of the inner product $\hat{\mathbf{w}}_i^T\hat{\mathbf{x}}$ corresponds to the minimization of the angle ψ_i. Furthermore, by (1.6.3), the neuron $i_{\mathbf{x}}^*$ satisfying the condition

$$\psi_{i_{\mathbf{x}}^*} = \min_{i=1,\ldots,k}\psi_i$$

is the winning neuron for the input vector $\mathbf{x}$.

The topological neighborhood is an important concept in the self-organizing network. When one neuron is activated, its closest neighbors tend to get excited more than those further away. Let neuron j be a typical one of the set of excited neurons, and let $d_{ji_{\mathbf{x}}^*}$ be the lateral distance between the winning neuron $i_{\mathbf{x}}^*$ and the excited neuron j. Main topological neighborhoods $h_{ji_{\mathbf{x}}^*}$ are stated as follows:

(a) Gaussian topological neighborhood is

$$h_{ji_{\mathbf{x}}^*}^{(G)} = \exp\left(-\frac{d_{ji_{\mathbf{x}}^*}^2}{2\sigma^2}\right).$$

Gaussian topological neighborhood is symmetric about 0, and it attains the maximum value 1 when $d_{ji_{\mathbf{x}}^*} = 0$; in other words, it attains the maximum at the winning neuron $i_{\mathbf{x}}^*$ when $d_{ji_{\mathbf{x}}^*} = 0$. The amplitude of Gaussian topological neighborhood decreases monotonically when $d_{ji_{\mathbf{x}}^*}$ increases, and it decays to zero when $d_{ji_{\mathbf{x}}^*}$ tends to infinity. Gaussian topological neighborhood makes the computation of the network converge quick. The parameter σ in a Gaussian topological neighborhood is time varying. As the parameter shrinks with time, the size of the topological neighborhood shrinks with time.

(b) Mexican-hat topological neighborhood is

$$h_{ji_{\mathbf{x}}^*}^{(M)} = \left(1 - \frac{d_{ji_{\mathbf{x}}^*}^2}{2\sigma^2}\right) \exp\left(-\frac{d_{ji_{\mathbf{x}}^*}^2}{2\sigma^2}\right).$$

Mexican-hat topological neighborhood is symmetric about 0 and attains the maximum 1 when $d_{ji_{\mathbf{x}}^*} = 0$. Its amplitude is monotonically positive when $d_{ji_{\mathbf{x}}^*}$ increases from 0 to $\sqrt{2}\sigma$ and is monotonically negative and tends to zero when $d_{ji_{\mathbf{x}}^*}$ increases from $\sqrt{2}\sigma$ to infinity. In application, due to complex computation, one uses often its simple forms, i.e., the big-hat topological neighborhood or the rectangular topological neighborhood.

(c) Big-hat topological neighborhood is

$$h_{ji_{\mathbf{x}}^*}^{(B)} = \begin{cases} -\frac{1}{3}, & -\sqrt{2}\sigma - \frac{1}{3} < d_{ji_{\mathbf{x}}^*} < -\sqrt{2}\sigma, \\ +1, & -\sqrt{2}\sigma < d_{ji_{\mathbf{x}}^*} < \sqrt{2}\sigma, \\ -\frac{1}{3}, & \sqrt{2}\sigma < d_{ji_{\mathbf{x}}^*} < \sqrt{2}\sigma + \frac{1}{3}, \\ 0, & \text{otherwise.} \end{cases}$$

Big-hat topological neighborhood is simpler than the Mexican-hat topological neighborhood. Big-hat topological neighborhood is symmetric about 0 and attains the maximum value 1 when $d_{ji_{\mathbf{x}}^*} = 0$. Its amplitude is 1 when $d_{ji_{\mathbf{x}}^*}$ increases from 0 to $\sqrt{2}\sigma$ and is $-\frac{1}{3}$ when $d_{ji_{\mathbf{x}}^*}$ increases from $\sqrt{2}\sigma$ to $\sqrt{2}\sigma + \frac{1}{3}$.

(d) Rectangular topological neighborhood is

$$h_{ji_{\mathbf{x}}^*}^{(R)} = \begin{cases} +1, & -\sqrt{2}\sigma < d_{ji_{\mathbf{x}}^*} < \sqrt{2}\sigma, \\ 0, & \text{otherwise.} \end{cases}$$

Rectangular topological neighborhood is simpler than the big-hat topological neighborhood or the Mexican-hat topological neighborhood. Rectangular topological neighborhood is symmetric about 0 and attains the maximum value 1 when $d_{ji_{\mathbf{x}}^*} = 0$. Its amplitude is 1 when $d_{ji_{\mathbf{x}}^*}$ increases from 0 to $\sqrt{2}\sigma$.

Finally, the synaptic-weight vectors of all excited neurons are adjusted by the updated formula:

$$\Delta \mathbf{w}_j = \eta h_{ji^*_{\mathbf{x}}}(\mathbf{x} - \mathbf{w}_j),$$

where $i^*_{\mathbf{x}}$ is the winning neuron, j is the excited neuron, η is time varying, and $\Delta \mathbf{w}_j$ is the change of the weight vector of neuron j. Based on these new weights, the new round of competitive process will be done.

1.6.2 Learning Vector Quantization Network

Learning vector quantization (LVQ) network is proposed based on the architecture of the self-organizing competitive network. A single hidden layer, called *Kohonen hidden layer*, is added to the self-organizing network. The LVQ network consists of an input layer, a Kohonen hidden layer, and an output layer. The network is a feedforward network. Source nodes in the input layer and the neuron nodes in Kohonen hidden layer are fully connected. Neuron nodes in the hidden layer are clustered. Neuron nodes in the output layer connect to different clusters of neuron nodes in the hidden layer.

The Kohonen hidden layer uses competitive learning to perform the cluster of patterns. Neurons in the Kohonen hidden layer compete among themselves to yield the only wining neuron. The competitive criterion is the Euclidean distance between $\hat{\mathbf{x}}$ and $\hat{\mathbf{w}}_i$:

$$\| \hat{\mathbf{x}} - \hat{\mathbf{w}}_i \| = \sqrt{\sum_{j=1}^{m} (\hat{x}_j - \hat{w}_{ji})^2} \qquad (i = 1, \dots, k),$$

where $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_m)$ is the unit vector of the input sample and $\hat{\mathbf{w}}_i = (\hat{w}_{1i}, \dots, \hat{w}_{mi})$ is the unit synaptic-weight vector. The winning neuron $i^*_{\mathbf{x}}$ satisfies the condition:

$$\| \hat{\mathbf{x}} - \hat{\mathbf{w}}_{i^*_{\mathbf{x}}} \| = \min_{i=1,\dots,k} \| \hat{\mathbf{x}} - \hat{\mathbf{w}}_i \| .$$

The output of the winning neuron $i^*_{\mathbf{x}}$ is 1, and the outputs of other neurons are all 0.

The output layer uses vector quantization learning to perform the pattern's classification. Vector quantization is based on an encoding–decoding process as follows:

Let $c(\mathbf{x})$ act as an encoder of the input vector $\mathbf{x}$ and $\mathbf{x}^*(c)$ act as a decoder of $c(\mathbf{x})$. The *expected distortion D* between the input vector $\mathbf{x}$ and its reconstruction vector $\mathbf{x}^* = \mathbf{x}^*(c(\mathbf{x}))$ is defined by

$$D = \frac{1}{2} \int_{-\infty}^{\infty} p_{\mathbf{x}}(\mathbf{x}) d(\mathbf{x}, \mathbf{x}^*) \mathrm{d}\mathbf{x},$$

where $p_{\mathbf{x}}(\mathbf{x})$ is a probability density function and $d(\mathbf{x}, \mathbf{x}^*)$ is a distortion measure. A distortion measure used commonly is the Euclidean distance, i.e.,

$$d(\mathbf{x}, \mathbf{x}^*) = \| \mathbf{x} - \mathbf{x}^* \| .$$

The optimum encoding–decoding scheme is determined by varying $c(\mathbf{x})$ and $\mathbf{x}^*(c)$ so as to minimize the expected distortion. The generalized Lloyd algorithm shows two necessary conditions for minimization of the expected distortion as follows:

(a) Given the input vector $\mathbf{x}$, choose the code $c = c(\mathbf{x})$ to minimize $d(\mathbf{x}, \mathbf{x}^*(c))$;
(b) Given the code c, compute the vector $\mathbf{x}^* = \mathbf{x}^*(c)$ as the centroid of input vectors $\mathbf{x}$ satisfying (a).

To implement vector quantization, the algorithm alternately optimizes the encoder $c(\mathbf{x})$ in accordance with (a) and then optimizes the decoder $\mathbf{x}^*(c)$ in accordance with (b) until the expected distortion D reaches a minimum.

In the output layer, neurons are prescribed which class they belong to. The synaptic weights connecting the hidden nodes to the output nodes are adjusted iteratively in a step-by-step fashion. If the winning neuron is in the prescribed class, the synaptic-weight vectors of all excited neurons are adjusted by the updated formula:

$$\Delta \mathbf{w}_i = +\eta(\mathbf{x} - \mathbf{w}_i) \qquad (i = 1, \ldots, k).$$

If not, they are adjusted by the updated formula:

$$\Delta \mathbf{w}_i = -\eta(\mathbf{x} - \mathbf{w}_i) \qquad (i = 1, \ldots, k),$$

where η is time varying.

1.7 Hopfield Network

Hopfield neural network, or simply *Hopfield network*, is a recurrent network with a single layer of neurons. Its activation functions may be continuous or discrete functions, so Hopfield networks are classified into continuous and discrete Hopfield networks.

1.7.1 Continuous Hopfield Network

A continuous Hopfield network consists of a set of fully connected neurons. The number of its feedback loops is equal to the number of neurons. The synaptic weights w_{ij} between neuron i and neuron j are symmetric, i.e., $w_{ij} = w_{ji}$ for all i, j; the adder of every neuron is constituted by a computing amplifier and a corresponding circuit; the work fashions of neurons are synchronous parallel; and the input and the output are both simulating quantities.

(a) Dynamics

Let $v_i(t)$ be the given induced local field. Assume that the activation function of the continuous Hopfield network is the hyperbolic tangent function:

$$x = \varphi_i(v) = \tanh\left(\frac{a_i v}{2}\right) = \frac{1-\exp(-a_i v)}{1+\exp(-a_i v)} \qquad (a_i \in \mathbb{Z}_+), \tag{1.7.1}$$

where the a_i is referred to as the *gain* of neuron i. It is clear by (1.7.1) that $|x| < 1$ and

$$\left.\frac{d\varphi_i}{dv}\right|_{v=0} = \frac{a_i}{2}.$$

So the slope of the activation function at the origin is $\frac{a_i}{2}$. By (1.7.1), it follows that

$$x + x\exp(-a_i v) = 1 - \exp(-a_i v),$$

So the inverse activation function is

$$v = \varphi_i^{-1}(x) = -\frac{1}{a_i}\log\left(\frac{1-x}{1+x}\right) \qquad (a_i \in \mathbb{Z}_+),$$

Since $|x| < 1$, the derivative of the inverse activation function is

$$\frac{d}{dx}\varphi_i^{-1}(x) = \frac{2}{a_i(1-x^2)} > 0.$$

The inverse activation function $\varphi_i^{-1}(x)$ is a monotonically increasing function of x.

If the continuous Hopfield network is of interconnection of N neurons, the dynamics of this network are defined by the system of differential equations:

$$C_j \frac{d}{dt} v_j(t) + \frac{v_j(t)}{R_j} = \sum_{i=1}^{N} w_{ij}\varphi_i(v_i(t)) + I_j \qquad (j = 1, \ldots, N), \tag{1.7.2}$$

where C_j is the leakage capacitance, R_j is the leakage resistance, w_{ij} represents conductance and is the symmetric synaptic weights: $w_{ij} = w_{ji}$ for all $i,\ j$, $v_i(t)$ is the given induced local field, and I_j is the externally applied bias.

(b) Energy Function

The dynamics of continuous Hopfield network are to seek out the minima of its *energy function* which is defined as

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_i x_j + \sum_{j=1}^{N}\frac{1}{R_j}\int_0^{x_j}\varphi_i^{-1}(x)\mathrm{d}x - \sum_{j=1}^{N} I_j x_j, \tag{1.7.3}$$

where x_j is the output of the jth neuron, $w_{ij} = w_{ji}$, I_j is the externally applied bias, R_j is the leakage resistance, and

$$\begin{aligned} x_i &= \varphi_i(v_i(t)) = \tfrac{1-\exp(-a_i v_i(t))}{1+\exp(-a_i v_i(t))}, \\ \varphi_i^{-1}(x) &= -\tfrac{1}{a_i} \log\left(\tfrac{1-x}{1+x}\right). \end{aligned}$$

In order to find the minima of the energy function, differentiating the energy function (1.7.3) with respect to the time t, it follows by (1.7.2) and that

$$\frac{dE}{dt} = -\sum_{j=1}^{N}\left(\sum_{i=1}^{N} w_{ij}x_i - \frac{\varphi_i^{-1}(x_j)}{R_j} + I_j\right)\frac{dx_j}{dt}.$$

Note that $x_j = \varphi_i(v_j(t))$ or $\varphi_i^{-1}(x_j) = v_j$. By (1.7.2), it follows that

$$\frac{dE}{dt} = -\sum_{j=1}^{N}\left(\sum_{i=1}^{N} w_{ij}\varphi_i(v_i) - \frac{v_j}{R_j} + I_j\right)\frac{dx_j}{dt} = -\sum_{j=1}^{N} C_j \frac{d}{dt}v_j(t)\frac{dx_j}{dt}.$$

Note that $v_j(t) = \varphi_j^{-1}(x_j)$. So

$$\frac{d}{dt}v_j(t) = \frac{d}{dx_j}\varphi_j^{-1}(x_j)\frac{dx_j}{dt},$$

and so

$$\frac{dE}{dt} = -\sum_{j=1}^{N} C_j \frac{d}{dx_j}\varphi_j^{-1}(x_j)\left(\frac{dx_j}{dt}\right)^2.$$

Note that

$$\left(\frac{dx_j}{dt}\right)^2 \geq 0, \qquad \frac{d}{dx_j}\varphi_j^{-1}(x_j) \geq 0$$

since $\varphi_j^{-1}(x_j)$ is a monotonically increasing function of x_j. Therefore, for all time t,

$$\frac{dE}{dt} \leq 0. \tag{1.7.4}$$

So the energy function of continuous Hopfield network is a monotonically decreasing function of time. If $\frac{dx_j}{dt} = 0$ for all time t, then $\frac{dE}{dt} = 0$. From this and (1.7.4), $\frac{dE}{dt} < 0$ except at the fixed point which is also called the *attractor*. Therefore, the continuous Hopfield network is asymptotically stable in the Lyapunov sense and the fixed points are the minima of the energy function, and vice versa.

1.7.2 Discrete Hopfield Network

A discrete Hopfield network consists of interconnection of a set of neurons with corresponding unit time delays. The number of feedback loops is equal to the number of neurons. The synaptic weights w_{ij} between neuron i and neuron j are symmetric, i.e., $w_{ij} = w_{ji}$. There is no self-feedback in the network, i.e., $w_{ii} = 0$. This means that the output of each neuron is fed back by a unit-time delay element to each of the other neurons in the network.

(a) Energy Function

For the discrete Hopfield network consisting of interconnection of N neurons, the state of the network is defined by $\mathbf{x} = (x_1, \dots, x_N)^T$. The synaptic weight matrix $W = (w_{ij})_{i,j=1,\dots,N}$ is symmetric: $w_{ij} = w_{ji}$ for all i, j and $w_{ii} = 0$ for all i, so the induced local field v_i of neuron i is

$$v_i = \sum_{j=1}^{N} w_{ij}x_j + b_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} w_{ij}x_j + b_i \qquad (i = 1, \dots, N),$$

where b_i is the externally applied bias to neuron i, and the state x_i of neuron i is modified by

$$x_i = \operatorname{sgn}(v_i) = \begin{cases} 1 & \text{if } v_i > 0 \\ -1 & \text{if } v_i < 0 \end{cases}$$

If $v_i = 0$, then the common convention is the neuron i remains in its previous state. The energy function of the discrete Hopfield network is defined as

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N} w_{ij}x_i x_j - \sum_{i=1}^{N} b_i x_i, \tag{1.7.5}$$

Let $\mathbf{b} = (b_1, \dots, b_N)^T$. Since

$$W = \begin{pmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & 0 & w_{23} & \cdots & w_{2N} \\ w_{31} & w_{32} & 0 & \cdots & w_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & w_{N3} & \cdots & 0 \end{pmatrix},$$

it is clear that

$$\mathbf{x}^T W\mathbf{x} = (x_1, \dots, x_N) \begin{pmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & 0 & w_{23} & \cdots & w_{2N} \\ w_{31} & w_{32} & 0 & \cdots & w_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & w_{N3} & \cdots & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix}$$

$$= \left(\sum_{\substack{j=1 \\ j\neq 1}}^{N} w_{j1}x_j, \ \sum_{\substack{j=1 \\ j\neq 2}}^{N} w_{j2}x_j, \dots, \ \sum_{\substack{j=1 \\ j\neq N}}^{N} w_{jN}x_j \right) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

$$= \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j\neq i}}^{N} w_{ji}x_jx_i$$

and

$$\mathbf{x}^T\mathbf{b} = (x_1, \dots, x_N) \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix} = \sum_{i=1}^{N} b_i x_i$$

From this and (1.7.5), the energy function of the discrete Hopfield network can be rewritten as

$$E = -\frac{1}{2}\mathbf{x}^T W\mathbf{x} - \mathbf{x}^T\mathbf{b}.$$

The energy of the discrete Hopfield network varies with the state of any neuron of the network. When the state of neuron m of the discrete Hopfield network varies from -1 to 1 due to $v_m > 0$, the corresponding energy of the network varies from the original energy E_1 to the latter energy E_2. Since the synaptic weight matrix is symmetric, by (1.7.5), the original energy E_1 is

$$\begin{aligned} E_1 &= -\tfrac{1}{2} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j\neq i}}^{N} w_{ij}x_ix_j - \sum_{i=1}^{N} b_ix_i \\ &= -\tfrac{1}{2} \sum_{\substack{i=1 \\ i\neq m}}^{N} \sum_{\substack{j=1 \\ j\neq i,m}}^{N} w_{ij}x_ix_j - \sum_{\substack{i=1 \\ i\neq m}}^{N} b_ix_i - \sum_{\substack{j=1 \\ j\neq m}}^{N} w_{mj}x_mx_j - b_mx_m. \end{aligned}$$

Note that the original state of the neuron m is -1, i.e., $x_m = -1$. Then, the original energy is

$$E_1 = -\frac{1}{2} \sum_{\substack{i=1 \\ i\neq m}}^{N} \sum_{\substack{j=1 \\ j\neq i,m}}^{N} w_{ij}x_ix_j - \sum_{\substack{i=1 \\ i\neq m}}^{N} b_ix_i + \sum_{\substack{j=1 \\ j\neq m}}^{N} w_{mj}x_j + b_m \tag{1.7.6}$$

and the latter energy E_2 is

$$E_2 = -\frac{1}{2}\sum_{\substack{i=1\\ i\neq m}}^{N}\sum_{\substack{j=1\\ j\neq i,m}}^{N} w_{ij}x_i x_j - \sum_{\substack{i=1\\ i\neq m}}^{N} b_i x_i - \sum_{\substack{j=1\\ j\neq m}}^{N} w_{mj}x_j - b_m.$$

Combining this with (1.7.6), the difference of the energy is

$$\Delta E = E_2 - E_1 = -2\left(\sum_{\substack{j=1\\ j\neq m}}^{N} w_{mj}x_j + b_m\right) = -2v_m.$$

Due to $v_m > 0$, it is clear that $\Delta E < 0$. It means that when the state of any neuron of the discrete Hopfield network increases from -1 to 1, the energy of the network decreases. Similarly, when the state of any neuron of the discrete Hopfield network decreases from 1 to -1 due to $v_m < 0$, the energy of the network increases.

(b) A Content-Addressable Memory

Discrete Hopfield network is applied as a content-addressable memory. The operation includes two phases: the storage phase and the retrieval phase.

- Storage Phase

Let M vectors $\xi_1, \xi_2, \ldots, \xi_M$ represent a known set of N-dimensional fundamental memories. The synaptic weights from neuron i to neuron j for the discrete Hopfield network are defined as

$$w_{ij} = \begin{cases} \frac{1}{N}\sum\limits_{\mu=1}^{M} \xi_{\mu i}\xi_{\mu j} & (i \neq j), \\ 0 & (i = j), \end{cases}$$

where

$$\xi_\mu = (\xi_{\mu 1}, \ldots, \xi_{\mu N})^T, \qquad \xi_{\mu i} = \pm 1 \qquad (i = 1, \ldots, N).$$

The matrix form of the synaptic weight matrix of the network is

$$W = \frac{1}{N}\sum_{\mu=1}^{M} \xi_\mu \xi_\mu^T - \frac{M}{N} I,$$

where I is the identity matrix and ξ_μ is stated as above, and the synaptic weight matrix is an $N \times N$ symmetric matrix: $W = W^T$.

- Retrieval Phase

Starting from an $N-$dimensional vector termed a probe, the asynchronous updating procedure is performed until the discrete Hopfield network produces a time-invariant state vector $\mathbf{y}$ which satisfies the *stability condition* $\mathbf{y} = \operatorname{sgn}(W\mathbf{y} + \mathbf{b})$ which is also called the *alignment condition*, where W is synaptic weight matrix of discrete Hopfield network and $\mathbf{b}$ is the externally applied bias vector. The state vector $\mathbf{y}$ that satisfies the alignment condition is called a *stable state*. Hence, the discrete Hopfield network always converges to a stable state.

Further Reading

H.Z. Abyaneh, M.B. Varkeshi, G. Golmohammadi, K. Mohammadi, Soil temperature estimation using an artificial neural network and co-active neuro-fuzzy inference system in two differen climates. Arab. J. Geosci. **9**, 377 (2016)

M.A. Amiri, Y. Amerian, M.S. Mesgari, Spatial and temporal monthly precipitation forecasting using wavelet transform and neural networks, Qara-Qum catchment, Iran. Arab. J. Geosci. **9**, 421 (2016)

P.D. Brooks, D. McKnight, K. Elder, Carbon limitation of soil respiration under winter snowpacks: potential feedbacks between growing season and winter carbon fluxes. Glob. Change Biol. **11**, 231–238 (2004)

A. Gersho, R.M. Gray, *Vector Quantization and Signal Compression* (Kluwer, Norwell, MA, 1992)

R.L. Hardy, Multiquadric equations of topography and other irregular surfaces. J. Geophys. Res. **76**, 1905–1915 (1971)

S. Haykin, *Neural Networks and Learning Machines*, 3rd edn. (New York, Pearson Education, 2008)

J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. U.S.A. **79**, 2554–2558 (1982)

J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Natl. Acad. Sci. U.S.A. **81**, 3088–3092 (1984)

J.J. Hopfield, Neurons, dynamics and computation. Phys. Today **47**, 40–46 (1994)

T. Kohonen, Self-organized formation of topologically correct feature maps. Biol. Cybern. **43**, 59–69 (1982)

T. Kohonen, The self-organizing map. Proc. Inst. Electr. Electron. Eng. **78**, 1464–1480 (1990)

T. Kohonen, *Self-organizing Maps*, 2nd edn. (Springer, Berlin, 1997)

A. Krzyzak, T. Linder, G. Lugosi, Nonparametric estimation and classification using radial basis functions. IEEE Trans. Neural Networks **7**, 475–487 (1996)

S.P. Luttrell, *Self-organization: A Derivation from First Principle of a Class of Learning Algorithms* (IEEE Conference on Neural Networks, Washington, DC, 1989)

C.A. Micchelli, Interpolation of scattered data: distance matrices and conditionally positive definite function. Constr. Approx. **2**, 11–22 (1986)

E.A. Nadaraya, On estimating regression. Theory Probab. Appl. **9**, 141–142 (1964)

K. Obermayer, H. Ritter, K. Schulten, Development and spatial structure of cortical feature maps: a model study, *Advances in Neural Information Proceeding Systems* (Morgan Kaufmann, San Mateo, CA, 1991), pp. 11–17

M. Ozturk, O. Salman, M. Koc, Artificial neural network model for estimating the soil temperature. Can. J. Soil Sci. **91**, 551–562 (2011)

E. Parzen, On estimation of a probability density function and mode. Ann. Math. Stat. **23**, 1065–1076 (1962)

H. Ritter, T. Martinetz, K. Schulten, *Neural Computation and Self-organizing Maps: An Introduction* (Addison-Wesley, Reading, MA, 1992)

M. Rosenblatt, Remarks on some nonparametric estimates of a density function. Ann. Math. Stat. **27**, 832–837 (1956)

M. Rosenblatt, Density estimates and Morkov sequences, in *Nonparametric Techniques in Statistical Inference*, ed. by M. Puri (Cambridge University Press, London, 1970)

A.J. Tenge, F.B.S. Kaihura, R. Lal, B.R. Singh, Diurnal soil temperature fluctuations for different erosional classes of an oxisol at Mlingano, Tanzania. Soil Tillage Res. **49**, 211–217 (1998)

D.S. Tourtzky, D.A. Pomerleau, What is hidden in the hidden layers? Byte **14**, 227–233 (1989)

H.L. Van Trees, *Trees, Detection, Estimation, and Modulation Theory, Part I* (Wiley, New York, 1968)

G.S. Watson, Smooth regression analysis. Sankhy*ā* Indian J. Stat. Ser. A **26**, 359–372 (1964)

L. Xu, A. Krzyzak, A. Yuille, On radial basis function nets and kernel regression: statistical consistency, convergency rates, and receptive field size. Neural Netw. **7**, 609–628 (1994)