14

Neural Network-based Clustering

14.1 Introduction

Clustering in the neural network literature is generally based on competitive learning (CL) model, which originated in the early 1970s through contributions of Christoph von der Malsburg (Malsburg, 1973). CL, virtually, is a neural network learning process where different neurons or processing elements compete on who is allowed to learn to represent the current input. In a CL model, a stream of input patterns to a network F_1 can train the adaptive weights that multiply the signals in the pathways from F_1 to a coding level F_2. Level F_2 is designed as a competitive network capable of choosing the node (or nodes) which receive(s) the largest total input. The winning population then triggers associative pattern learning to update the adaptive weights.

Clustering algorithms implemented by distortion-based CL techniques commonly have the prototypes corresponding to the weights of neurons, for example, the centre of their receptive field in the input feature space. A common feature of the CL clustering algorithms is a competitive stage which precedes each learning step and decides to what extent a neuron may adapt its weights to a new input pattern. The goal of CL is the minimisation of the distortion or quantisation error in vector quantisation.

Kohonen made particularly strong implementation of CL in his work on learning vector quantisation (LVQ) and self-organising maps (SOM) – also known as self-organising feature maps (SOFM) (Kohonen, 1990). Intrinsically, LVQ performs supervised learning, and is not categorised as a clustering algorithm. Nevertheless, its learning properties provide an insight to describe the potential data structure using the prototype vectors in the competitive layer. It inspires many other CL clustering algorithms, such as generalised LVQ (GLVQ) by Pal, Bezdek and Tsao (1993), Neural-gas network by Martinetz, Berkovich and Schulten (1993), and so on. SOM is one of the most popular clustering algorithms, ideally suited to exploratory data analysis, allowing one to impose partial structure on the clusters and facilitating easy visualisation and interpretation. However, SOM also suffers the problems caused by a number of user-predefined parameters, for example the size of the lattice, the number of clusters, and so on. Additionally, trained SOM may suffer from input space density misrepresentation, where areas of low pattern density may be over-represented and areas of high density under-represented (Kohonen, 2001).

Adaptive resonance theory (ART) is another learning model growing out of the CL model, which was first introduced by Grossberg (1980) and developed by Carpenter and Grossberg (1987a, b, 1988, 1990; Carpenter, Grossberg and Reynolds, 1991; Carpenter, Grossberg and Rosen 1991a,b). ART is a cognitive and neural theory of how the brain quickly learns to categorise, recognise, and predict objects and events in a changing world. One of the key computational ideas within ART is that top-down learned expectations focus attention upon bottom-up information in a way that protects previously learned memories from being washed away by new learning, and enables new learning to be automatically incorporated into the total knowledge base of the system in a globally self-consistent way.

Most prototype-based CL algorithms employ either the winner-take-all (WTA) paradigm or the winner-take-most (WTM) paradigm. The major issue with the WTA paradigm is the possible existence of dead nodes. In such cases, some prototypes can never become a winner because of inappropriate initialisation, and therefore they have no contribution to learning. WTM decreases the dependency on the initialisation of prototype locations; however, an undesirable side effect is that since all prototypes are attracted to each input pattern, some of them are detracted from their corresponding clusters. Zhang and Liu proposed a self-splitting competitive learning (SSCL) clustering based on a distinct paradigm: one-prototype-take-one-cluster (OPTOC) (Zhang and Liu, 2002).

Aforementioned prototype-based algorithms have the advantage of being able to incorporate knowledge about the global shape or size of clusters by using appropriate prototypes and distance measures in the objective function. However, these algorithms suffer from three major drawbacks, namely, the difficulty in determining the number of clusters, the sensitivity to noise and outliers, and the sensitivity to initialisation. Rhouma and Frigui developed a self-organisation of pulse-coupled oscillators algorithm, called self-organising oscillator networks (SOON) (Rhouma and Frigui, 2001), which was used to analyse microarray gene expression data later by Salem, Jack and Nandi (2008). The SOON algorithm has its root in a biological process with an interesting physical characteristic: Fireflies flash at random when considered by themselves; however, they exhibit the characteristic of firing together when in groups that are physically close to each other. The objective of this chapter is to introduce these neural network-based algorithms.

14.2 Algorithms

14.2.1 SOM

SOM belongs to one of the categories of neural network architectures, where neighbouring cells in a neural network compete in their activities by means of mutual lateral interactions, and develop adaptively into specific detectors of different signal patterns. The objective of SOM is to represent high-dimensional input patterns with prototype vectors that can be visualised in a usually two-dimensional lattice structure.

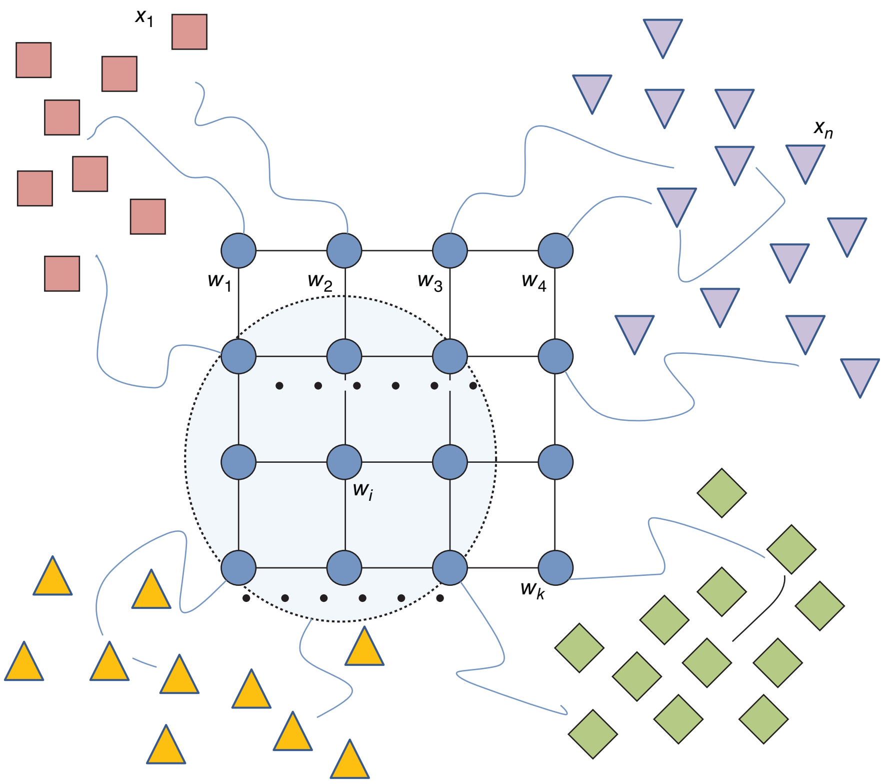

Figure 14.1 Demonstration of the strategy of SOM. Each node on the grid represents a neuron with a weight vector $\boldsymbol{w}_i, \boldsymbol{i} = 1, \ldots, \boldsymbol{K}$. Only the winning neuron, for example $\boldsymbol{w}_i$ and its surrounding neurons, which are in the dotted circle, have the opportunity to update

Consider a 4×4 grid SOM depicted in Figure 14.1. Note that the arrangement of the grid is not necessary to be rectangular; it can be hexagonal, circular, and so on. The grid in Figure 14.1 has $K = 16$ neurons and each neuron has a weight vector, denoted by $\boldsymbol{w}_i$, $i = 1, \ldots, K$ which is initialised randomly. Let $\boldsymbol{X} = \{\boldsymbol{x}_n | n = 1, \ldots, N\}$ be the input data. We calculate the Euclidean distances between input data $\boldsymbol{X}$ and weight vectors, and then we can obtain for each data vector $\boldsymbol{x}_n$ the best match unit (BMU) that minimises the Euclidean distance between $\boldsymbol{x}_n$ and weight vectors, expressed as shown in Equation (14.1).

$$\mathrm{BMU} = \arg\min_j \left(\boldsymbol{x}_n - \boldsymbol{w}_j\right) \tag{14.1}$$

For the WTA paradigm, only BMU has the opportunity to update its weight vector. Virtually, SOM employs the WTM paradigm, in which at each learning step all neurons within a neighbourhood set around BMU can also be updated. The width or radius of this neighbourhood set is time-varying: it shrinks monotonically with time and ends with only BMU in the set. Let us define the neighbourhood set as N_s. The updating process in time t may be expressed as Equation (14.2).

Table 14.1 Summary of the clustering process of SOM by Kohonen (1990)

Step 1	Define the topology of SOM and initialise each node's weights $\boldsymbol{w}_i(0), i=1,\ldots,K$, randomly;
Step 2	Find the best matching unit (BMU) by calculating the distance between the input vector and the weights of each node, that is $J=\arg_j\min(\boldsymbol{x}-\boldsymbol{w}_j)$;
Step 3	The radius of the neighbourhood around the BMU is calculated. The size of the neighbourhood decreases with each iteration;
Step 4	Each node in the BMU's neighbourhood has its weights adjusted to become more like the BMU. Nodes closest to the BMU are altered more than the nodes furthest away in the neighbourhood;
Step 5	Repeat from step 2 for enough iterations for convergence.

$$\boldsymbol{w}_i(t+1)=\begin{cases}\boldsymbol{w}_i(t)+h(t)[\boldsymbol{x}_n-\boldsymbol{w}_i(t)] & \textbf{\textit{if}}\ i\in N_s(t)\\ \boldsymbol{w}_i(t) & \textbf{\textit{if}}\ i\notin N_s(t)\end{cases} \tag{14.2}$$

where $h(t)$ is the neighbourhood function that is often defined as given in Equation (14.3),

$$h(t)=\alpha(t)\exp\left(-\frac{\boldsymbol{r}_{\mathrm{BMU}}-\boldsymbol{r}_i{}^2}{2\sigma^2(t)}\right) \tag{14.3}$$

where $\alpha(t)$ is the monotonically decreasing learning rate, $\boldsymbol{r}$ represents the position of corresponding neuron, and $\sigma(t)$ is the monotonically decreasing kernel width function. The summary of the clustering process of SOM is given in Table 14.1. In MATLAB, SOM can be performed by a combination of several functions, namely ***newsom***, which initialises a SOM network, ***train***, which trains the SOM network, and ***sim***, which performs the clustering using the trained network. Note that the input data to the ***train*** function are the same as those to the ***sim*** function for clustering purposes. There are three packages in *R* providing SOM functions, namely ***kohonen*** package, ***som*** package, and ***wccsom*** package. In Weka, the function ***SelfOrganizingMap*** can be called from package ***weka.clusterers***.

14.2.2 GLVQ

LVQ discovers cluster structure hidden in unlabelled data. In principle, *k*-means, which was introduced in Chapter 10, and LVQ are very much alike. In LVQ, the salient feature is that the LVQ network consists of two layers, namely input layer and competitive layer. Each neuron in the competitive layer has a weight vector (or prototype) attached to it. The prototypes $\boldsymbol{W}=\{\boldsymbol{w}_i|i=1,\ldots,K\}$ are essentially a network array of cluster centres. When an input vector $\boldsymbol{x}_n$ is submitted to this network, distances are computed between each weight vector $\boldsymbol{w}_i$ and $\boldsymbol{x}_n$. The neurons in the competitive layer compete and a winner neuron with minimum distance is found. Subsequently, the weight vector of the winner neuron is updated by using Equation (14.4).

$$\boldsymbol{w}_i(t+1)=\boldsymbol{w}_i(t)+\alpha(t)[\boldsymbol{x}_k-\boldsymbol{w}_i(t)] \tag{14.4}$$

LVQ suffers problems which result from two causes: (1) an improper choice of initial neurons, and (2) the WTA strategy.

Pal *et al.* developed a GLVQ algorithm to circumvent these issues (Pal, Bezdek and Tsao, 1993). Let $L(\boldsymbol{X}, \boldsymbol{W})$ be a loss function which measures the locally weighted mismatch of $\boldsymbol{x}_n$ with respect to the winner neuron, as given in Equation (14.5),

$$L(\boldsymbol{X},\boldsymbol{W}) = \sum_{n=1}^{N}\sum_{k=1}^{K} g_{nk}\|\boldsymbol{x}_n - \boldsymbol{w}_k\|^2 \tag{14.5}$$

where Equation (14.6) holds.

$$g_{nk} = \begin{cases} 1 & \text{if } k \text{ is winner} \\ \dfrac{1}{\sum_{j=1}^{K}\|\boldsymbol{x}_n - \boldsymbol{w}_j\|^2} & \text{otherwise} \end{cases} \tag{14.6}$$

Therefore, for a fixed set of data points $\boldsymbol{X}$, the problem reduces to the unconstrained optimisation problem as set out in Equation (14.7).

$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}} L(\boldsymbol{X},\boldsymbol{W}) \tag{14.7}$$

The solution of this minimisation problem can be approximated by local gradient descent search of $L(\boldsymbol{X}, \boldsymbol{W})$. Let us define $D = \sum_{j=1}^{K}\|\boldsymbol{x}_n - \boldsymbol{w}_j\|^2$, then the update rules are formulated as shown in Equation (14.8),

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \alpha(t)[\boldsymbol{x}_n - \boldsymbol{w}_i(t)]\frac{D^2 - D + \|\boldsymbol{x}_n - \boldsymbol{w}_i(t)\|^2}{D^2} \tag{14.8}$$

for the winner neuron, say i, of the data point $\boldsymbol{x}_n$; and for other $(K-1)$ neurons Equation (14.9) holds.

$$\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) + \alpha(t)\left[\boldsymbol{x}_n - \boldsymbol{w}_j(t)\right]\frac{\|\boldsymbol{x}_n - \boldsymbol{w}_j(t)\|^2}{D^2} \tag{14.9}$$

To avoid possible oscillations of the solution, the amount of correction should be reduced as iteration proceeds, thus $\alpha(t)$ is defined as $\alpha_0(1-t/T)$ to satisfy the condition $t \to \infty$, $\alpha(t) \to 0$, where $\alpha_0 \in (0,1)$ and T is the maximum number of iterations.

14.2.3 Neural-gas

Neural-gas proposed by Martinetz, Berkovich and Schulten (1993) is a neural network-based clustering inspired by LVQ and SOM. The algorithm was named ‘neural-gas’ because it employs the dynamics of the feature vectors during the adaptation process, which distribute themselves like gas within the data space.

To avoid being confined to local minima during the adaptation procedure, like k-means, an adaptation approach called ‘soft-max’ is introduced into the neural-gas algorithm. Moreover, a neighbourhood ranking strategy is also employed so that not only the winning neuron but all neurons depending on their proximity to input data vector can be updated. Each time data vector $\boldsymbol{x}_n$ is presented, the neighbourhood ranking $(\boldsymbol{w}_{i_0}, \boldsymbol{w}_{i_1}, \ldots, \boldsymbol{w}_{i_{K-1}})$ of the weight vectors is determined according to their distances from $\boldsymbol{x}_n$, where $\boldsymbol{w}_{i_0}$ is the closest neuron to $\boldsymbol{x}_n$, and $\|\boldsymbol{w}_{i_{k-1}} - \boldsymbol{x}_n\| < \|\boldsymbol{w}_{i_k} - \boldsymbol{x}_n\|$. Let $k_i(\boldsymbol{x}_n, \boldsymbol{w})$ denote the number k associated with each vector $\boldsymbol{w}_i$. The adaptation step for adjusting the $\boldsymbol{w}_i$ is given by Equation (14.10),

$$\Delta \boldsymbol{w}_i = \epsilon \cdot h_\lambda(k_i(\boldsymbol{x}_n, \boldsymbol{w})) \cdot (\boldsymbol{x}_n - \boldsymbol{w}_i),\ i = 1, \ldots, K \tag{14.10}$$

where the step size $\epsilon \in [0,1]$ describes the overall extent of the modification and $h_\lambda(k_i(\boldsymbol{x}_n, \boldsymbol{w})) = \exp(-k_i(\boldsymbol{x}_n, \boldsymbol{w})/\lambda)$.
The dynamics of the neurons obey a stochastic gradient descent on the cost function, given in Equation (14.11),

$$L(\boldsymbol{X}, \boldsymbol{W}) = \sum_{n=1}^{N} \sum_{i=1}^{K} \frac{1}{2C(\lambda)} p_i(\boldsymbol{x}_n) h_\lambda(k_i(\boldsymbol{x}_n, \boldsymbol{w})) \|\boldsymbol{w}_i - \boldsymbol{x}_n\|^2 \tag{14.11}$$

where $p_i(\boldsymbol{x}_n)$ is the fuzzy membership of data vector $\boldsymbol{x}_n$ to cluster i, and is expressed as Equation (14.12),

$$p_i(\boldsymbol{x}_n) = \frac{h_\lambda(k_i(\boldsymbol{x}_n, \boldsymbol{w}))}{C(\lambda)} \tag{14.12}$$

where Equation (14.13) holds.

$$C(\lambda) = \sum_{i=1}^{K} h_\lambda(k_i(\boldsymbol{x}_n, \boldsymbol{w})) \tag{14.13}$$

There are two main advantages of the neural-gas algorithm: (1) it converges quickly to low distortion errors; (2) it reaches a distortion error lower than that resulting from k-means and SOM. However, the nature of the neural-gas algorithm requires knowledge of the number of neurons in advance, which limits the use of neural-gas in practical applications where the problem is to determine a suitable number of neurons *a priori.* Depending on the complexity of the data distribution to be modelled, very different numbers of neurons may be appropriate. Nevertheless, in light of the neural-gas algorithm, many other algorithms were proposed to circumvent this problem, for example the growing neural-gas (GNG) algorithm (Fritzke, 1995), and the growing cell structures (GCS) algorithm (Fritzke, 1994).

14.2.4 ART

ART proposes a solution of the *stability-plasticity* dilemma (Grossberg, 1976, 1980): an adequate algorithm must be capable of plasticity in order to learn about significant new events, yet it must also remain stable in response to irrelevant or often repeated events. ART can learn arbitrary input patterns in a stable, fast, and self-organising way. ART itself does not possess a neural network architecture; it is rather a learning theory in which resonance in neural circuits can trigger fast learning. Since it was invented more than two decades ago, ART has become a large family of neural network architectures with many variants, which are summarised in Table 14.2. It is not possible for us to detail all ART algorithms because of the limit of space. We focus on the basic concept of ART.

ART-1, which is the first version of ART networks for binary data, is shown in Figure 14.2. There are two major subsystems: *attentional subsystem* and *orienting subsystem.*

Table 14.2 Summary of the variants of ART

Name	Comments	References
ART-1	The simplest variety of ART networks, accepting only binary inputs	Carpenter and Grossberg (1987b)
ART-2	Extended to analog input patterns	Carpenter and Grossberg (1987a)
ART-2A	A streamlined form of ART-2 with a drastically accelerated runtime	Carpenter, Grossberg and Reynolds (1991)
ART-3	A new mechanism originating from elaborate biological processes to achieve more efficient parallel search in hierarchical structures	Carpenter and Grossberg (1990)
Fuzzy ART	Fuzzy logic was incorporated into ART's pattern recognition, thus generalising ART	Carpenter, Grossberg and Rosen (1991a)
ARTMAP	Known as Predictive ART, combines two slightly modified ART-1 or ART-2 units into a supervised learning structure	Carpenter, Grossberg and Rosen (1991b)
Fuzzy ARTMAP	ARTMAP using fuzzy ART units	Carpenter *et al.* (1992)
AHN	Adaptive Hamming net, a fast-learning ART 1 model without searching	Hung and Lin (1995)
Gaussian ARTMAP	A synthesis of a Gaussian classifier and ART neural network	Williamson (1996)
PART	Projective ART	Cao and Wu (2002)

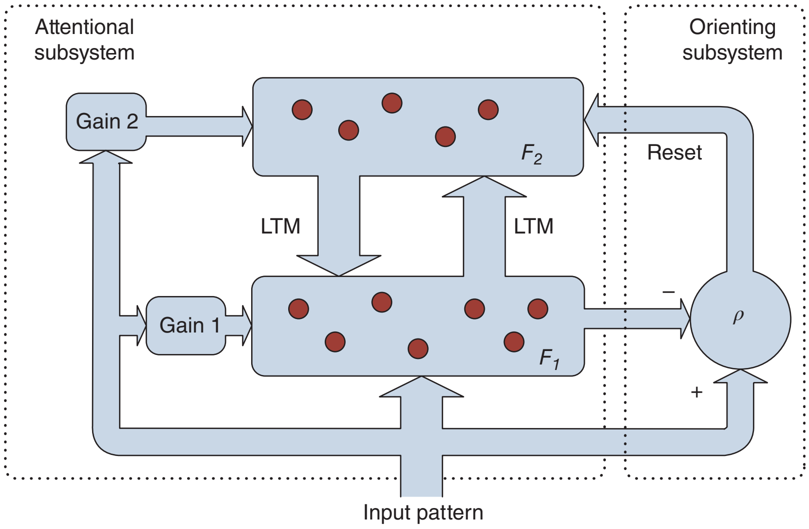

Figure 14.2 ART-1 consists of an attentional subsystem and an orienting subsystem. The attentional subsystem has two STM layers, F_1 and F_2. LTM traces between F_1 and F_2 multiply the signal in these pathways. Gain controls enable F_1 and F_2 to distinguish current stages of a running circle. The orienting subsystem generates a reset wave to F_2 when mismatches between bottom-up and top-down patterns occur at F_1

Multiple interacting memory systems are needed to monitor and adaptively react to the novelty of events. Interactions between two functionally complementary subsystems are needed to process familiar and unfamiliar events. Familiar events are processed with an attentional subsystem. This subsystem establishes precise internal representations of and responses to familiar events. It also builds up the learned top-down expectations that help to stabilise the learned bottom-up codes of familiar events. The orienting subsystem is essential for expressing whether a novel pattern is familiar and well represented by an existing recognition code or unfamiliar and in need of a new recognition code. It resets the attentional subsystem when an unfamiliar event occurs.

The attentional subsystem consists of a two-layer short-term memory (STM) structure where F_1 is called feature representation field and F_2 is called category representation field. A stream of input patterns to a network F_1 can train the adaptive weights, or long-term memory (LTM) traces that multiply the signals in the pathways from F_1 to a coding level F_2. Each neuron in F_1 is connected to all neurons in F_2 via the continuous-valued bottom-up weight matrix $\boldsymbol{W}^{12}$ and top-down weight matrix $\boldsymbol{W}^{21}$. The prototypes of clusters are stored in layer F_2. Processing in ART-1 can be divided into four phases: recognition, comparison, search, and learn.

Recognition phase is also known as bottom-up activation. Initially, if no input pattern is applied, all recognition in F_2 is disabled and two control gains G_1 and G_2, are set to zero. This causes all F_2 elements to be zero, giving them equal chance to win the subsequent recognition competition. The gain control 2 depends only on the input pattern: $G_2 = 1$ if there is an input; $G_2 = 0$, otherwise. The gain control 1 depends on both the input pattern and the output from F_2, denoted by O_2: $G_1 = 1$, if there is an input and $O_2 = 0$; $G_1 = 0$, otherwise. Each node in F_1 whose activity is beyond the threshold $(G_1 = 1)$ sends excitatory outputs to the nodes in F_2. The F_1 output pattern O_1 is multiplied by the LTM traces $\boldsymbol{W}^{12}$. Each node in F_2 sums up all its LTM gated signals, as shown in Equation (14.14).

$$v_{2j} = \sum_i O_{1i} W_{ij}^{12} \tag{14.14}$$

These connections represent the input pattern classification categories, where each weight stores one category. The output O_{2j} is defined so that the element that receives the largest input should be the winner. As such, the layer F_2 works as a WTA strategy described by Equation (14.15),

$$O_{2j} = \begin{cases} 1 & \text{if } G_2 = 1 \cap v_{2j} = \max\{v_{2j}\} \\ 0 & \text{otherwise} \end{cases} \tag{14.15}$$

where $\cap$ represents the logical AND operation. The F_2 unit that receives the largest F_1 output is the one that best matches the input vector category, thus wins the competition. The F_2 winner node fires and simultaneously inhibits all other nodes in the layer.

In the comparison phase, also known as top-down template matching, the STM activation pattern O_2 on F_2 generates a top-down template on F_1. This pattern is multiplied by the LTM traces $\boldsymbol{W}^{21}$ and each node in F_1 obtains values shown in Equation (14.16).

$$v_{1i} = \sum_j O_{2j} W_{ij}^{21} \tag{14.16}$$

The most active recognition unit from F_2 passes a signal back to comparison layer F_1. Now G_1 is inhibited because the output from F_2 is active. If there is a good match between the top-down template and the input vector, the system becomes stable and learning then occurs; otherwise, the reset layer in the orienting subsystem then inhibits the F_2 layer. There is a threshold, called vigilance level $\rho \in (0,1]$. If the match degree is less than the vigilance level, the reset signal is then sent to F_2.

The salient difference between ART-1 and ART-2 (fuzzy ART) is that the binary AND operator in ART-1 is replaced by the fuzzy AND operator, which is defined as $A \wedge B = \min(A,B)$. The learning of the LTM traces W_j when the node j is active, is then given by Equation (14.17).

$$W_j^{new} = \beta\left(\boldsymbol{x} \wedge W_j^{old}\right) + (1-\beta)W_j^{old} \tag{14.17}$$

Many variants of ART have been developed and applied to large-scale technological and biological applications by authors around the world. Readers who are interested may be referred to a quite recent survey paper by Grossberg (2013). ART-1, ART-2, and ARTMAP are available in *R* package ***RSNNS***.

14.2.5 OPTOC

OPTOC is a CL paradigm proposed by Zhang and Liu (2002). Unlike WTA and WTM paradigms, the key technique used in OPTOC is that, for each prototype, an online learning vector, asymptotic property vector (APV) is assigned to guide the learning of this prototype. With the 'help' of the APV, each prototype will locate only one natural cluster and ignore other clusters in the case that the number of prototypes is less than that of clusters. One of the well-known critical problems with CL is the difficulty in determining the number of clusters. With the OPTOC learning paradigm, SSCL algorithm starts from only a single prototype which is randomly initialised in the feature space. During the learning period, one of the prototypes (initially, the only single prototype) will be chosen to split into two prototypes based on a split validity measure. This self-splitting behaviour terminates if no more prototypes are suitable for further splitting.

Given each prototype, $\boldsymbol{P}_k$, the key technique is that an online learning vector, APV $\boldsymbol{A}_k$ is assigned to guide the learning of this prototype. For simplicity, $\boldsymbol{A}_k$ represents the APV for prototype $\boldsymbol{P}_k$ and n_{A_k} denotes the learning counter (winning counter) of $\boldsymbol{A}_k$. As a necessary condition of OPTOC mechanism, $\boldsymbol{A}_k$ is required to initialise at a random location, which is far from its associated prototype $\boldsymbol{P}_k$ and n_{A_k} is initially zero. Taking the input pattern $\boldsymbol{x}_n$ as a neighbour if it satisfies the condition $\langle \boldsymbol{P}_k, \boldsymbol{x}_n \rangle \le \langle \boldsymbol{P}_k, \boldsymbol{A}_k \rangle$, where $\langle \cdot, \cdot \rangle$ is the inner product operator. To implement the OPTOC paradigm, $\boldsymbol{A}_k$ is updated online to construct a dynamic neighbourhood of $\boldsymbol{P}_k$. The patterns 'outside' of the dynamic neighbourhood will contribute less to the learning of $\boldsymbol{P}_k$ as compared with those 'inside' patterns.

The update of $\boldsymbol{A}_k$ depends on the relative locations of the input data point $\boldsymbol{x}_n$, prototype $\boldsymbol{P}_k$ and $\boldsymbol{A}_k$ itself, as mathematically given by Equation (14.18),

$$\boldsymbol{A}_k(t+1) = \boldsymbol{A}_k(t) + \frac{1}{n_{A_k}} \cdot \delta_k \cdot (\boldsymbol{x}_n - \boldsymbol{A}_k(t)) \cdot \Theta(\boldsymbol{P}_k(t), \boldsymbol{x}_n, \boldsymbol{A}_k(t)) \tag{14.18}$$

where $0 < \delta_k \le 1$, and δ_k is defined as given in Equation (14.19),

$$\delta_k = \left(\frac{\langle \boldsymbol{P}_k, \boldsymbol{A}_k \rangle}{\langle \boldsymbol{P}_k, \boldsymbol{x}_n \rangle + \langle \boldsymbol{P}_k, \boldsymbol{A}_k \rangle} \right)^2 \tag{14.19}$$

and $\Theta(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ is a general function given by Equation (14.20)

$$\Theta(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) = \begin{cases} 1 & \langle \boldsymbol{a},\boldsymbol{b} \rangle \leq \langle \boldsymbol{a},\boldsymbol{c} \rangle \\ 0 & \text{otherwise} \end{cases} \tag{14.20}$$

For each update of $\boldsymbol{A}_k$, its learning counter $n_{\boldsymbol{A}_k}$ is computed as shown in Equation (14.21).

$$n_{\boldsymbol{A}_k} = n_{\boldsymbol{A}_k} + \delta_k \cdot \Theta(\boldsymbol{P}_k(t), \boldsymbol{x}_n, \boldsymbol{A}_k(t)) \tag{14.21}$$

Thus, it may be observed that APV $\boldsymbol{A}_k$ always attempts to move toward $\boldsymbol{P}_k$. The update of the prototype $\boldsymbol{P}_k$ is given by Equation (14.22),

$$\boldsymbol{P}_k(t+1) = \boldsymbol{P}_k(t) + \alpha_k(\boldsymbol{x}_n - \boldsymbol{P}_k(t)) \tag{14.22}$$

where Equation (14.23) holds.

$$\alpha_k = \left(\frac{\langle \boldsymbol{P}_k, \boldsymbol{A}_k \rangle}{\langle \boldsymbol{P}_k, \boldsymbol{x}_n \rangle + \langle \boldsymbol{P}_k, \boldsymbol{A}_k \rangle} \right)^2 \tag{14.23}$$

In addition to the APV, there is another auxiliary vector, called distant property vector (DPV) $\boldsymbol{R}_k$, assisting the cluster, which contains more than one prototype, to split. Let $n_{\boldsymbol{R}_k}$ denote the learning counter for $\boldsymbol{R}_k$, which is initialised to zero. $\boldsymbol{R}_k$ will be updated to a distant location from $\boldsymbol{P}_k$. The efficiency of splitting is improved by determining the update schedule of $\boldsymbol{R}_k$ adaptively from the analysis of the feature space. Contrary to the APV $\boldsymbol{A}_k$, the DPV $\boldsymbol{R}_k$ always tries to move away from $\boldsymbol{P}_k$. The update of DPV $\boldsymbol{R}_k$ is given by Equation (14.24),

$$\boldsymbol{R}_k(t+1) = \boldsymbol{R}_k(t) + \frac{1}{n_{\boldsymbol{R}_k}} \cdot \rho_k \cdot (\boldsymbol{x}_n - \boldsymbol{R}_k(t)) \cdot \Theta(\boldsymbol{P}_k(t), \boldsymbol{R}_k(t), \boldsymbol{x}_n) \tag{14.24}$$

where the learning rate of $\boldsymbol{R}_k$, ρ_k, is given by Equation (14.25).

$$\rho_k = \left(\frac{\langle \boldsymbol{P}_k, \boldsymbol{x}_n \rangle}{\langle \boldsymbol{P}_k, \boldsymbol{x}_n \rangle + \langle \boldsymbol{P}_k, \boldsymbol{R}_k \rangle} \right)^2 \tag{14.25}$$

The SSCL algorithm based on the OPTOC CL paradigm is considered as a solution of two long standing critical problems in clustering, namely (1) the difficulty in determining the number of clusters, and (2) the sensitivity to prototype initialisation.

14.2.6 SOON

Rhouma and Frigui developed a self-organisation of pulse-coupled oscillators algorithm (Rhouma and Frigui, 2001), which was later named as SOON (Salem, Jack and Nandi, 2008). SOON has its root in a biological process that fireflies flash at random when considered by themselves; however, they exhibit the characteristic of firing together when in groups that are physically close to each other. Technically, SOON is an efficient synchronisation model that organises a population of integrate-and-fire oscillators into stable and structured groups. Each oscillator fires synchronically with all the others within its group, but the groups themselves fire with a constant phase difference.

Let $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_N\}$ be a set of N oscillators, where each oscillator $\mathcal{O}_i$ is characterised by a phase ϕ_i and a state variable s_i, given by Equation (14.26),

$$s_i = f_i(\phi_i) = \frac{1}{b}\ln\left[1 + \left(e^b - 1\right)\phi_i\right] \tag{14.26}$$

where b is a constant value used to control the curve of the oscillator. Positive values of b will make the curve concave down, while negative values of b will make the curve concave up. $\phi_i \in [0,1]$ is the phase angle of the oscillator, and determines the likelihood of the oscillator to fire, where 0 is just fired and 1 is firing. The output of the oscillator s_i is bounded in the range [0, 1] for all values of $f_i(\phi_i)$ by Equation (14.27),

$$s_i^* = B\left(s_i + \epsilon_j(\phi_i)\right) \tag{14.27}$$

where $B(\cdot)$ is a limiting function as given by Equation (14.28),

$$B(x) = \begin{cases} x & \text{if } 0 \le x \le 1 \\ 0 & \text{if } x < 0 \\ 1 & \text{if } x > 1 \end{cases} \tag{14.28}$$

and $\epsilon_j(\phi_i)$ is the coupling strength of an oscillator j at a given phase ϕ_i, which is written as shown in Equation (14.29).

$$\epsilon_j(\phi_i) = \begin{cases} C_E\left[1 - \left(\frac{d_{ij}}{\delta_0}\right)^2\right], & \text{if } d_{ij} \le \delta_0 \\ -C_I\left[\left(\frac{d_{ij} - \delta_0}{\delta_1 - \delta_0}\right)^2\right], & \text{if } \delta_0 < d_{ij} \le \delta_1 \\ -C_I, & \text{otherwise} \end{cases} \tag{14.29}$$

Having decided on a limit distance δ_0, δ_1 is set to be five times δ_0. The coupling function promotes all oscillators that lie within the distance δ_0, increasing the phase value by the constant of excitation C_E, multiplied by the factor depending on the ratio of the distance between the winning oscillator and the oscillator under consideration, and δ_0. This, in turn, makes the group of oscillators more likely to synchronise with the winning oscillator in future iterations. The phase of all those with distance lying in the interval $\delta_0 < d_{ij} \le \delta_1$ are inhibited by C_I, the coefficient of inhibition multiplied by a ratio that takes into account how close the oscillator under consideration is to the winning oscillator. All values of $d_{ij} > \delta_1$ are hard limited to $-C_I$. The value of C_E is typically relatively small, of the order of 0.1 ~ 0.2. The value of C_I is normally set to the value C_E/N (N is the number of data points), as any given data point is likely to be inhibited more often than it is likely to be excited.

Once a group of oscillators synchronise, their dynamics must remain identical in order to keep the synchronisation. Let $G=\{\mathcal{O}_{g_1},\ldots,\mathcal{O}_{g_n}\}$ be a set of n oscillators that have synchronised. To keep these oscillators synchronised, the coupling strength of group members should be the same [Equation (14.30)].

$$\epsilon_i(\phi_{g_1})=\epsilon_i(\phi_{g_2})=\cdots=\epsilon_i(\phi_{g_n}),\quad \text{or}\quad r_{ig_1}=r_{ig_2}=\cdots=r_{ig_n}=\hat{r}_{iG} \tag{14.30}$$

There are three possible choices for $\hat{r}_{iG}$, corresponding to single linkage, complete linkage, and average linkage, respectively, and these are examined in Equation (14.31).

$$\hat{r}_{iG}=\begin{cases}\min(r_{ig_1},r_{ig_2},\cdots,r_{ig_n})\\ \max(r_{ig_1},r_{ig_2},\cdots,r_{ig_n})\\ \dfrac{1}{n}\sum_{j=1}^{n} r_{ig_j}\end{cases} \tag{14.31}$$

Different choices may generate quite different results in many circumstances.

There are two SOON algorithms: SOON-1 and SOON-2 (Rhouma and Frigui, 2001; Salem, Jack and Nandi, 2008), which are summarised in Table 14.3 and Table 14.4, respectively. SOON-1 was designed for the application with only relational data available; for example, networks. SOON-2 was designed for the datasets where each data object is characterised by M numerical features. Compared with SOON-1, SOON-2 is more efficient because it uses prototypes to both represent clusters and avoid computing and storing the pairwise distances, which is a potential problem when the size of the dataset is big. SOON-2 starts with a set of K initial prototypes $\{\boldsymbol{P}_k\}$, $k=1,\ldots,K$, and each data object $\boldsymbol{x}_i$ is represented by an oscillator $\mathcal{O}_i$. The distance between oscillator i and prototype k, is denoted by $d_{ik}=d(\boldsymbol{x}_i,\boldsymbol{P}_k)$. Note that the distance is adjustable. If $\mathcal{O}_i$ belongs to one of the synchronised groups, then its distance will

Table 14.3 Summary of the SOON-1 algorithm

```
Step 1  Construct the relative distance matrix $\boldsymbol{D}=\{d_{ij}\}$; Initialise phases $\phi_i$ randomly;
Step 2  Identify the next oscillator to fire: $\left\{\mathcal{O}_i:\phi_i=\max_{j=1,\ldots,N}\phi_j\right\}$;
Step 3  Bring $\phi_i$ to threshold, and adjust other phases: $\phi_k=\phi_k+(1-\phi_i)$ for $k=1,\ldots,N$;
Step 4  **FOR** all oscillators $\mathcal{O}_j$, $(j\neq i)$ DO
            Compute state variable $s_k=f(\phi_k)$ using Equation (14.26);
            Compute coupling strength $\epsilon_i(\phi_k)$ using Equation (14.29);
            Adjust state variables using Equation (14.27);
            Compute new phases using $\phi_k=f^{-1}(s_k)$
        **END FOR**
Step 5  Identify synchronised oscillators and reset their phases;
Step 6  Adjust relations of all oscillators that have synchronised in Equation (14.31);
Step 7  Repeat Step 2 until synchronised group stabilises.
```

Table 14.4 Summary of the SOON-2 algorithm

Step 1	Select a distance measure $d(\cdot,\cdot)$; Initialise phases ϕ_i, $i=1,\ldots,N$ randomly; Set K, and initialise the prototype P_k randomly for $k=1,\ldots,K$;
Step 2	Identify the next oscillator to fire: $\left\{\mathcal{O}_i : \phi_i = \max\limits_{j=1,\ldots,N} \phi_j\right\}$;
Step 3	Identify the closest prototype to $\mathcal{O}_i$: $P^* : d(\mathcal{O}_i, P^*) = \min\limits_{k=1,\ldots,K} d(\mathcal{O}_j, P_k)$;
Step 4	Compute $d(\mathcal{O}_j, P^*)$ for $j=1,N$ and adjust them using Equation (14.32)
Step 5	Bring ϕ_i to threshold, and adjust other phases: $\phi_k = \phi_k + (1-\phi_i)$ for $k=1,\ldots,N$;
Step 6	**FOR** all oscillators $\mathcal{O}_j$, $(j \neq i)$ DO
	Compute state variable $s_k = f(\phi_k)$ using Equation (14.26);
	Compute coupling strength $\epsilon_i(\phi_k)$ using Equation (14.29);
	Adjust state variables using Equation (14.27);
	Compute new phases using $\phi_k = f^{-1}(s_k)$
	END FOR
Step 7	Identify synchronised oscillators and reset their phases;
Step 8	Update prototype $\boldsymbol{P}_k$;
Step 9	Repeat Step 2 until synchronised group stabilises.

be replaced by the average distances of all oscillators that constitute the group G_k, which is written as Equation (14.32).

$$\hat{d}_{ik} = \begin{cases} d_{ik} & \text{if } \mathcal{O}_i \notin G_k \\ \dfrac{\sum_{l \in G_k} d_{il}}{|G_k|} & \text{if } \mathcal{O}_i \in G_k \end{cases} \tag{14.32}$$

14.3 Discussion

There have been many neural network-based clustering algorithms applied in the bioinformatics field. The collections of publicly accessible resources for neural network-based clustering are shown in Table 14.5. A successful example of their application in the analysis of gene expression data was by Tamayo and colleagues (1999), where SOM was found to be significantly superior to hierarchical clustering and k-means in both robustness and accuracy. Hsu *et al.* proposed an algorithm combining unsupervised hierarchical clustering and SOM to perform class discovery and marker-gene identification in microarray gene expression data

Table 14.5 Collection of publicly accessible resources for neural network-based clustering

Algorithm name	Platform	Package	Function
SOM	*R*	kohonen/som/wccsom	som
	MATLAB		newsom/train/sim
	weka	Weka.clusterers	SelfOrganizingMap
ART	*R*	RSNNS	art1/art2/artmap

(Hsu, Tang and Halgamuge, 2003). Wang *et al.* also employed SOM as the first step of their algorithm (Wang *et al.*, 2003). The aim of applying the SOM procedure in the algorithm was to find map units that could represent the configuration of the input dataset, and at the same time to achieve a continuous mapping from the input gene space to a lattice. Chavez-Alvarez *et al.* conducted a study very recently to use SOM to cluster multiple yeast cell cycle datasets (Chavez-Alvarez, Chavoya and Mendez-Vazquez, 2014). Besides SOM, there are also many other algorithms applied in the bioinformatics field; for example, fuzzy ART was applied to analyse the time series expression data during sporulation of budding yeast (Tomida *et al.*, 2002); another variant of ART, called projective ART, was developed to select specific genes for each subtype in a cancer diagnosis marker-extraction study of soft-tissue sarcomas (Takahashi, Kobayashi and Honda, 2005; Takahashi *et al.*, 2006), and later, the analytical results were further investigated (Takahashi *et al.*, 2013). A self-splitting and merging competitive clustering algorithm based on the OPTOC paradigm was applied to identify biologically relevant groups of genes (Wu *et al.*, 2004); SOON was applied to cluster microarray data by Salem, Jack and Nandi (2008). We will discuss these applications in more detail in Chapter 19.

References

Cao, Y. and Wu, J. (2002). Projective ART for clustering data sets in high dimensional spaces. *Neural Networks,* **15**(1), pp. 105–120.

Carpenter, G.A. and Grossberg, S. (1987a). ART 2: self-organization of stable category recognition codes for analog input patterns. *Applied Optics,* **26**(23), pp. 4919–4930.

Carpenter, G.A. and Grossberg, S. (1987b). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image processing,* **37**(1), pp. 54–115.

Carpenter, G.A. and Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer,* **21**(3), pp. 77–88.

Carpenter, G.A. and Grossberg, S. (1990). ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks,* **3**(2), pp. 129–152.

Carpenter, G.A., Grossberg, S. and Reynolds, J.H. (1991). ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks,* **4**(5), pp. 565–588.

Carpenter, G.A., Grossberg, S. and Rosen, D.B. (1991a). ART 2-A: an adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks,* **4**(4), pp. 493–504.

Carpenter, G.A., Grossberg, S. and Rosen, D.B. (1991b). Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks,* **4**(6), pp. 759–771.

Carpenter, G.A. Grossberg, S., Markuzon, N. *et al.* (1992). Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks,* **3**(5), pp. 698–713.

Chavez-Alvarez, R., Chavoya, A. and Mendez-Vazquez, A. (2014). Discovery of possible gene relationships through the application of self-organizing maps to DNA microarray databases. *PloS One*, **9**(4), e93233.

Fritzke, B. (1994). Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural Networks,* **7**(9), pp. 1441–1460.

Fritzke, B. (1995). A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems,* **7**, pp. 625–632.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biological Cybernetics,* **23**(3), pp. 121–134.

Grossberg, S. (1980). How does a brain build a cognitive code? *Psychological Review,* **87**(1), pp. 1–52.

Grossberg, S. (2013). Adaptive Resonance Theory: how a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks,* **37**, pp. 1–47.

Hsu, A.L., Tang, S.-L. and Halgamuge, S.K. (2003). An unsupervised hierarchical dynamic self-organizing approach to cancer class discovery and marker gene identification in microarray data. *Bioinformatics,* **19** (16), pp. 2131–2140.

Hung, C.-A. and Lin, S.-F. (1995). Adaptive Hamming net: a fast-learning ART 1 model without searching. *Neural Networks,* **8**(4), pp. 605–618.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE,* **78**(9), pp. 1464–1480.

Kohonen, T. (2001). *Self-organizing maps*, Springer, New York.

Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik,* **14**(2), pp. 85–100.

Martinetz, T.M., Berkovich, S.G. and Schulten, K.J. (1993). 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks,* **4**(4), pp. 558–569.

Pal, N.R., Bezdek, J.C. and Tsao, E.-K. (1993). Generalized clustering networks and Kohonen's self-organizing scheme., *IEEE Transactions on Neural Networks,* **4**(4), pp. 549–557.

Rhouma, M.B.H. and Frigui, H. (2001). Self-organization of pulse-coupled oscillators with application to clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **23**(2), pp. 180–195.

Salem, S.A., Jack, L.B. and Nandi, A.K. (2008). Investigation of self-organizing oscillator networks for use in clustering microarray data. *IEEE Transactions on NanoBioscience,* **7**(1), pp. 65–79.

Takahashi, H., Kobayashi, T. and Honda, H. (2005). Construction of robust prognostic predictors by using projective adaptive resonance theory as a gene filtering method. *Bioinformatics,* **21**(2), pp. 179–186.

Takahashi, H., Nemoto, T., Yoshida, T. *et al.* (2006). Cancer diagnosis marker extraction for soft tissue sarcomas based on gene expression profiling data by using projective adaptive resonance theory (PART) filtering method. *BMC Bioinformatics*, **7**(1), p. 399.

Takahashi, H., Nakayama, R., Hayashi, S. *et al.* (2013). Macrophage Migration Inhibitory Factor and Stearoyl-CoA Desaturase 1: Potential Prognostic Markers for Soft Tissue Sarcomas Based on Bioinformatics Analyses. *PloS One*, **8**(10), e78250.

Tamayo, P., Slonim, D., Mesirov, J. *et al.* (1999). Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation. *Proceedings of the National Academy of Sciences,* **96**(6), pp. 2907–2912.

Tomida, S., Hanai, T., Honda, H. and Kobayashi, T. (2002). Analysis of expression profile using fuzzy adaptive resonance theory. *Bioinformatics,* **18**(8), pp. 1073–1083.

Wang, J., Bø, T.H., Jonassen, I. *et al.* (2003). Tumor classification and marker gene prediction by feature selection and fuzzy c-means clustering using microarray data. *BMC Bioinformatics*, **4**(1), p. 60.

Williamson, J.R. (1996). Gaussian ARTMAP: A neural network for fast incremental learning of noisy multidimensional maps. *Neural Networks,* **9**(5), pp. 881–897.

Wu, S., Liew, A.-C., Yan, H. and Yang, M. (2004). Cluster analysis of gene expression data based on self-splitting and merging competitive learning. *IEEE Transactions on Information Technology in Biomedicine,* **8**(1), pp. 5–15.

Zhang, Y.-J. and Liu, Z.-Q. (2002). Self-splitting competitive learning: a new on-line clustering paradigm. *IEEE Transactions on Neural Networks,* **13**(2), pp. 369–380.