4. ANN models and Learning algorithms for biomedical signal processing

Chapter 4
ANN models and Learning algorithms for biomedical signal processing

4.1 Keywords

- **Neural Network:**

  A human brain has an ability to make decisions when presented with complex, noisy, irrelevant or partial information. Neural network can be defined as an information processing system inspired by the structure of brain.

- **Neural computing:**

  It is a new approach to information processing. It is fast and vital alternative to normal sequential processing. The large power lies in parallel processing architecture. It is a domain in which attempts are made to make computer think, react and compute.

- **Neural Processing:**

  Analogous to the read/write operation in computer systems, there are two phases of neural processing:

  I. **Learning/Training Phase:** A data set is used to determine weights that define Neural network which learns by adaptively updating the synaptic weights.
  II. **Retrieving Phase:** The trained model uses the result to process real test pattern and yield results.

- **Learning:**

  The artificial neural network produces response based on information encoded in its structure. Usually weights on the interconnections between the neurons, store the information. The weights are adjusted to produce the desired response.

  There exist two primary types of neural network learning:

I. **Supervised learning:** Supervised learning is a process of training a neural network by giving it examples of the task we want it to learn. i.e., learning with a teacher. The way this is done is by providing a set of pairs of vectors (patterns), where the first pattern of each pair is an example of an input pattern that the network might have to process and the second pattern is the output pattern that the network should produce for that input which is known as a target

output pattern for whatever input pattern. This technique is mostly applied to feed forward type of neural networks.

II. **Unsupervised learning**: Unsupervised learning is a process when the network is able to discover statistical regularities in its input space and automatically develops. Different modes of behavior to represent different classes of inputs (in practical applications some 'labeling' is required after training, since it is not known at the outset which mode of behavior will be associated with a given input class)? Kohonen's self -organizing (topographic) map neural networks use this type of learning.

- **Threshold:**

It is used to restrict the neuron output to meaningful value. It effectively adds an off set to the weighted sum.

- **BASIS Function:**

It describes functional description of connection network.. i.e. a mapping of input to produce output. For linear mapping, Linear Basis Functions (LBF) is used while for nonlinear mapping, Radial Basis Function (RBF) is used.

- **Solution Approach:**

This is used to decide how many networks should be used for multi category classification. Typically one output node is used to represent one class. Two approaches are: All Class One Network (ACON) and One Class One Network (OCON).

4.2 Introduction

Neural networks are one of a group of intelligence technologies for data analysis that differ from other classical analysis techniques by learning about chosen subject from the data provided, rather than being programmed by the user in a traditional sense. Neural networks gather their knowledge by detecting the patterns and relationships in our data.

Building a software application is very much easier if we can break the function of the application into smaller functions that we can easily implement. It is also beneficial to have general-purpose tools that we can connect together to produce the application. In many applications a neural network can form one of these blocks. The range of functions that neural nets can perform is very large, and neural nets can only perform some of these functions. In building an application, the standard method is to break the overall function into blocks of code. These blocks receive input data from outside

the system (user input) or from other blocks, perform some processing depending on the code written into the block and then output data to other blocks or to outside the system (system output).

The processing or procedure that is written into each block is often called an algorithm. Programmers have access to many standard algorithms, which they can incorporate into their programs for such tasks as computing the difference between two dates or converting one currency into another. However, sometimes there is no algorithm available for a functional block, and creating one may be very time consuming.

A key benefit of neural networks is that they can be used to build a model of the system or subject of interest from just the data provided. In the situations where the inputs and outputs that are known, but what happens internally is not known, the neural network can model this system from the data. Neural nets are powerful solutions to such problems.

The characteristics of neural networks are massively parallel structure, a high degree of interconnections, the propensity for storing experiential knowledge and the capabilities of high speed computation, nonlinear mapping, and self organization. The high-speed computational capabilities of neural networks can be used to solve implement real time problem.

In the formative years of neural networks (1943-1958), several researchers stand out for their pioneering contributions:

- McCulloch and Pitts (1943) for introducing the idea of neural networks as computing machines.
- Hebb (1949) for postulating the first rule for self-organized learning.
- Rosenblatt (1958) for proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).

In 1970s, Grossberg developed his Adaptive Resonance Theory (ART), a number of hypotheses about the underlying principles governing biological neural systems [1]. These ideas served as the basis for later work by Carpenter and Grossberg involving three classes of ART architectures: ART 1 [2], ART2 [3] and ART3 [4]. These are self organizing neural network implementations of pattern clustering algorithms. Other

important theory on self organizing systems was pioneered by Kohonen with his work on feature maps [5], [6].

In the 1980s, Hopfield and other introduced outer product rules as well as equivalent approaches based on the early work of Hebb [7] for training a class of recurrent networks called Hopfield models now a days [8], [9]. Kosko expanded some of the ideas of Hopfield and Grossberg to develop his adaptive Bidirectional Associative Memory (BAM) [10], a network model-employing differential as well as Hebbian and competitive learning laws.

4.3 Network Architectures

The ANN architectures are summarized in **Table 4.1.**

Table 4.1 Neural Network Architectures

Supervised	Unsupervised	Fixed Weights	Advance Networks
• Perceptron • Decision Based (DBNN) • ADALINE • MLP • Hidden Markov • Temporal Dynamic Model	• Neo-recognition • Feature Map • CPL • ARN • Principal Component	• Hamming Net • Hopfield Net • Combinatorial Optimization	• Probabilistic NN • General Regression NN

The ANN can be also be classified as Feed forward networks and Feed back networks.

4.3.1 Feed Forward Networks

Neural network which have property of forwarding signals in one direction without returning nor feeding them back are termed as feed forward networks. The neurons on a same layer are not connected to each other, they have connections coming into them from the neuron in the layer just below and connections going from them to the neurons on the next layer above.

4.3.1.1 Perceptron

It is the simplest form of a neural network used for the classification of patterns said to be linearly separable. It consists of a single neuron with adjustable synaptic weights and bias. The perceptron built around a single neuron is limited to performing pattern classification with only two classes (hypotheses). By expanding the output

(computation) layer of the perceptron to include more than one neuron, classification with more than two classes is possible.

The single neuron forms the basis of an adaptive filter. The LMS algorithm is simple to implement yet highly effective in application. For a dynamical system, the mathematical characterization of which is unknown, a set of labeled input-output data generated by the system at discrete instants of time at some uniform rate can be available. Specifically, when an m-dimensional stimulus

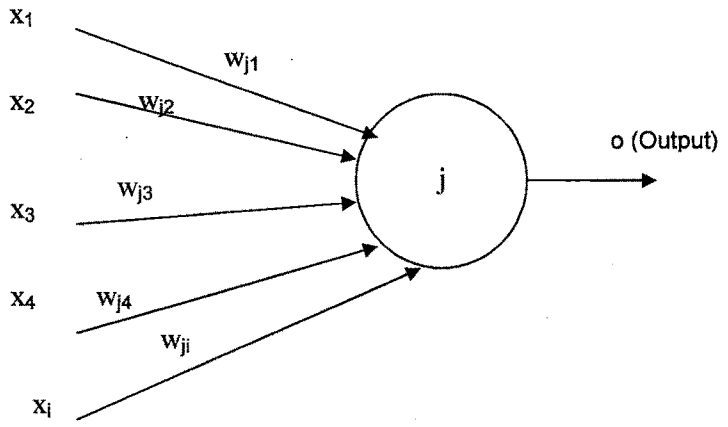


Figure 4.1 Single neuron

Since the neuron is linear, the output $y(i)$ is exactly the same as the induced local field $v(i)$; that is,

$$y(i) = v(i) = \sum_{k=1}^{m} wk(i)xk(i) \quad \text{.....(4.1)}$$

Where $W_1(i)$, $W_2(i)$, ..., $W_m(i)$ are the m synaptic weights of the neuron, measured at time i. In matrix form y(i) may be expressed as an inner product of the vectors x(i) and w(i):

$$Y(i) = x'(i)w(i) \quad \text{.....(4.2)}$$

Where

$$W(i) = [W_1(i), W_2(i), \ldots W_m(i)]^T$$

The neuron's output y(i) is compared to the corresponding output d(i) received from the unknown system at time i. Typically, y(i) is different from d(i); hence, their comparison results in the error signal.

$$e(i) = d(i) - y(i) \quad \text{.....(4.3)}$$

The manner in which the error signal e(i) is used to control the adjustments to the neuron's synaptic weights is determined by the **cost function** used to derive the adaptive filtering algorithm of interest.

4.3.1.2 Multilayer Perceptron

In general Neural Networks often have many layers, and usually all layers are adaptive. **Figure 4.2** depicts block diagram of MLP.

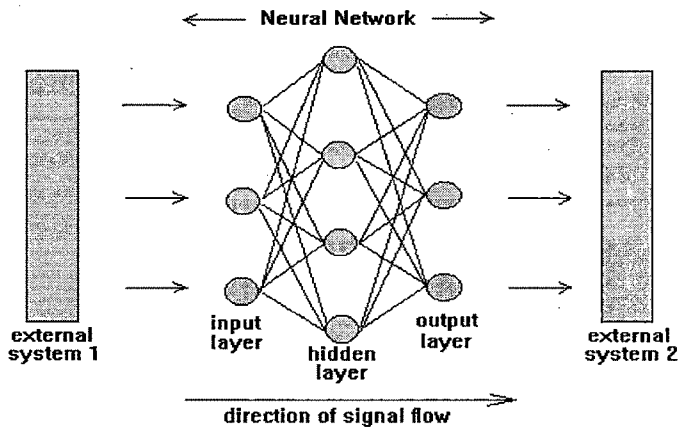


Figure 4.2 Block diagram: Multilayer Perceptron (MLP)

Figure 4.3 shows detailed structure of a fully connected 3-layer adaptive MLP.

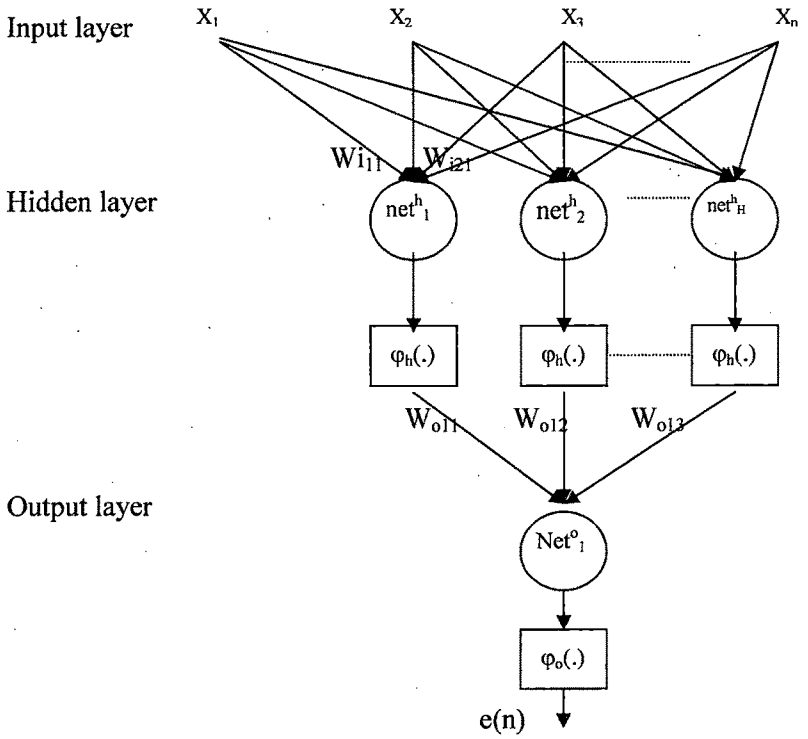


Figure 4.3 Structure: Multilayer Perceptron

4.3.1.3 RBF Neural Network

Radial Basis Function (RBF) Networks were introduced into the neural network literature by Broomhead and Lowe (1988) [18]. The RBF network model is motivated by the locally tuned response observed in biological neurons. Neurons with locally tuned response characteristics can be found in many parts of biologic nervous system. The radial threshold function is used.

The RBF network has a feed forward structure consisting of a single hidden layer of locally tuned units which are fully interconnected to an output layer of linear units as shown in **Figure 4.4.**

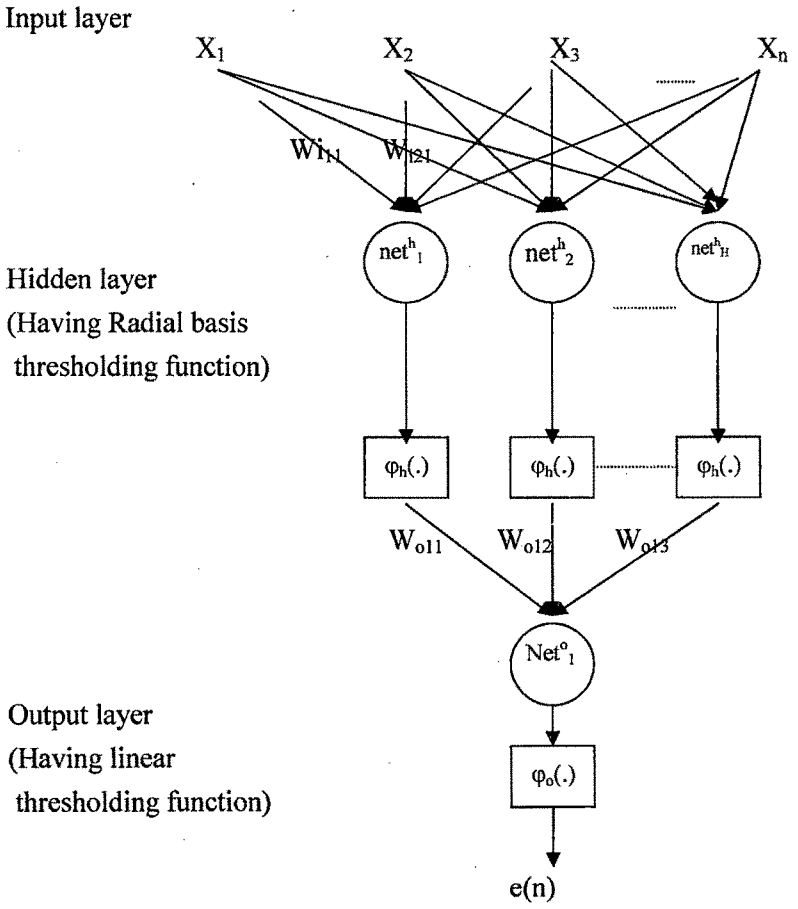


Figure 4.4 Structure of RBF ANN

These networks are best suited for approximating continuous or piecewise continuous real-valued mappings f: $R^n \rightarrow R^L$, where n is sufficiently small. All hidden units simultaneously receive the n-dimensional real valued input vector **x.** Each hidden-unit

output z_j is obtained by calculating the "closeness" of the input x of an n-dimensional parameter μ_j associated with the j^{th} hidden unit, the response is given by:

$$z_j(\mathbf{x}) = K\left(\frac{\|x-\mu_j\|}{\sigma j^2}\right) \quad \text{..... (4.4)}$$

where K is strictly positive radially symmetric function(kernel) with a unique maximum at its "centre" μ_j and drops off rapidly to zero away from centre. Parameter σ_j is the "width" of the respective field in the input space of unit j. Hence at small distance $\|x - \mu_j\|$ as compared to the width σ_j z_j will have appreciable value.

Output of the RBF network is the L dimensional activity vector y whose l^{th} component is given by

$$y_l(\mathbf{x}) = \sum_{j=1}^{J} w_{ij} \; z_j(x) \quad \text{..... (4.5)}$$

Commonly, RBF use Gaussian basis function for the hidden units:

$$z_j(\mathbf{x}) = \exp\left(\frac{\|X-\mu_j\|^2}{2\sigma j^2}\right) \quad \text{..... (4.6)}$$

where σ_j and μj are the standard deviation and mean of the j^{th} unit receptive field, respectively. Other possible choice for the basis function is logistic function. Training of RBF network may take different forms. In many applications, the first (RBF) layer and the second (combination weight) layer are trained separately. To improve further the performance, a combined training for the whole network just like back propagation has also been proposed (Poggio & Girosi, 1990a) which leads to a better performance.

4.3.2 Feedback Networks

Another architecture is Feedback neural network. The recurrent neural nets have a complete connectivity. There is neither layers layout nor difference between neurons. The most well known neural network with such architecture is the "Hopfield" network (**Figure 4.5**). Networks with this structure are also called associative networks.

The special features of the Hopfield networks are distributed representation and control. It ahs fault tolerance. The global minima can be achieved by simulated annealing.

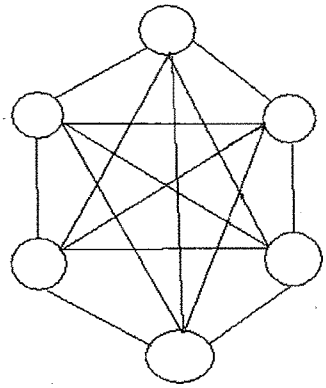


Figure 4.5 Hopfield Neural Network architecture

4.3.2.1 Recurrent Neural Networks

The fundamental feature of a recurrent network is that the network contains at least one feed-back connection, so activation can flow around in a loop. The architectures of recurrent networks can take many different forms. However, they all share the following common features:

1. They incorporate some form of static multi-layer perceptron as a sub-system.
2. They exploit the powerful non-linear mapping capabilities of the multi-layer perceptron, plus some form of memory.

Learning can be achieved by gradient descent procedures. The basic fully recurrent network (state space model) has the hidden unit activations feeding back into the network along with the inputs of the next time step as in **Figure 4.6**

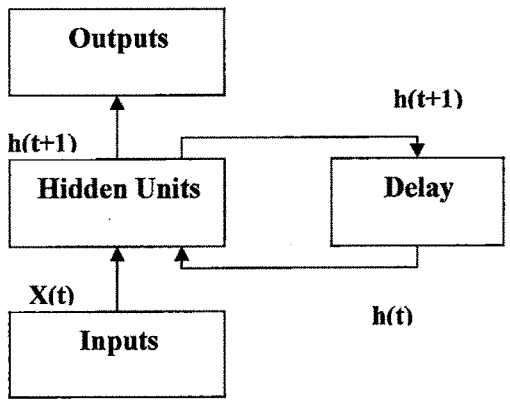


Figure 4.6 Recurrent Neural Network

4.3.2.2 Self Organizing Maps (SOM)

SOM system is also known as a Kohonen Network. This has a feed-forward structure with a single computational layer of neurons arranged in rows and columns. Each neuron is fully connected to all the source units in the input layer.
Each input neuron is connected to each and every neuron on the competitive layer which are organised as a two dimensional grid. **Figure 4.7** shows a Kohonen network with 2 inputs and 25 neurons on the competitive layer.

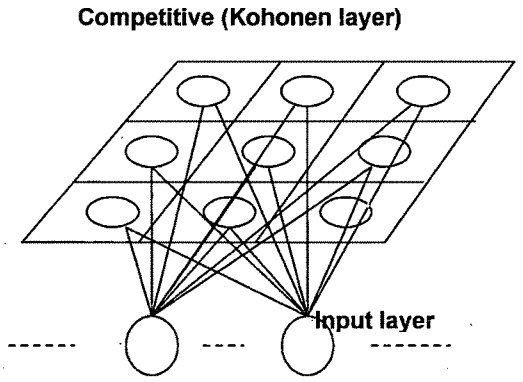


Figure 4.7 A Kohonen Self Organising Grid - 2 Dimensional Output

Once the SOM algorithm has converged, the feature map displays important statistical characteristics of the input space.

Given an input vector x, the feature map F provides a winning neuron I(x) in the output space and the weight vector $w_I(x)$ provides the coordinates of the image of that neuron in the input space **(Figure 4.8).**

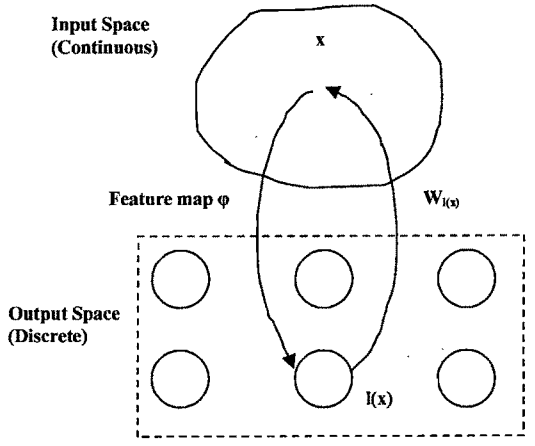
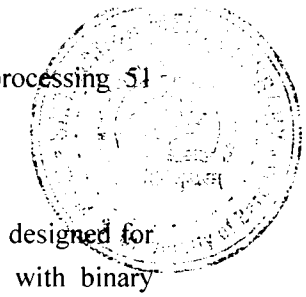


Figure 4.8 Input v/s output space



4.3.2.3 Adaptive resonance theory (ART) neural network

They are range of unsupervised incremental clustering neural networks designed for dynamically self-organizing data. The ART-1 network manages both with binary input vectors, and ART-2 with continuous input vectors. All ART systems implement a process of matching the inputs with cluster templates that leads to a resonant state and weights updating. If no sufficiently well matched template exists, a new template is created and set equal to the current input pattern. ARTMAP is a supervised version that learns input-output mappings by linking together two ART-1 modules by an intermediate layer or map field. Fuzzy versions of ART-1 give rise to the Fuzzy ARTMAP architecture. ART-3 allows a series of ART-2 modules to be linked into a processing hierarchy.

4.4 Neural Network Learning

A prescribed set of well-defined rules for the solution of a learning problem is called a **learning algorithm.** ANN undergoes changes in its free parameters as a result of applied stimulation. The learning algorithms can be classified as supervised and unsupervised depending upon whether the output is known on unknown.

4.4.1 Supervised learning

The training or learning is implemented using input-output pairs in the form of built-in knowledge. The neural network is exposed to a training vector drawn from the knowledge base. The network parameters are adjusted under the combined influence of the training vector and the error signal. The **error signal** is defined as the difference between the desired response and the actual response of the network. This adjustment is carried out iteratively in a step-by-step fashion until specified objective function is reached. When this condition is reached, the neural network is said to be trained and can generate output for any input condition. ISE or MSE can be used as a performance measure for the system over the training sample, defined as a function of the free parameters of the system. This function may be visualized as a multidimensional error-performance surface or error surface, with the free parameters as coordinates. The true error surface is averaged over all possible input-output examples. A supervised learning system is able to do this with the useful information it has about the gradient of the error surface corresponding to the current behavior of the system. The gradient of an error surface at any point is a vector that points in the direction of **steepest descent**.

4.4.1.1 Error Correction Learning

Consider the simple case of a neuron k constituting the only computational node in the output layer of a feed forward neural network, as depicted in **Figure 4.9(a)**.

Neuron k is driven by a signal vector **x**(n) produced by one or more layers of hidden neurons, which are themselves driven by an input vector (stimulus) applied to the source nodes (i.e., input layer) of the neural network. The argument n denotes discrete time. The output signal of neuron k is denoted by $y_k(n)$. This output signal, representing the only output of the neural network, is compared to a desired response or target output, denoted by $d_k(n)$. Consequently, an *error* signal, denoted by $e_k(n)$, is produced,

$$e_k(n) = d_k(n) - Y_k(n) \quad \text{.....(4.7)}$$

The error signal $e_k(n)$ actuates a control mechanism, the purpose of which is to apply a sequence of corrective adjustments to the synaptic weights of neuron k. The corrective adjustments are designed to make the output signal $y_k(n)$ come closer to the desired response $d_k(n)$ in a step-by-step manner. This objective is achieved by minimizing a **cost function or index of performance**, $\xi(n)$, defined in terms of the error signal $e_k(n)$ as:

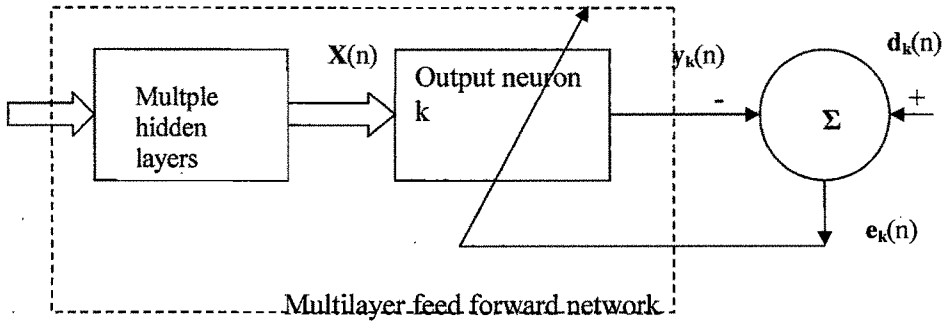
$$\xi(n) = \tfrac{1}{2} e^2_k(n) \quad \text{.....(4.8)}$$

In this **error-correction** learning, minimization of the cost function $\xi(n)$ leads to a learning rule called the **delta rule** or **Widrow-Hoff rule**. Let $W_{kj}(n)$ denote the value of synaptic weight w_{kj} *of* neuron k excited by element $x_j(n)$ of the signal vector x(n) at time step n. According to the delta rule, the adjustment $\Delta w_{kj}(n)$ applied to the synaptic weight w_{kj} at time step n is defined by

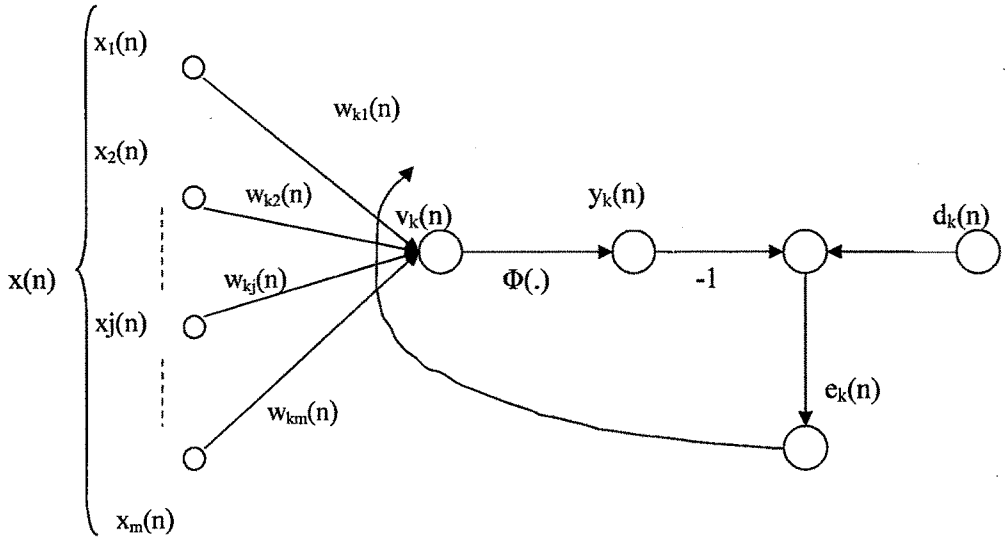
$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \quad \text{.....(4.9)}$$

where η is a positive constant that determines the **rate of learning** as one we proceeds from one step in the learning process to another. η is called **learning-rate parameter.**

Delta rule presumes that the error signal is directly measurable. For this measurement to be feasible we clearly need a supply of desired response from some external source, which is directly accessible to neuron k. In other words, neuron k is visible to the outside world, as depicted in **Figure 4.9(a)**. Synaptic adjustments made by the delta rule are localized around neuron *k*.



4.9 (a) Artificial Neural Network with highlighted k^{th} output node



4.9 (b) Signal flow graph

Figure 4.9 Error correction learning

After computed the synoptic adjustment $\Delta w_{kj}(n)$, the updated value of synaptic weight w_{kj} is determined by

$$Wkj(n + 1) = w_{kj}(n) + \Delta w_{kj}(n) \quad \text{.....(4.10)}$$

In effect, $w_{kj}(n)$ and $Wkj(n + 1)$ are the old and new values of synaptic weight, w_{kj} respectively.

Figure 4.9(b) shows a signal-flow graph representation of the error-correction learning process, focusing on the activity surrounding neuron k. From **Figure 4.9(b)** it is seen that error-correction learning is an example of a closed-loop feedback system.

From control theory we know that the stability of such a system is determined by those parameters that constitute the feedback loops of the system. Here, there is a single feedback loop, and one of those parameters of particular interest is the learning-rate parameter η. It is therefore important that η is carefully selected to ensure that the stability or convergence of the iterative learning process is achieved.

4.4.1.2 Memory based learning

Most of the past experiences are explicitly stored in a large memory of correctly classified input-output examples: $\{(x_i, d_i)\}^N_{i=1}$, where x_i denotes an input vector and d_i denotes the corresponding desired response. When classification of a test vector $\mathbf{x}_{test}$ (not seen before) is required, the algorithm responds by retrieving and analyzing the training data in a "local neighborhood" of $\mathbf{x}_{test}$.

All memory-based learning algorithms involve two essential ingredients:

- Criterion used for defining the local neighbourhood of the test vector $\mathbf{x}_{test}$.
- Learning rule applied to the training examples in the local neighborhood of $\mathbf{x}_{test}$.

The algorithms differ from each other in the way in which these two ingredients are defined.

4.4.2 Neurodynamic programming

The learning of an input-output mapping is performed through continued interaction with the environment to minimize a scalar index of performance. In **Figure 4.10** a reinforcement learning system is built around a critic that converts a primary reinforcement signal received from the environment into a higher quality reinforcement signal called the heuristic reinforcement signal, both of which are scalar inputs.

The system is designed to learn under **delayed reinforcement**, which means that the system observes a temporal sequence of stimuli (state vectors) also received from the environment, which eventually result in the generation of the heuristic reinforcement signal. The goal of learning is to minimize a **cost-to-go function**, defined as the expectation of the cumulative cost of actions taken over a sequence of steps instead of simply the immediate cost. Certain actions taken earlier in that sequence of time steps is in fact the best determinants of overall system behavior. The function of the learning machine, which constitutes the second component of the system, is to discover these actions and to feed them back to the environment.

Delayed-reinforcement learning is difficult to perform for two basic reasons:

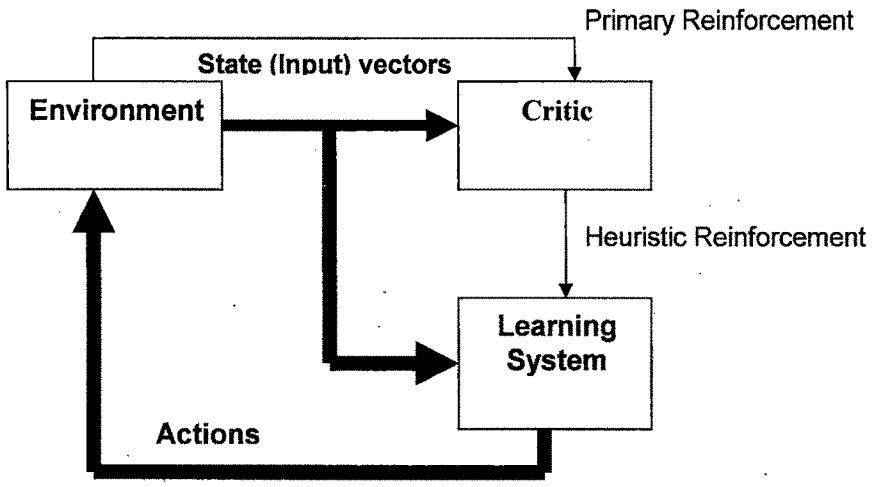


Figure 4.10 Neurodynamic programming

- There is no teacher to provide a desired response at each step of the learning process.
- The delay incurred in the generation of the primary reinforcement signal implies that the learning machine must solve a temporal credit assignment problem. By this we mean that the learning machine must be able to assign credit and blame individually to each action in the sequence of time steps that led to the final outcome, while the primary reinforcement may only evaluate the outcome.

4.4.3 Unsupervised learning

In unsupervised or self-organized learning provision is made for a task-independent measure of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically.

To perform unsupervised learning a competitive learning rule is used. The Neural Network consists of two layers: an input layer and a competitive layer. The input layer receives the available data. The competitive layer consists of neurons that compete with each other (in accordance with a learning rule) for the "opportunity" to respond to features contained in the input data. In its simplest form, the network operates in accordance with a "**winner-takes-all**" strategy. In such a strategy the

neuron with the greatest total input "wins" the competition and turns on; all the other neurons then switch off.

The stages of the SOM algorithm are:

1. **Initialization** –Choose random values for the initial weight vectors W.
2. **Sampling** – Draw a sample training input vector x from the input space.
3. **Matching** – Find the winning neuron I(x) that has weight vector closest to the input vector, i.e. the minimum value of

$$d_j(x)= \sum_{i=1}^{D}(x_i - w_{ji})^2 \quad \text{.....(4.11)}$$

4. **Updating** – Apply the weight update equation

$$\Delta w_{ji}= \eta(t)T_{j,I(x)}(t)(x_i-w_{ij}) \quad \text{.....(4.12)}$$

where $T_{j,I(x)}(t)$ is a Gaussian neighbourhood and $\eta(t)$ is the learning rate.

5. **Continuation** –Keep returning to step 2 until the feature map stops changing.

4.4.3.1 Kohonen algorithm

The most famous unsupervised learning algorithms known in neural networks is the Kohonen algorithm. The way this algorithm works is completely different from the way back-error propagation works. The network is presented with a set of training input patterns without giving any target output pattern for any of the inputs. One of the patterns is chosen randomly. Each neuron in the input layer of the network takes on the value of the corresponding entry in the input pattern. Then, a distance value (also known as the **Euclidean Distance**) is computed for each of the neurons on the competitive layer. The competitive neuron with the smallest Euclidean distance is known as the **winning node**. The way this first step is proceeded is as follows:

Let the chosen input pattern from the data set be:

$$I = (I_1, I_2, I_3, ..., I_n) \quad \text{.....(4.13)}$$

i.e. There are 'n' input neurons in the input layer.
Now suppose there are 'm' neurons in the competitive layer, where we call Ui the i'th neuron. So the whole of the competitive layer will be:

$U = (U_1, U_2, U_3, ..., U_m)$ .....(4.14)

For each competitive neuron U_i there is a set of n incoming connections from the n input neurons on the input layer. Each connection has a weight value W. So, for any neuron U_i on the competitive layer the set of its incoming connections weights is:

$W_i = (W_{i1}, W_{i2}, W_{i3}, ..., W_{in})$ .....(4.15)

The Euclidean distance value D_i of a neuron U_i in the competitive layer whenever an input pattern I is presented at the input layer is the following:

$$D_i = \sqrt{\sum_{j=1}^{n}(I_j - w_{ij})^2} \quad \text{.....(4.16)}$$

The competitive unit (U_c) with the lowest distance at this stage is the closest to the current input pattern therefore it is its representative.

After the winning node is identified, the next step is to identify the neighbourhood around it, which is simply the set of competitive units which are close to that winning node. **Figure 4.11** shows an example of a neighbourhood around a winning node, which is just the set of neurons that are within the square that is centred on the winning node U_c.

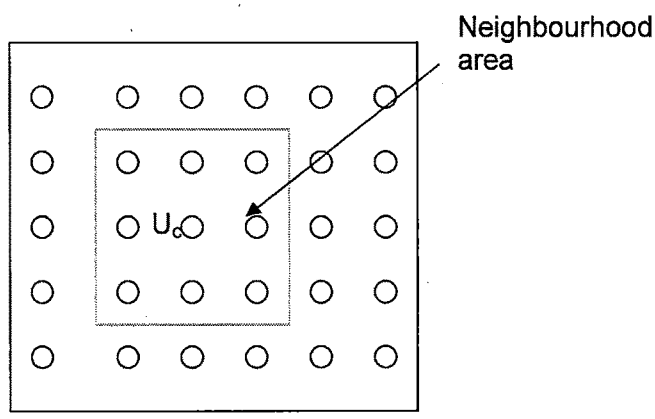


Figure 4.11 A 6x5 Kohonen Grid showing the size of neighbourhood influence around node Uc

The size of the neighbourhood, H, starts with a big enough size and decreases as the number of times the network has gone through all the data patterns increases such that:

$$H_t = H_0(1 - t/T) \quad \text{.....(4.17)}$$

Where H_t denotes the actual neighbourhood size, H_0 denotes the initial neighbourhood size which should be in the range of [0 1], t denotes the current learning epoch and T denotes the total number of epochs to be done. Where an epoch is when the network has swept through all the data patterns once.

During the learning stage, the weights of the incoming connections of each competitive neuron in the neighbourhood of the winning one are updated, including the winning node. Within the neighbourhood, the amount by which the weights of each node change depends on how far it is from the winning node. The closer the neurons are to the winning node, the bigger the amount by which their weights change. This value is computed using a function similar to the **sync** function which states

$\sin(d_{ci})/2d_{ci}$

where d_{ci} denotes the distance from the current node U_i to the winner U_c. The overall update equation for Kohonen Nets in this system is:

$$\Delta w_{ij} = \alpha(I_j - w_{ij}) \frac{\sin d_{ci}}{2d_{ci}} \quad \text{.....(4.18)}$$

if unit U_i is in the neighbourhood of winning unit U_c.

Where α is the learning rate. Typical choices are in the range [0.2 0.5]. The value of α is then decreased as training epochs increase such that:

$$\alpha_t = \alpha_o (1-t/T) \quad \text{.....(4.19)}$$

where α_o denotes the starting point learning rate value, a t denotes the current learning rate value, t and T are same as above.

If unit U_i is not in the neighbourhood of the winning node, no changes in its weights are made.

4.4.3.2 Competitive Learning

In competitive learning as the name implies, the output neurons of a neural network compete among themselves to become active (fired). Whereas in a neural network

based on Hebbian learning several output neurons may be active simultaneously, in competitive learning only a single output neuron is active at any one time. Hence competitive learning is highly suited to discover statistically salient features used to classify a set of input patterns.

There are three basic elements to a competitive learning rule (Rumelhart and Zipser, 1985):

- A set of neurons that are all the same except for some randomly distributed synaptic weights, and which therefore respond differently to a given set of input patterns.
- A limit imposed on the "strength" of each neuron.
- A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron pair group, is active (i.e., "ON") at a time. Neuron that wins the competition is called a **winner-takes-all neuron.**

Accordingly the individual neurons of the network learn to specialize on ensembles of similar patterns; in so doing they become feature detectors for different classes of input patterns.

4.4.3.3 Vector Quantization

One aim of using a Self Organizing Map (SOM) is to encode a large set of input vectors {x} by finding a smaller set of "representatives" or "prototypes" or "code-book vectors" $\{w_I(x)\}$ that provide a good approximation to the original input space. This is the basic idea of vector quantization theory, the motivation of which is dimensionality reduction or data compression.

In effect, the error of the vector quantization approximation is the total squared distance

$$D = \sum_x \| x - w_{I(x)} \|^2 \qquad \text{.....(4.20)}$$

between the input vectors {x} and their representatives $\{w_I(x)\}$, and goal is to minimize this. We shall see that performing a gradient descent style minimization of D does lead to the SOM weight update algorithm, which confirms that it is generating the best possible discrete low dimensional approximation to the input space (at least assuming it does not get trapped in a local minimum of the error function).

Probably the best way to think about vector quantization is in terms of general encoders and decoders. Suppose c(x) acts as an encoder of the input vector x, and x′(c) acts as a decoder of c(x), then one can attempt to get back to x with minimal loss of information.

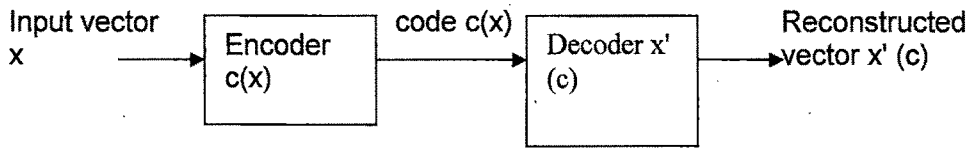


Figure 4.12 Encoding and Decoding in Vector Quantization

Generally, the input vector x will be selected at random according to some probability function p(x). Then the optimum encoding-decoding scheme is determined by varying the functions c(x) and x′(c) to minimize the expected distortion defined by

$$D = \sum_{X} \| X - X'(c(X)) \|^2 = \int dp_X(X) \| X - X'(c(X)) \|^2 \quad \text{.....(4.21)}$$

The necessary conditions for minimizing the expected distortion D in general situations are embodied in the two conditions of the generalized Lloyd algorithm:

Condition 1 Given the input vector x, choose the code c = c(x) to minimize the squared error distortion.
Condition 2 Given the code c, compute the reconstruction vector x′(c) as the centroid of those input vectors x that satisfy condition 1.

To implement vector quantization, the algorithm works in batch mode by alternately optimizing the encoder c(x) in accordance with condition 1, and then optimizing the decoder in accordance with condition 2, until D reaches a minimum. To overcome the problem of local minima, it may be necessary to run the algorithm several times with different initial code vectors.