

EBM-803 UNIT-II

Contents

| | | |
|-------|---|----|
| 1 | Unsupervised learning | 2 |
| 1.1 | Applications in genomics | 2 |
| 1.2 | Supervised and unsupervised learning | 2 |
| 2 | CLUSTERING | 3 |
| 2.1 | Typical cluster models include:..... | 4 |
| 2.2 | Clusterings can be roughly distinguished as: | 4 |
| 2.3 | Clustering algorithms | 4 |
| 2.3.1 | Connectivity based clustering (hierarchical clustering) | 5 |
| 2.3.2 | Centroid-based clustering | 6 |
| 2.3.3 | Distribution-based clustering | 7 |
| 2.3.4 | Density-based clustering | 7 |
| 2.3.5 | | 8 |
| 2.3.6 | | 8 |
| 2.4 | Recent developments..... | 8 |
| 2.5 | Applications | 9 |
| 3 | Competitive learning..... | 9 |
| 3.1 | Architecture and implementation | 9 |
| 3.2 | Example algorithm | 10 |
| 4 | Kohonen Network..... | 11 |
| 4.1 | How a Kohonen Network Recognizes | 11 |
| 4.2 | The Structure of the Kohonen Neural Network..... | 11 |
| 4.3 | Normalizing the Input | 12 |
| 4.4 | Calculating Each Neuron's Output | 13 |
| 4.5 | Mapping to Bipolar | 13 |
| 4.6 | Choosing the Winner..... | 14 |
| 4.7 | How a Kohonen Network Learns | 14 |
| 4.8 | Learning Rate | 15 |
| 4.9 | Adjusting Weights..... | 15 |
| 4.9.1 | Calculating the Error..... | 16 |
| 5 | Hebbian learning..... | 16 |
| 5.1 | Hebbian Learning | 17 |
| 5.2 | Mathematical Formulation | 18 |
| 5.3 | Plausibility..... | 18 |
| 6 | Adaptive resonance theory (ART) | 18 |
| 6.1 | Introduction | 18 |
| 6.2 | The Adaptive Resonance Theory: ART | 19 |
| 6.2.1 | ART1: The simplified neural network model | 19 |
| 6.2.2 | Operation | 20 |
| 6.2.3 | ART1: The original model | 21 |
| 6.2.4 | ART Applications | 22 |

1 Unsupervised learning

In machine learning, the problem of **unsupervised learning** is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning and reinforcement learning.

Unsupervised learning is closely related to the problem of density estimation in statistics. However unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data. Many methods employed in unsupervised learning are based on data mining methods used to preprocess data.

Approaches to unsupervised learning include:

- clustering (e.g., k-means, mixture models, hierarchical clustering)
- hidden Markov models,
- blind signal separation using feature extraction techniques for dimensionality reduction (e.g., principal component analysis, independent component analysis, non-negative matrix factorization, singular value decomposition)

Among neural network models, the self-organizing map (SOM) and adaptive resonance theory (ART) are commonly used unsupervised learning algorithms. The SOM is a topographic organization in which nearby locations in the map represent inputs with similar properties. The ART model allows the number of clusters to vary with problem size and lets the user control the degree of similarity between members of the same clusters by means of a user-defined constant called the vigilance parameter. ART networks are also used for many pattern recognition tasks, such as automatic target recognition and seismic signal processing. The first version of ART was "ART1", developed by Carpenter and Grossberg (1988).

1.1 Applications in genomics

Unsupervised learning techniques are widely used to reduce the dimensionality of high dimensional genomic data sets that may involve hundreds of thousands of variables. For example, weighted correlation network analysis is often used for identifying clusters (referred to as modules), modeling the relationship between clusters, calculating fuzzy measures of cluster (module) membership, identifying intramodular hubs, and for studying cluster preservation in other data sets.

1.2 Supervised and unsupervised learning

The learning algorithm of a neural network can either be supervised or unsupervised.

A neural net is said to learn supervised, if the desired output is already known.

Example: pattern association

Suppose, a neural net shall learn to associate the following pairs of patterns. The input patterns are decimal numbers, each represented in a sequence of bits. The target patterns are given in form of binary values of the decimal numbers:

| input pattern | target pattern |
|---------------|----------------|
| 0001 | 001 |
| 0010 | 010 |
| 0100 | 011 |
| 1000 | 100 |

While learning, one of the input patterns is given to the net's input layer. This pattern is propagated through the net (independent of its structure) to the net's output layer. The output layer generates an output pattern which is then compared to the target pattern. Depending on the difference between output and target, an error value is computed.

This output error indicates the net's learning effort, which can be controlled by the "imaginary supervisor". The greater the computed error value is, the more the weight values will be changed.

Neural nets that learn unsupervised have no such target outputs.

It can't be determined what the result of the learning process will look like.

During the learning process, the units (weight values) of such a neural net are "arranged" inside a certain range, depending on given input values. The goal is to group similar units close together in certain areas of the value range.

This effect can be used efficiently for pattern classification purposes.

2 CLUSTERING

Cluster analysis or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It will often be necessary to modify data preprocessing and model parameters until the result achieves the desired properties.

Besides the term *clustering*, there are a number of terms with similar meanings, including *automatic classification*, *numerical taxonomy*, *botryology* (from Greek βότρυς "grape") and *typological analysis*. The subtle differences are often in the usage of the results: while in data mining, the resulting groups are the matter of interest, in automatic classification the resulting discriminative power is of interest. This often leads to misunderstandings between researchers coming from the fields of data mining and machine learning, since they use the same terms and often the same algorithms, but have different goals.

2.1 Typical cluster models include:

- Connectivity models: for example hierarchical clustering builds models based on distance connectivity.
- Centroid models: for example the k-means algorithm represents each cluster by a single mean vector.
- Distribution models: clusters are modeled using statistical distributions, such as multivariate normal distributions used by the Expectation-maximization algorithm.
- Density models: for example DBSCAN and OPTICS defines clusters as connected dense regions in the data space.
- Subspace models: in Biclustering (also known as Co-clustering or two-mode-clustering), clusters are modeled with both cluster members and relevant attributes.
- Group models: some algorithms do not provide a refined model for their results and just provide the grouping information.
- Graph-based models: a **clique**, i.e., a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical form of cluster. Relaxations of the complete connectivity requirement (a fraction of the edges can be missing) are known as quasi-cliques.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example a hierarchy of clusters embedded in each other.

2.2 Clusterings can be roughly distinguished as:

- hard clustering: each object belongs to a cluster or not
- soft clustering (also: fuzzy clustering): each object belongs to each cluster to a certain degree (e.g. a likelihood of belonging to the cluster)

There are also finer distinctions possible, for example:

- strict partitioning clustering: here each object belongs to exactly one cluster
- strict partitioning clustering with outliers: objects can also belong to no cluster, and are considered outliers.
- overlapping clustering (also: alternative clustering, multi-view clustering): while usually a hard clustering, objects may belong to more than one cluster.
- hierarchical clustering: objects that belong to a child cluster also belong to the parent cluster
- subspace clustering: while an overlapping clustering, within a uniquely defined subspace, clusters are not expected to overlap.

2.3 Clustering algorithms

Clustering algorithms can be categorized based on their cluster model, as listed above. The following overview will only list the most prominent examples of clustering algorithms, as there are possibly over 100 published clustering algorithms. Not all provide models for their clusters and can thus not easily be categorized.

There is no objectively "correct" clustering algorithm, but as it was noted, "clustering is in the eye of the beholder." The most appropriate clustering algorithm for a particular problem often needs to be chosen experimentally, unless there is a mathematical reason to prefer one cluster model over another. It should be noted that an algorithm that is designed for one kind of model has no chance on

a data set that contains a radically different kind of model. For example, k-means cannot find non-convex clusters.

2.3.1 Connectivity based clustering (hierarchical clustering)

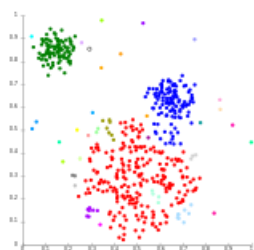
Main article: Hierarchical clustering

Connectivity based clustering, also known as *hierarchical clustering*, is based on the core idea of objects being more related to nearby objects than to objects farther away. These algorithms connect "objects" to form "clusters" based on their distance. A cluster can be described largely by the maximum distance needed to connect parts of the cluster. At different distances, different clusters will form, which can be represented using a dendrogram, which explains where the common name "hierarchical clustering" comes from: these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix.

Connectivity based clustering is a whole family of methods that differ by the way distances are computed. Apart from the usual choice of distance functions, the user also needs to decide on the linkage criterion (since a cluster consists of multiple objects, there are multiple candidates to compute the distance to) to use. Popular choices are known as single-linkage clustering (the minimum of object distances), complete linkage clustering (the maximum of object distances) or UPGMA ("Unweighted Pair Group Method with Arithmetic Mean", also known as average linkage clustering). Furthermore, hierarchical clustering can be agglomerative (starting with single elements and aggregating them into clusters) or divisive (starting with the complete data set and dividing it into partitions).

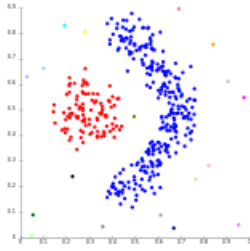
These methods will not produce a unique partitioning of the data set, but a hierarchy from which the user still needs to choose appropriate clusters. They are not very robust towards outliers, which will either show up as additional clusters or even cause other clusters to merge (known as "chaining phenomenon", in particular with single-linkage clustering). In the general case, the complexity is $\mathcal{O}(n^3)$, which makes them too slow for large data sets. For some special cases, optimal efficient methods (of complexity $\mathcal{O}(n^2)$) are known: SLINK for single-linkage and CLINK for complete-linkage clustering. In the data mining community these methods are recognized as a theoretical foundation of cluster analysis, but often considered obsolete. They did however provide inspiration for many later methods such as density based clustering.

- Linkage clustering examples



-

Single-linkage on Gaussian data. At 35 clusters, the biggest cluster starts fragmenting into smaller parts, while before it was still connected to the second largest due to the single-link effect.



-

Single-linkage on density-based clusters. 20 clusters extracted, most of which contain single elements, since linkage clustering does not have a notion of "noise".

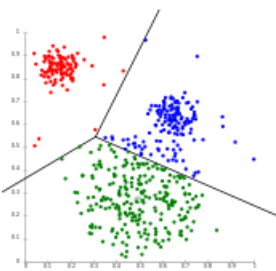
2.3.2 Centroid-based clustering

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to k , k -means clustering gives a formal definition as an optimization problem: find the k cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized.

Most k -means-type algorithms require the number of clusters - k - to be specified in advance, which is considered to be one of the biggest drawbacks of these algorithms. Furthermore, the algorithms prefer clusters of approximately similar size, as they will always assign an object to the nearest centroid. This often leads to incorrectly cut borders in between of clusters (which is not surprising, as the algorithm optimized cluster centers, not cluster borders).

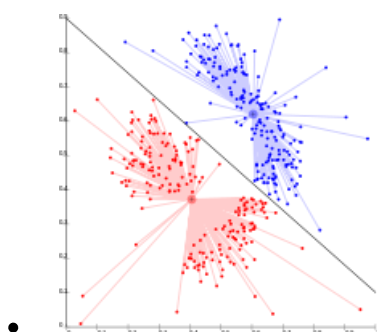
K -means has a number of interesting theoretical properties. On the one hand, it partitions the data space into a structure known as a Voronoi diagram. On the other hand, it is conceptually close to nearest neighbor classification, and as such is popular in machine learning. Third, it can be seen as a variation of model based classification, and Lloyd's algorithm as a variation of the Expectation-maximization algorithm for this model discussed below.

- k -Means clustering examples



-

K -means separates data into Voronoi-cells, which assumes equal-sized clusters (not adequate here)



-

K-means cannot represent density-based clusters

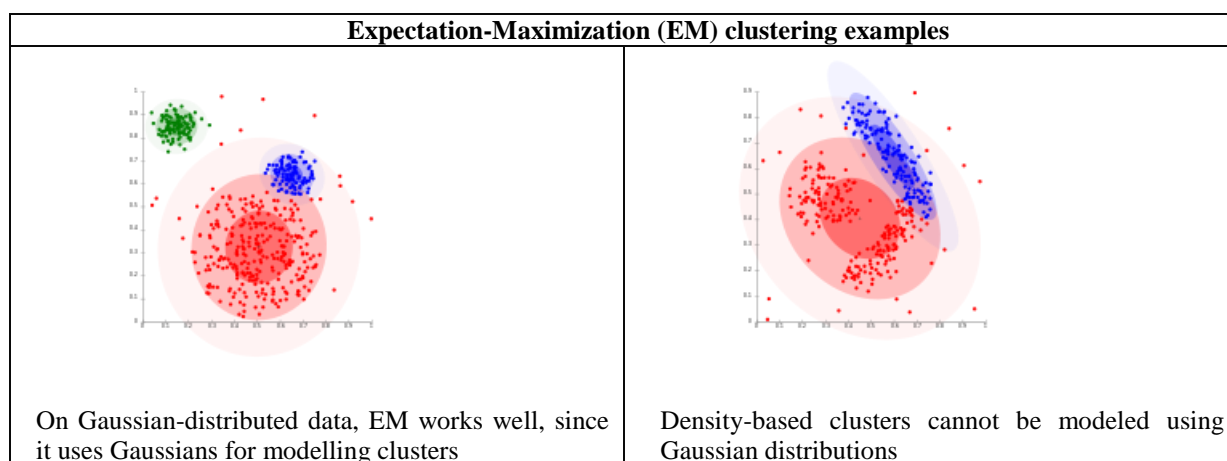
2.3.3 Distribution-based clustering

The clustering model most closely related to statistics is based on distribution models. Clusters can then easily be defined as objects belonging most likely to the same distribution. A nice property of this approach is that this closely resembles the way artificial data sets are generated: by sampling random objects from a distribution.

While the theoretical foundation of these methods is excellent, they suffer from one key problem known as overfitting, unless constraints are put on the model complexity. A more complex model will usually always be able to explain the data better, which makes choosing the appropriate model complexity inherently difficult.

One prominent method is known as Gaussian mixture models (using the expectation-maximization algorithm). Here, the data set is usually modelled with a fixed (to avoid overfitting) number of Gaussian distributions that are initialized randomly and whose parameters are iteratively optimized to fit better to the data set. This will converge to a local optimum, so multiple runs may produce different results. In order to obtain a hard clustering, objects are often then assigned to the Gaussian distribution they most likely belong to; for soft clusterings, this is not necessary.

Distribution-based clustering is a semantically strong method, as it not only provides you with clusters, but also produces complex models for the clusters that can also capture correlation and dependence of attributes. However, using these algorithms puts an extra burden on the user: to choose appropriate data models to optimize, and for many real data sets, there may be no mathematical model available the algorithm is able to optimize (e.g. assuming Gaussian distributions is a rather strong assumption on the data).



2.3.4 Density-based clustering

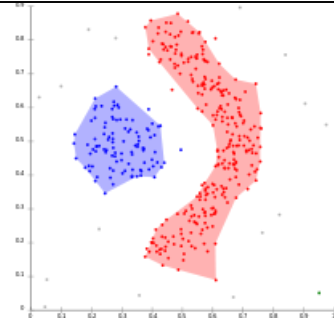
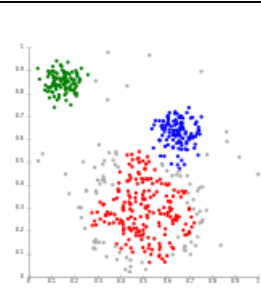
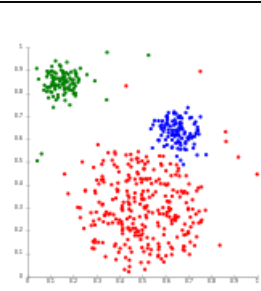
In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas - that are required to separate clusters - are usually considered to be noise and border points.

The most popular density based clustering method is DBSCAN. In contrast to many newer methods, it features a well-defined cluster model called "density-reachability". Similar to linkage based clustering, it is based on connecting points within certain distance thresholds. However, it only connects points that satisfy a density criterion, in the original variant defined as a minimum number

of other objects within this radius. A cluster consists of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other methods) plus all objects that are within these objects' range. Another interesting property of DBSCAN is that its complexity is fairly low - it requires a linear number of range queries on the database - and that it will discover essentially the same results (it is deterministic for core and noise points, but not for border points) in each run, therefore there is no need to run it multiple times. OPTICS is a generalization of DBSCAN that removes the need to choose an appropriate value for the range parameter ϵ , and produces a hierarchical result related to that of linkage clustering. DeLi-Clu, Density-Link-Clustering combines ideas from single-linkage clustering and OPTICS, eliminating the ϵ parameter entirely and offering performance improvements over OPTICS by using an R-tree index.

The key drawback of DBSCAN and OPTICS is that they expect some kind of density drop to detect cluster borders. Moreover, they cannot detect intrinsic cluster structures which are prevalent in the majority of real life data. A variation of DBSCAN, EnDBSCAN, efficiently detects such kinds of structures. On data sets with, for example, overlapping Gaussian distributions - a common use case in artificial data - the cluster borders produced by these algorithms will often look arbitrary, because the cluster density decreases continuously. On a data set consisting of mixtures of Gaussians, these algorithms are nearly always outperformed by methods such as EM clustering that are able to precisely model this kind of data.

Density-based clustering examples

| | | |
|---|--|---|
|  <p>Density-based clustering with DBSCAN.</p> <p>2.3.5</p> |  <p>DBSCAN assumes clusters of similar density, and may have problems separating nearby clusters</p> |  <p>OPTICS is a DBSCAN variant that handles different densities much better</p> <p>2.3.6</p> |
|---|--|---|

2.4 Recent developments

In recent years considerable effort has been put into improving algorithm performance of the existing algorithms. Among them are *CLARANS* and *BIRCH*. With the recent need to process larger and larger data sets (also known as big data), the willingness to trade semantic meaning of the generated clusters for performance has been increasing. This led to the development of pre-clustering methods such as canopy clustering, which can process huge data sets efficiently, but the resulting "clusters" are merely a rough pre-partitioning of the data set to then analyze the partitions with existing slower methods such as k-means clustering. Various other approaches to clustering have been tried such as seed based clustering.

For high-dimensional data, many of the existing methods fail due to the curse of dimensionality, which renders particular distance functions problematic in high-dimensional spaces. This led to new clustering algorithms for high-dimensional data that focus on subspace clustering (where only some attributes are used, and cluster models include the relevant attributes for the cluster) and correlation clustering that also looks for arbitrary rotated ("correlated") subspace clusters that can be modeled by

giving a correlation of their attributes. Examples for such clustering algorithms are CLIQUE and SUBCLU.

Ideas from density-based clustering methods (in particular the DBSCAN/OPTICS family of algorithms) have been adopted to subspace clustering (HiSC, hierarchical subspace clustering and DiSH) and correlation clustering (HiCO, hierarchical correlation clustering, 4C using "correlation connectivity" and ERiC exploring hierarchical density-based correlation clusters).

Several different clustering systems based on mutual information have been proposed. One is Marina Meilă's *variation of information* metric; another provides hierarchical clustering. Using genetic algorithms, a wide range of different fit-functions can be optimized, including mutual information. Also message passing algorithms, a recent development in Computer Science and Statistical Physics, has led to the creation of new types of clustering algorithms.

2.5 Applications

- **Plant and animal ecology**
cluster analysis is used to describe and to make spatial and temporal comparisons of communities (assemblages) of organisms in heterogeneous environments; it is also used in plant systematics to generate artificial phylogenies or clusters of organisms (individuals) at the species, genus or higher level that share a number of attributes
- **Transcriptomics**
clustering is used to build groups of genes with related expression patterns (also known as coexpressed genes). Often such groups contain functionally related proteins, such as enzymes for a specific pathway, or genes that are co-regulated. High throughput experiments using expressed sequence tags (ESTs) or DNA microarrays can be a powerful tool for genome annotation, a general aspect of genomics.
- **Sequence analysis**
clustering is used to group homologous sequences into gene families. This is a very important concept in bioinformatics, and evolutionary biology in general. See evolution by gene duplication.
- **High-throughput genotyping platforms**
clustering algorithms are used to automatically assign genotypes.
- **Human genetic clustering**
The similarity of genetic data is used in clustering to infer population structures.
- **Medical imaging**
On PET scans, cluster analysis can be used to differentiate between different types of tissue and blood in a three-dimensional image. In this application, actual position does not matter, but the [voxel](#) intensity is considered as a vector, with a dimension for each image that was taken over time. This technique allows, for example, accurate measurement of the rate a radioactive tracer is delivered to the area of interest, without a separate sampling of arterial blood, an intrusive technique that is most common today.
- **IMRT segmentation**
Clustering can be used to divide a fluence map into distinct regions for conversion into deliverable fields in MLC-based Radiation Therapy.

3 Competitive learning

Competitive learning is a form of unsupervised learning in artificial neural networks, in which nodes compete for the right to respond to a subset of the input data. A variant of Hebbian learning, competitive learning works by increasing the specialization of each node in the network. It is well suited to finding clusters within data.

Models and algorithms based on the principle of competitive learning include vector quantization and self-organising maps (Kohonen maps).

3.1 Architecture and implementation

Competitive Learning is usually implemented with Neural Networks that contain a hidden layer which is commonly known as "competitive layer". Every competitive neuron i is described by a

vector of weights $\mathbf{w}_i = (w_{i1}, \dots, w_{id})^T, i = 1, \dots, M$ and calculates the similarity measure between the input data $\mathbf{x}^n = (x_{n1}, \dots, x_{nd})^T \in \mathbb{R}^d$ and the weight vector \mathbf{w}_i .

For every input vector, the competitive neurons “compete” with each other to see which one of them is the most similar to that particular input vector. The winner neuron m sets its output $o_i = 1$ and all the other competitive neurons set their output $o_i = 0, i = 1, \dots, M, i \neq m$.

Usually, in order to measure similarity the inverse of the Euclidean distance is used: $\|\mathbf{x} - \mathbf{w}_i\|$ between the input vector \mathbf{x}^n and the weight vector \mathbf{w}_i .

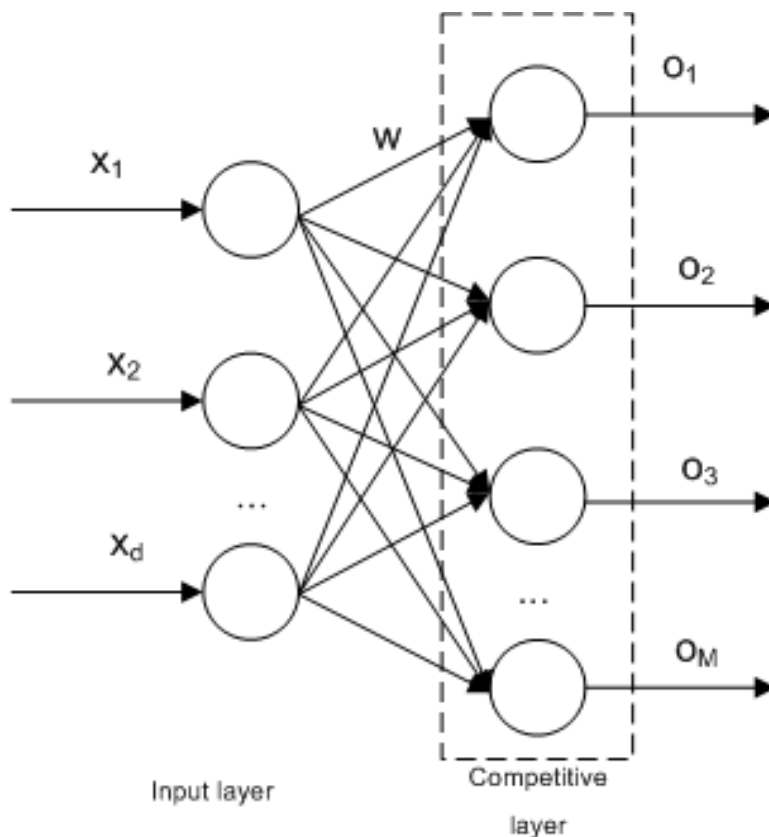


Figure 1 Competitive neural network architecture

3.2 Example algorithm

Here is a simple competitive learning algorithm to find three clusters within some input data.

1. (Set-up.) Let a set of sensors all feed into three different nodes, so that every node is connected to every sensor. Let the weights that each node gives to its sensors be set randomly between 0.0 and 1.0. Let the output of each node be the sum of all its sensors, each sensor's signal strength being multiplied by its weight.
2. When the net is shown an input, the node with the highest output is deemed the winner. The input is classified as being within the cluster corresponding to that node.
3. The winner updates each of its weights, moving weight from the connections that gave it weaker signals to the connections that gave it stronger signals.

Thus, as more data are received, each node converges on the centre of the cluster that it has come to represent and activates more strongly for inputs in this cluster and more weakly for inputs in other clusters.

4 Kohonen Network

The Kohonen neural network differs considerably from the feedforward back propagation neural network. The Kohonen neural network differs both in how it is trained and how it recalls a pattern. The Kohonen neural network does not use any sort of activation function. Further, the Kohonen neural network does not use any sort of a bias weight.

Output from the Kohonen neural network does not consist of the output of several neurons. When a pattern is presented to a Kohonen network one of the output neurons is selected as a "winner". This "winning" neuron is the output from the Kohonen network. Often these "winning" neurons represent groups in the data that is presented to the Kohonen network. For example, in Chapter 7 we will examine an OCR program that uses 26 output neurons. These 26 output neurons map the input patterns into the 26 letters of the Latin alphabet.

The most significant difference between the Kohonen neural network and the feed forward back propagation neural network is that the Kohonen network trained in an unsupervised mode. This means that the Kohonen network is presented with data, but the correct output that corresponds to that data is not specified. Using the Kohonen network this data can be classified into groups. We will begin our review of the Kohonen network by examining the training process.

It is also important to understand the limitations of the Kohonen neural network. You will recall from the previous chapter that neural networks with only two layers can only be applied to linearly separable problems. This is the case with the Kohonen neural network. Kohonen neural networks are used because they are a relatively simple network to construct that can be trained very rapidly.

4.1 How a Kohonen Network Recognizes

I will now show you how the Kohonen neural network recognizes a pattern. We will begin by examining the structure of the Kohonen neural network. Once you understand the structure of the Kohonen neural network, and how it recognizes patterns, you will be shown how to train the Kohonen neural network to properly recognize the patterns you desire. We will begin by examining the structure of the Kohonen neural network.

4.2 The Structure of the Kohonen Neural Network

The Kohonen neural network works differently than the feed forward neural network that we learned about in Chapter 5. The Kohonen neural network contains only an input and output layer of neurons. There is no hidden layer in a Kohonen neural network. First we will examine the input and output to a Kohonen neural network.

The input to a Kohonen neural network is given to the neural network using the input neurons. These input neurons are each given the floating point numbers that make up the input pattern to the network. A Kohonen neural network requires that these inputs be normalized to the range between -1 and 1. Presenting an input pattern to the network will cause a reaction from the output neurons.

The output of a Kohonen neural network is very different from the output of a feed forward neural network. If we had a neural network with five output neurons we would be given an output that consisted of five values. This is not the case with the Kohonen neural network. In a Kohonen neural

network only one of the output neurons actually produces a value. Additionally, this single value is either true or false. When the pattern is presented to the Kohonen neural network, one single output neuron is chosen as the output neuron. Therefore, the output from the Kohonen neural network is usually the index of the neuron (i.e. Neuron #5) that fired. The structure of a typical Kohonen neural network is shown in Figure 6.1.

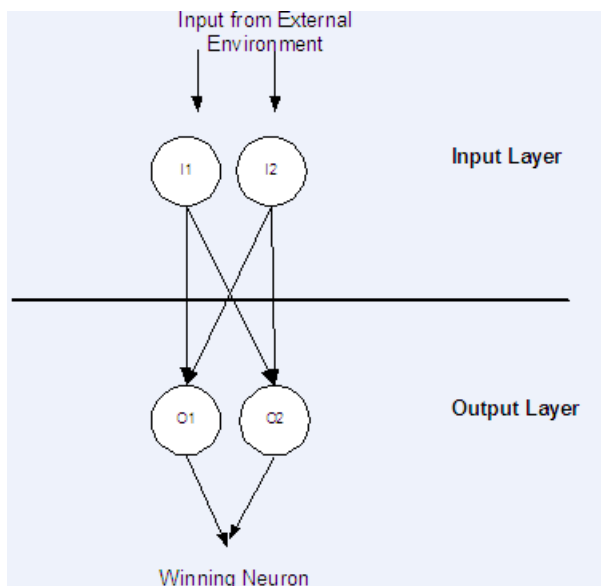


Figure 4.1: A Kohonen Neural Network

Now that you understand the structure of the Kohonen neural network we will examine how the network processes information. To examine this process we will step through the calculation process. For this example we will consider a very simple Kohonen neural network. This network will have only two input and two output neurons. The input given to the two input neurons is shown in Table 6.1.

Table 4.1: Sample Inputs to a Kohonen Neural Network

| | |
|---------------------|------|
| Input Neuron 1 (I1) | 0.5 |
| Input Neuron 2 (I2) | 0.75 |

We must also know the connection weights between the neurons. These connection weights are given in Table 4.2.

Table 4.2: Connection weights in the sample Kohonen neural network

| | |
|--------|-----|
| I1->O1 | 0.1 |
| I2->O1 | 0.2 |
| I1->O2 | 0.3 |
| I2->O2 | 0.4 |

Using these values we will now examine which neuron would win and produce output. We will begin by normalizing the input.

4.3 Normalizing the Input

The Kohonen neural network requires that its input be normalized. Because of this some texts refer to the normalization as a third layer. For the purposes of this book the Kohonen neural network is

considered a two layer network because there are only two actual neuron layers at work in the Kohonen neural network.

The requirements that the Kohonen neural network places on its input data is one of the most severe limitations of the Kohonen neural network. Input to the Kohonen neural network should be between the values -1 and 1. In addition, each of the inputs should fully use the range. If one, or more, of the input neurons were to use only the numbers between 0 and 1, the performance of the neural network would suffer.

To normalize the input we must first calculate the "vector length" of the input data, or vector. This is done by summing the squares of the input vector. In this case it would be.

$$(0.5 * 0.5) + (0.75 * 0.75)$$

This would result in a "vector length" of 0.8125. If the length becomes too small, say less than the length is set to that same arbitrarily small value. In this case the "vector length" is a sufficiently large number. Using this length we can now determine the normalization factor. The normalization factor is the reciprocal of the square root of the length. For our value the normalization factor is calculated as follows.

$$\frac{1}{\sqrt{0.8125}}$$

This results in a normalization factor of 1.1094. This normalization process will be used in the next step where the output layer is calculated.

4.4 Calculating Each Neuron's Output

To calculate the output the input vector and neuron connection weights must both be considered. First the "dot product" of the input neurons and their connection weights must be calculated. To calculate the dot product between two vectors you must multiply each of the elements in the two vectors. We will now examine how this is done.

The Kohonen algorithm specifies that we must take the dot product of the input vector and the weights between the input neurons and the output neurons. The result of this is as follows.

$$|0.5 \ 0.75| \bullet |0.1 \ 0.2| = (0.5 * 0.75) + (0.1 * 0.2)$$

As you can see from the above calculation, the dot product would be 0.395. This calculation will be performed for the first output neuron. This calculation will have to be done for each of the output neurons. Through this example we will only examine the calculations for the first output neuron. The calculations necessary for the second output neuron are calculated in the same way.

This output must now be normalized by multiplying it by the normalization factor that was determined in the previous step. You must now multiply the dot product of 0.395 by the normalization factor of 1.1094. This results in an output of 0.438213. Now that the output has been calculated and normalized it must be mapped to a bipolar number.

4.5 Mapping to Bipolar

As you may recall from Chapter 2 a bipolar number is an alternate way of representing binary numbers. In the bipolar system the binary zero maps to -1 and the binary remains a 1. Because the input to the neural network normalized to this range we must perform a similar normalization to the

output of the neurons. To make this mapping we add one and divide the result in half. For the output of 0.438213 this would result in a final output of 0.7191065.

The value 0.7191065 is the output of the first neuron. This value will be compared with the outputs of the other neuron. By comparing these values we can determine a "winning" neuron.

4.6 Choosing the Winner

We have seen how to calculate the value for the first output neuron. If we are to determine a winning output neuron we must also calculate the value for the second output neuron. We will now quickly review the process to calculate the second neuron. For a more detailed description you should refer to the previous section.

The second output neuron will use exactly the same normalization factor as was used to calculate the first output neuron. As you recall from the previous section the normalization factor is 1.1094. If we apply the dot product for the weights of the second output neuron and the input vector we get a value of 0.45. This value is multiplied by the normalization factor of 1.1094 to give the value of 0.0465948. We can now calculate the final output for neuron 2 by converting the output of 0.0465948 to bipolar yields 0.49923.

As you can see we now have an output value for each of the neurons. The first neuron has an output value of 0.7191065 and the second neuron has an output value of 0.49923. To choose the winning neuron we choose the output that has the largest output value. In this case the winning neuron is the first output neuron with an output of 0.7191065, which beats neuron two's output of 0.49923.

You have now seen how the output of the Kohonen neural network was derived. As you can see the weights between the input and output neurons determine this output. In the next section we will see how you can adjust these weights can be adjusted to produce output that is more suitable for the desired task. The training process is what modified these weights. The training process will be described in the next section.

4.7 How a Kohonen Network Learns

In this section you will learn to train a Kohonen neural network. There several steps involved in this training process. Overall the process for training a Kohonen neural network involves stepping through several epochs until the error of the Kohonen neural network is below acceptable level. In this section we will learn these individual processes. You'll learn how to calculate the error rate for Kohonen neural network, you'll learn how to adjust the weights for each epoch. You will also learn to determine when no more epochs are necessary to further train the neural network.

The training process for the Kohonen neural network is competitive. For each training set one neuron will "win". This winning neuron will have its weight adjusted so that it will react even more strongly to the input the next time. As different neurons win for different patterns, their ability to recognize that particular pattern will be increased.

The learning process is as roughly as follows:

- initialise the weights for each output unit
- loop until weight changes are negligible
 - for each input pattern
 - present the input pattern
 - find the winning output unit
 - find all units in the neighbourhood of the winner

- update the weight vectors for all those units
- reduce the size of neighbourhoods if required

The winning output unit is simply the unit with the weight vector that has the smallest Euclidean distance to the input pattern. The neighbourhood of a unit is defined as all units within some distance of that unit on the map (not in weight space). In the demonstration below all the neighbourhoods are square. If the size of the neighbourhood is 1 then all units no more than 1 either horizontally or vertically from any unit fall within its neighbourhood. The weights of every unit in the neighbourhood of the winning unit (including the winning unit itself) are updated using

$$\vec{w}_i = \vec{w}_i + \alpha (\vec{x}_i - \vec{w}_i) \quad (21)$$

This will move each unit in the neighbourhood closer to the input pattern. As time progresses the learning rate and the neighbourhood size are reduced. If the parameters are well chosen the final network should capture the natural clusters in the input data.

4.8 Learning Rate

The learning rate is a constant that will be used by the learning algorithm. The learning rate must be a positive number less than 1. Typically the learning rate is a number such as .4 or .5. In the following section the learning rate will be specified by the symbol alpha.

Generally setting the learning rate to a larger value will cause the training to progress faster. Though setting the learning rate to too large a number could cause the network to never converge. This is because the oscillations of the weight vectors will be too great for the classification patterns to ever emerge. Another technique is to start with a relatively high learning rate and decrease this rate as training progresses. This allows initial rapid training of the neural network that will be "fine tuned" as training progresses.

The learning rate is just a variable that is used as part of the algorithm used to adjust the weights of the neurons. In the next section we will see how these weights are adjusted using the learning rate.

4.9 Adjusting Weights

The entire memory of the Kohonen neural network is stored inside of the weighted connections between the input and output layer. The weights are adjusted in each epoch. An epoch occurs when training data is presented to the Kohonen neural network and the weights are adjusted based on the results of this item of training data. The adjustments to the weights should produce a network that will yield more favorable results the next time the same training data is presented. Epochs continue as more and more data is presented to the network and the weights are adjusted.

Eventually the return on these weight adjustments will diminish to the point that it is no longer valuable to continue with this particular set of weights. When this happens the entire weight matrix is reset to new random values. This forms a new cycle. The final weight matrix that will be used will be the best weight matrix determined from each of the cycles. We will now examine how these weights are transformed.

The original method for calculating the changes to weights, which was proposed by Kohonen, is often called the additive method. This method uses the following equation.

$$w^{t+1} = \frac{w^t + cx}{\|w^t + cx\|}$$

The variable x is the training vector that was presented to the network. The variable w^t is the weight of the winning neuron, and the variable w^{t+1} is the new weight. The double vertical bars represent the vector length. This method will be implemented in the Kohonen example shown later in this chapter.

The additive method generally works well for Kohonen neural networks. Though in cases where the additive method shows excessive instability, and fails to converge, an alternate method can be used. This method is called the subtractive method. The subtractive method uses the following equations.

$$e = x - w^t$$

$$w^{t+1} = w^t + ce$$

These two equations show you the basic transformation that will occur on the weights of the network. In the next section you will see how these equations are implemented as a Java program, and their use will be demonstrated.

4.9.1 Calculating the Error

Before we can understand how to calculate the error for chronic neural network must first understand what the error means. The coming neural network is trained in an unsupervised fashion so the definition of the error is somewhat different involving the normally think of as an error.

As your alternate previous Chapter unsupervised training involved Calculating and error which was the difference between the anticipated output of the neural network and the actual output of the neural network. In this chapter we are examining unsupervised training. And unsupervised training there is no anticipated output. Because of this you may be wondering exactly how we can calculate an error. The answer is that the error where Calculating is not be true error, or at least not an error in the normal sense of the word.

The purpose of the Kohonen neural network is to classify the input into several sets. The error for the Kohonen neural network, therefore, must be able to measure how well the network is classifying these items. We will examine two methods for determining the error in this section. There is no official way to calculate the error for a Kohonen neural network. The error is just a percent number that gives an idea of how well the Kohonen network is classifying the input into the output groups. The error itself is not used to modify the weights, as was done in the back propagation algorithm. The method to determine this error will be discussed when we see how to implement a Java training method.

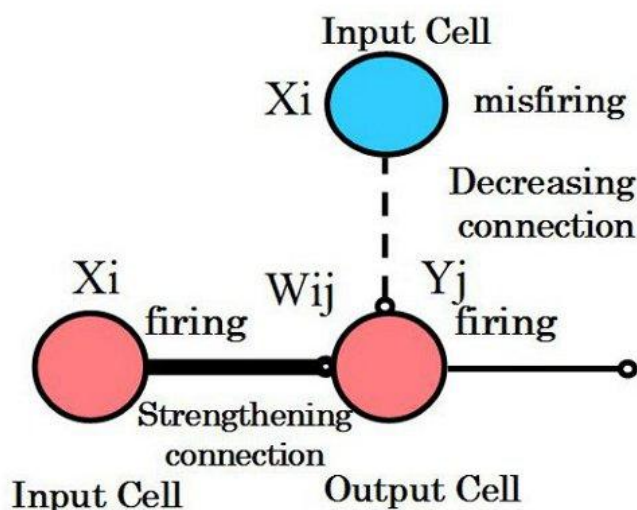
5 Hebbian learning

Hebbian theory is a theory in neuroscience, which proposes an explanation for the adaptation of neurons in the brain during the learning process. It describes a basic mechanism for synaptic plasticity, where an increase in synaptic efficacy arises from the presynaptic cell's repeated and persistent stimulation of the postsynaptic cell. Introduced by Donald Hebb in his 1949 book *The Organization of Behavior*, the theory is also called **Hebb's rule**, **Hebb's postulate**, and **cell assembly theory**. Hebb states it as follows:

"Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability.... When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

The theory is often summarized as "Cells that fire together, wire together". However, this summary should not be taken literally. Hebb emphasized that cell A needs to 'take part in firing' cell B, and such causality can only occur if cell A fires just before, not at the same time as, cell B. This important aspect of causation in Hebb's work foreshadowed what we now know about spike-timing-dependent plasticity, which requires temporal precedence. The theory attempts to explain associative or **Hebbian learning**, in which simultaneous activation of cells leads to pronounced increases in synaptic strength between those cells, and provides a biological basis for errorless learning methods for education and memory rehabilitation.

$$\Delta W_{ij} = \eta X_i \cdot Y_j$$



ΔW_{ij} is the strength of the change in synaptic weight
 X_i is the output of the input cell
 Y_j is the output of the output cell
 η is the learning coefficient

5.1 Hebbian Learning

The Hebbian Learning Rule is a learning rule that specifies how much the weight of the connection between two units should be increased or decreased in proportion to the product of their activation.

Hebbian learning is one of the oldest learning algorithms, and is based in large part on the dynamics of biological systems. A synapse between two neurons is strengthened when the neurons on either side of the synapse (input and output) have highly correlated outputs. In essence, when an input neuron fires, if it frequently leads to the firing of the output neuron, the synapse is strengthened.

Following the analogy to an artificial system, the tap weight is increased with high correlation between two sequential neurons.

5.2 Mathematical Formulation

Mathematically, we can describe Hebbian learning as:

$$w_{ij}[n + 1] = w_{ij}[n] + \eta x_i[n] x_j[n]$$

Here, η is a learning rate coefficient, and x are the outputs of the i th and j th elements.

5.3 Plausibility

The Hebbian learning algorithm is performed locally, and doesn't take into account the overall system input-output characteristic. This makes it a plausible theory for biological learning methods, and also makes Hebbian learning processes ideal in VLSI hardware implementations where local signals are easier to obtain.

6 Adaptive resonance theory (ART)

6.1 Introduction

One of the nice features of human memory is its ability to learn many new things without necessarily forgetting things learned in the past. A frequently cited example is the ability to recognize your parents even if you have not seen them for some time and have learned many new faces in the interim. It would be highly desirable if we could impart this same capability to an Artificial Neural Networks. Most neural networks will tend to forget old information if we attempt to add new information incrementally. When developing an artificial neural network to perform a particular pattern-classification operation, we typically proceed by gathering a set of exemplars, or training patterns, then using these exemplars to train the system.

During the training, information is encoded in the system by the adjustment of weight values. Once the training is deemed to be adequate, the system is ready to be put into production, and no additional weight modification is permitted.

This operational scenario is acceptable provided the problem domain has well-defined boundaries and is stable. Under such conditions, it is usually possible to define an adequate set of training inputs for whatever problem is being solved. Unfortunately, in many realistic situations, the environment is neither bounded nor stable.

Consider a simple example. Suppose you intend to train a backpropagation to recognize the silhouettes of a certain class of aircraft. The appropriate images can be collected and used to train the network, which is potentially a time-consuming task depending on the size of the network required. After the network has learned successfully to recognize all of the aircraft, the training period is ended and no further modification of the weights is allowed.

If, at some future time, another aircraft in the same class becomes operational, you may wish to add its silhouette to the store of knowledge in your neural network. To do this, you would have to retrain the network with the new pattern plus all of the previous patterns. Training on only the new silhouette could result in the network learning that pattern quite well, but forgetting previously

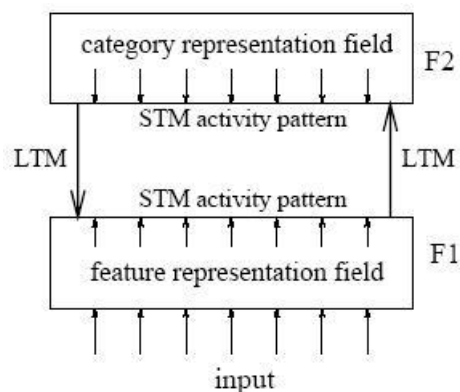
learned patterns. Although retraining may not take as long as the initial training, it still could require a significant investment.

6.2 The Adaptive Resonance Theory: ART

In 1976, Grossberg (Grossberg, 1976) introduced a model for explaining biological phenomena. The model has three crucial properties:

1. a normalisation of the total network activity. Biological systems are usually very adaptive to large changes in their environment. For example, the human eye can adapt itself to large variations in light intensities;
2. contrast enhancement of input patterns. The awareness of subtle differences in input patterns can mean a lot in terms of survival. Distinguishing a hiding panther from a resting one makes all the difference in the world. The mechanism used here is contrast enhancement;
3. short-term memory (STM) storage of the contrast-enhanced pattern. Before the input pattern can be decoded, it must be stored in the short-term memory. The long-term memory (LTM) implements an arousal mechanism (i.e., the classification), whereas the STM is used to cause gradual changes in the LTM.

The system consists of two layers, F1 and F2, which are connected to each other via the LTM

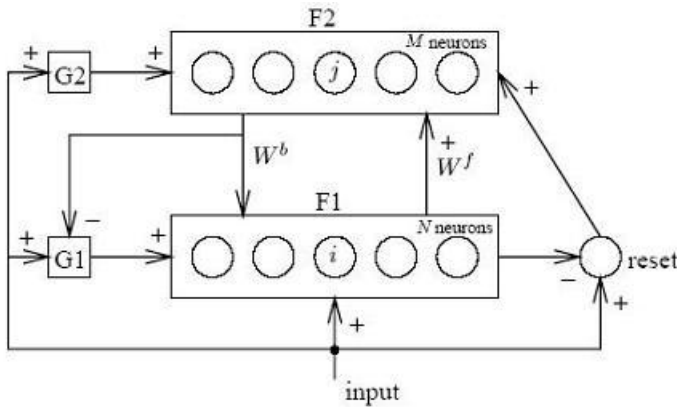


The ART architecture.

The input pattern is received at F1, whereas classification takes place in F2. As mentioned before, the input is not directly classified. First a characterization takes place by means of extracting features, giving rise to activation in the feature representation field. The expectations, residing in the LTM connections, translate the input pattern to a categorization in the category representation field. The classification is compared to the expectation of the network, which resides in the LTM weights from F2 to F1. If there is a match, the expectations are strengthened, otherwise the classification is rejected.

6.2.1 ART1: The simplified neural network model

The ART1 simplified model consists of two layers of binary neurons (with values 1 and 0), called F1 (the comparison layer) and F2 (the recognition layer)



The ART1 neural network.

Each neuron in F1 is connected to all neurons in F2 via the continuous-valued forward long term memory (LTM) W^f , and vice versa via the binary-valued backward LTM W^b . The other modules are gain 1 and 2 (G1 and G2), and a reset module.

Each neuron in the comparison layer receives three inputs: a component of the input pattern, a component of the feedback pattern, and a gain G1. A neuron outputs a 1 if and only if at least three of these inputs are high: the ‘two-thirds rule.’ The neurons in the recognition layer each compute the inner product of their incoming (continuous-valued) weights and the pattern sent over these connections. The winning neuron then inhibits all the other neurons via lateral inhibition.

Gain 2 is the logical ‘or’ of all the elements in the input pattern x . Gain 1 equals gain 2, except when the feedback pattern from F2 contains any 1; then it is forced to zero. Finally, the reset signal is sent to the active neuron in F2 if the input vector x and the output of F1 differ by more than some vigilance level.

6.2.2 Operation

The network starts by clamping the input at F1. Because the output of F2 is zero, G1 and G2 are both on and the output of F1 matches its input. The pattern is sent to F2, and in F2 one neuron becomes active. This signal is then sent back over the backward LTM, which reproduces a binary pattern at F1. Gain 1 is inhibited, and only the neurons in F1 which receive a ‘one’ from both x and F2 remain active. If there is a substantial mismatch between the two patterns, the reset signal will inhibit the neuron in F2 and the process is repeated.

1. Initialisation:

$$w_{ji}^b(0) = 1$$

$$w_{ij}^f(0) = \frac{1}{1 + N}$$

where N is the number of neurons in F1, M the number of neurons in F2, $0 < i < N$,

and $0 \leq j < M$

2. Apply the new input pattern x :

$$y_i^f = \sum_{j=1}^M w_{ij}^f(t) x_j;$$

3. compute the activation values y_0 of the neurons in F2:
4. select the winning neuron k ($0 \leq k < m$): $k = \text{argmax}_k y_k^f$
5. vigilance test: if

$$\frac{w_k^b(t) \cdot x}{x \cdot x} > \rho,$$

where \cdot denotes inner product, go to step 7, else go to step 6. Note that $w_k^b \cdot x$ essentially is the inner product $x^* \cdot x$, which will be large if x^* and x near to each other;

6. neuron k is disabled from further activity. Go to step 3;
7. Set for all l, $0 \leq l < N$:

$$w_{kl}^b(t+1) = w_{kl}^b(t) x_l,$$

$$w_{lk}^f(t+1) = \frac{w_{kl}^b(t) x_l}{\frac{1}{2} + \sum_{i=1}^N w_{ki}^b(t) x_i};$$

8. re-enable all neurons in F2 and go to step 2.

backward LTM from:

| input pattern | output 1 | output 2 | output 3 | output 4 |
|---------------|----------|------------|------------|------------|
| C | C | not active | not active | not active |
| E | C | E | not active | not active |
| F | C | E | F | not active |
| F | C | E | F | not active |
| F | C | E | F | F |

An example of the behaviour of the Carpenter Grossberg network for letter patterns. The binary input patterns on the left were applied sequentially. On the right the stored patterns (i.e., the weights of W_b for the first four output units) are shown.

6.2.3 ART1: The original model

In later work, Carpenter and Grossberg (Carpenter & Grossberg, 1987a, 1987b) present several neural network models to incorporate parts of the complete theory. We will only discuss the first model, ART1.

The network incorporates a follow-the-leader clustering algorithm (Hartigan, 1975). This algorithm tries to fit each new input pattern in an existing class. If no matching class can be found, i.e., the distance between the new pattern and all existing classes exceeds some threshold, a new class is created containing the new pattern.

The novelty in this approach is that the network is able to adapt to new incoming patterns, while the previous memory is not corrupted. In most neural networks, such as the backpropagation network, all patterns must be taught sequentially; the teaching of a new pattern might corrupt the weights for all previously learned patterns. By changing the structure of the network rather than the weights, ART1 overcomes this problem.

6.2.4 ART Applications

- Adaptive Resonance Theory (ART) class of neural networks was used to detect and classify anomalies. Compared with other network types, ART networks are fast, efficient learners and retain memory while learning new patterns. Various ART networks were trained using simulation, and tested in the field using the testbed. Using this technology, it will be possible to improve the reliability of Bio Medical systems with little or no additional sensing equipment compared to typical installations.
- ART neural networks are capable to perform efficient classification and to recognize new states when necessary, they seems to be a proper tool for classification of operational states in various medical systems.
- To monitor medical diagnosis systems and detect existing or developing faults.