CHAPTER 3

ARTIFICIAL NEURAL NETWORKS

D. T. PHAM, M. S. PACKIANATHER, A. A. AFIFY
Manufacturing Engineering Centre, Cardiff University, Cardiff CF24 3AA, United Kingdom

INTRODUCTION

Artificial neural networks are computational models of the brain. There are many types of neural networks representing the brain's structure and operation with varying degrees of sophistication. This chapter provides an introduction to the main types of networks and presents examples of each type.

1. TYPES OF NEURAL NETWORKS

Neural networks generally consist of a number of interconnected processing elements (PEs) or neurons. How the inter-neuron connections are arranged and the nature of the connections determine the structure of a network. How the strengths of the connections are adjusted or trained to achieve a desired overall behaviour of the network is governed by its learning algorithm. Neural networks can be classified according to their structures and learning algorithms.

1.1 Structural Categorisation

In terms of their structures, neural networks can be divided into two types: feedforward networks and recurrent networks.

Feedforward networks: In a feedforward network, the neurons are generally grouped into layers. Signals flow from the input layer through to the output layer via unidirectional connections, the neurons being connected from one layer to the next, but not within the same layer. Examples of feedforward networks include the multi-layer perceptron (MLP) [Rumelhart and McClelland, 1986], the radial basis function (RBF) network [Broomhead and Lowe, 1988; Moody and Darken, 1989], the learning vector quantization (LVQ) network [Kohonen, 1989], the cerebellar model articulation control (CMAC) network [Albus, 1975a], the group-method of data handling (GMDH) network [Hecht-Nielsen, 1990] and some spiking neural networks [Maass, 1997]. Feedforward networks can most naturally perform static mappings between an input space and an output space: the output at a given instant is a function only of the input at that instant.

Recurrent networks: In a recurrent network, the outputs of some neurons are fedback to the same neurons or to neurons in preceding layers. Thus, signals can flow in both forward and backward directions. Examples of recurrent networks include the Hopfield network [Hopfield, 1982], the Elman network [Elman, 1990] and the Jordan network [Jordan, 1986]. Recurrent networks have a dynamic memory: their outputs at a given instant reflect the current input as well as previous inputs and outputs.

1.2 Learning Algorithm Categorisation

Neural networks are trained by two main types of learning algorithms: supervised and unsupervised learning algorithms. In addition, there exists a third type, reinforcement learning, which can be regarded as a special form of supervised learning.

Supervised learning: A supervised learning algorithm adjusts the strengths or weights of the inter-neuron connections according to the difference between the desired and actual network outputs corresponding to a given input. Thus, supervised learning requires a *teacher* or *supervisor* to provide desired or target output signals. Examples of supervised learning algorithms include the delta rule [Widrow and Hoff, 1960], the generalised delta rule or backpropagation algorithm [Rumelhart and McClelland, 1986] and the LVQ algorithm [Kohonen, 1989].

Unsupervised learning: Unsupervised learning algorithms do not require the desired outputs to be known. During training, only input patterns are presented to the neural network which automatically adapts the weights of its connections to cluster the input patterns into groups with similar features. Examples of unsupervised learning algorithms include the Kohonen [Kohonen, 1989] and Carpenter-Grossberg Adaptive Resonance Theory (ART) [Carpenter and Grossberg, 1988] competitive learning algorithms.

Reinforcement learning: As mentioned before, reinforcement learning is a special case of supervised learning. Instead of using a teacher to give target outputs, a reinforcement learning algorithm employs a critic only to evaluate the goodness of the neural network output corresponding to a given input. An example of a reinforcement learning algorithm is the genetic algorithm (GA) [Holland, 1975; Goldberg, 1989].

2. NEURAL NETWORKS EXAMPLE

This section briefly describes the example neural networks and associated learning algorithms cited previously.

2.1 Multi-layer Perceptron (MLP)

MLPs are perhaps the best known type of feedforward networks. Figure 1a shows an MLP with three layers: an input layer, an output layer and an intermediate or hidden layer. Neurons in the input layer only act as buffers for distributing the input signals x_i to neurons in the hidden layer. Each neuron j (Figure 1b) in the hidden layer sums up its input signals x_i after weighting them with the strengths of the respective connections w_{ji} from the input layer and computes its output y_j as a function f of the sum, viz.

(1) $$y_j = f\left(\sum w_{ji} x_i\right)$$

f can be a simple threshold function or a sigmoidal, hyperbolic tangent or radial basis function (see Table 1).

The output of neurons in the output layer is computed similarly.

The backpropagation (BP) algorithm, a gradient descent algorithm, is the most commonly adopted MLP training algorithm. It gives the change Δw_{ji} in the weight

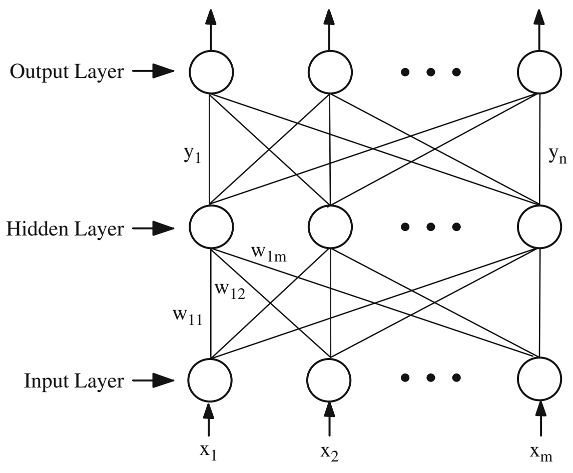


Figure 1a. A multi-layer perceptron

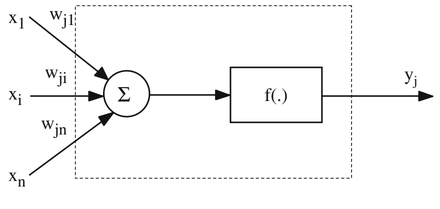


Figure 1b. Details of a neuron

Table 1. Activation functions

Type of Functions	Functions
Linear	$f(s) = s$
Threshold	$f(s) = \begin{cases} +1, & if\ s > s_t \\ -1, & otherwise \end{cases}$
Sigmoid	$f(s) = 1/(1 + exp(-s))$
Hyperbolic tangent	$f(s) = (1 - exp(-2s))/(1 + exp(2s))$
Radial basis function	$f(s) = exp(-s^2/\beta^2)$

of a connection between neurons i and j as follows:

$$\Delta w_{ji} = \eta \delta_j x_i \tag{2}$$

where η is a parameter called the learning rate and δ_j is a factor depending on whether neuron j is an output neuron or a hidden neuron. For output neurons,

$$\delta_j = \left(\frac{\partial f}{\partial net_j} \right) \left(y_j^{(t)} - y_j \right) \tag{3}$$

and for hidden neurons,

$$\delta_j = \left(\frac{\partial f}{\partial net_j} \right) \sum_q w_{qj} \delta_q \tag{4}$$

In Equation (3), net_j is the total weighted sum of input signals to neuron j and $y_j^{(t)}$ is the target output for neuron j.

As there are no target outputs for hidden neurons, in Equation (4), the difference between the target and actual output of a hidden neuron j is replaced by the weighted sum of the δ_q terms already obtained for neurons q connected to the output of j. Thus, iteratively, beginning with the output layer, the δ term is computed for neurons in all layers and weight updates determined for all connections. The weight updating process can take place after the presentation of each training pattern (pattern-based training) or after the presentation of the whole set of training patterns (batch training). In either case, a training epoch is said to have been completed when all training patterns have been presented once to the MLP.

For all but the most trivial problems, several epochs are required for the MLP to be properly trained. A commonly adopted method to speed up the training is to add a "momentum" term to Equation (2) which effectively lets the previous weight change influence the new weight change, viz:

$$\Delta w_{ji}(k+1) = \eta \delta_j x_i + \mu \Delta w_{ji}(k) \tag{5}$$

where $\Delta w_{ji}(k+1)$ and $\Delta w_{ji}(k)$ are weight changes in epochs $(k+1)$ and (k) respectively and μ is the "momentum" coefficient.

Another learning method suitable for training MLPs is the genetic algorithm (GA). This is an optimisation algorithm based on evolution principles. The weights of the connections are considered genes in a chromosome. The goodness or fitness of the chromosome is directly related to how well trained the MLP is. The algorithm starts with a randomly generated population of chromosomes and applies genetic operators to create new and fitter populations. The most common genetic operators are the selection, crossover and mutation operators. The selection operator chooses chromosomes from the current population for reproduction. Usually, a biased selection procedure is adopted which favours the fitter chromosomes. The crossover operator creates two new chromosomes from two existing chromosomes by cutting them at a random position and exchanging the parts following the cut. The mutation operator produces a new chromosome by randomly changing the genes of an existing chromosome. Together, these operators simulate a guided random search method which can eventually yield the optimum set of weights to minimise the differences between the actual and target outputs of the neural network. Further details of genetic algorithms can be found in the chapter on Soft Computing and its Applications in Engineering and Manufacture.

2.2 Radial Basis Function (RBF) Network

Large multi-layer perceptron (MLP) networks take a long time to train. This has led to the construction of alternative networks such as the Radial Basis Function (RBF) network [Cichocki and Unbahauen, 1993; Hassoun, 1995; Haykin, 1999]. The RBF network is the most used network after MLPs. Figure 2 shows the structure of a RBF network which consists of three layers. The input layer neurons receive the inputs $x_1, \ldots, x_M$. The hidden layer neurons provide a set of activation functions that constitute an arbitrary "basis" for the input patterns in the input space to be expanded into the hidden space by way of non-linear transformation. At the input of each hidden neuron, the distance between the centre of each activation or basis function and the input vector is calculated. Applying the basis function to this distance produces the output of the hidden neuron. The RBF network output y is formed by the neuron in the output layer as a weighted sum of the hidden layer neuron activation.

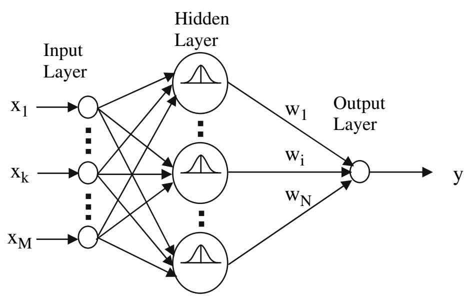


Figure 2. The RBF network

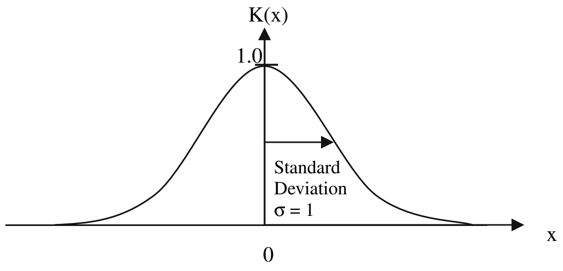


Figure 3. The Radial Basis Function

The basis function is generally chosen to be a standard function which is positive at its centre $x = 0$ and then decreases uniformly to zero on either side as shown in Figure 3. A common choice is the Gaussian distribution function:

$$K(x) = \exp\left(-\frac{x^2}{2}\right) \tag{6}$$

This function can be shifted to an arbitrary centre, $x = c$, and stretched by varying its standard deviation σ as follows:

$$K\left(\frac{(x-c)}{\sigma}\right) = \exp\left(-\frac{(x-c)^2}{2\sigma^2}\right) \tag{7}$$

The output of the RBF network y is given by:

$$y = \sum_{i=1}^{N} w_i K\left(\frac{\|x-c_i\|}{\sigma_i}\right)_{\forall x} \tag{8}$$

where w_i is the weight of the hidden neuron i, c_i the centre of basis function i and σ_i the standard deviation of the function. $\|x-c_i\|$ is the norm of $(x-c_i)$. There are various ways to calculate the norm. The most common is the Euclidean norm given by:

$$\|x-c_i\| = \sqrt{(x_1-c_{i1})^2 + (x_2-c_{i2})^2 + \ldots. + (x_M-c_{iM})^2} \tag{9}$$

This norm gives the distance between the two points x and c_i in N-dimensional space. All points x that are the same radial distance from c_i give the same value for the norm and hence the same value for the basis function. Hence the basis functions are called Radial Basis Functions. Obtaining the values for w_i, c_i and σ_i requires training the RBF network. Because the basis functions are differentiable, back-propagation could be used as with MLP networks. Training of a multiple-input single-output RBF network can proceed as follows:

(i) choose the number N of hidden units;
There is no firm guidance available for this. The selection of N is normally made by trial and error. In general, the smallest N that gives the RBF network an acceptable performance is adopted.

(ii) choose the centres, c_i;
Centre selection could be performed in three different ways [Haykin, 1999]:
  a) Trial and error:
  Centres can be selected by trial and error. This is not always easy if little is known about underlying functional behaviour of data. Usually, the centres are spread evenly or randomly over N-dimensional input space.
  b) Self-organized selection:
  An adaptive unsupervised method can be used to learn where to place the centres.
  c) Supervised selection:
  A supervised learning process, commonly error correction learning, can be deployed to fix the centres.

(iii) choose stretch constants, σ_i;
Several heuristics are available. A popular way is to set σ_i equal to the distance to nearest neighbour. First the distances between centres are computed then the nearest distance is chosen to be the value of σ_i.

(iv) calculate weights, w_i.
When c_i and w_i are known, the outputs of hidden units $(O_1, \ldots\ldots, O_N)^T$ can be calculated for any pattern of inputs $x = (x_1, \ldots, x_M)$. Assuming there are P input patterns x in the training set, there will be P sets of hidden unit outputs that can be calculated. These can be assembled in a $N \times P$ matrix:

$$(10) \qquad O = \begin{bmatrix} O_1^{(1)} & O_1^{(2)} & \ldots\ldots\ldots\ldots & O_1^{(P)} \\ O_2^{(1)} & O_2^{(2)} & \ldots\ldots\ldots\ldots & O_2^{(P)} \\ . & & & \\ . & & & \\ O_N^{(1)} & O_N^{(2)} & \ldots\ldots\ldots\ldots & O_N^{(P)} \end{bmatrix}$$

If the output $y^{(i)}$ of the RBF network corresponding to training input pattern $x^{(i)}$ is $y^{(i)} = O_1^{(i)} w_1 + O_2^{(i)} w_2 + \ldots + O_N^{(i)} w_N$, the following equation can be obtained:

$$(11) \qquad y = \begin{bmatrix} y^{(1)} \\ . \\ . \\ y^{(P)} \end{bmatrix} = \begin{bmatrix} O_1^{(1)} & O_N^{(1)} \\ . & \\ . & \\ O_1^{(P)} & O_N^{(P)} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ \\ \\ w_N \end{bmatrix} = O^T \cdot w$$

y is the vector of actual outputs corresponding to the training inputs x. Ideally, y should be equal to d, the desired/target outputs. Unknown coefficients w_i can be chosen to minimise the sum-squared-error of y compared with d. It can be shown that this is achieved when:

$$(12) \qquad w = (O.O^{\mathrm{T}})^{-1}.O.d$$

2.3 Learning Vector Quantization (LVQ) Network

Figure 4 shows an LVQ network which comprises three layers of neurons: an input buffer layer, a hidden layer and an output layer. The network is fully connected between the input and hidden layers and partially connected between the hidden and output layers, with each output neuron linked to a different cluster of hidden neurons. The weights of the connections between the hidden and output neurons are fixed to 1. The weights of the input-hidden neuron connections form the components of *reference* vectors (one reference vector is assigned to each hidden neuron). They are modified during the training of the network. Both the hidden neurons (also known as Kohonen neurons) and the output neurons have binary outputs. When an input pattern is supplied to the network, the hidden neuron whose reference vector is closest to the input pattern is said to win the competition for being activated and thus allowed to produce a "1". All other hidden neurons are forced to produce a "0". The output neuron connected to the cluster of hidden neurons that contains the winning neuron also emits a "1" and all other output neurons a "0". The output neuron that produces a "1" gives the class of the input pattern, each output neuron being dedicated to a different class. The simplest LVQ training procedure is as follows:

(i) initialise the weights of the reference vectors;
(ii) present a training input pattern to the network;
(iii) calculate the (Euclidean) distance between the input pattern and each reference vector;
(iv) update the weights of the reference vector that is closest to the input pattern, that is, the reference vector of the winning hidden neuron. If the latter belongs

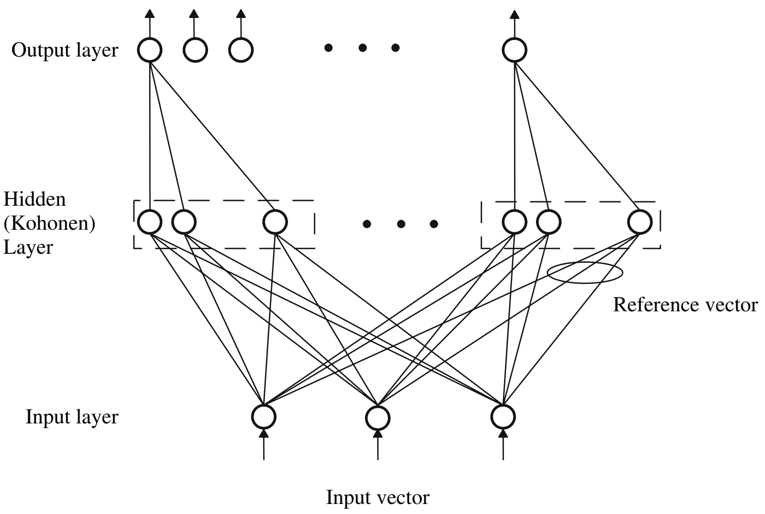


Figure 4. Learning Vector Quantization network

to the cluster connected to the output neuron in the class that the input pattern is known to belong to, the reference vector is brought closer to the input pattern. Otherwise, the reference vector is moved away from the input pattern;

(v) return to (ii) with a new training input pattern and repeat the procedure until all training patterns are correctly classified (or a stopping criterion is met).

For other LVQ training procedures, see for example [Pham and Oztemel, 1994].

2.4 CMAC Network

CMAC (Cerebellar Model Articulation Control) [Albus, 1975a, 1975b, 1979a, 1979b; An et al 1994] can be considered a supervised feedforward neural network with the characteristics of a fuzzy associative memory. A basic CMAC module is shown in Figure 5.

CMAC consists of a series of mappings:

$$S \xrightarrow{e} M \xrightarrow{f} A \xrightarrow{g} u \tag{13}$$

where

$S = \{\text{input vectors}\}$
$M = \{\text{intermediate variables}\}$
$A = \{\text{association cell vectors}\}$
$u = \text{output of CMAC} \equiv h(S)$
$h \equiv g \cdot f \cdot e$

(a) Input encoding ($S \to M$ mapping)

The $S \to M$ mapping is a set of submappings, one for each input variable:

$$S \to M = \begin{bmatrix} s_1 \to m_1 \\ s_2 \to m_2 \\ \vdots \\ s_n \to m_n \end{bmatrix} \tag{14}$$

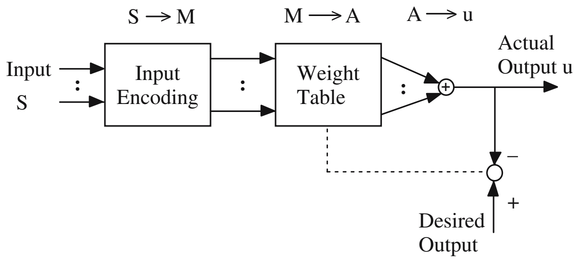


Figure 5. A basic CMAC module

The range of s_1 is coarsely discretised using the quantising functions $q_1, q_2, \dots, q_k$. Each function divides the range into k intervals. The intervals produced by function q_{j+1} are offset by one kth of the range compared to their counterparts produced by function q_j. m_i is a set of k intervals generated by q_1 to q_k respectively.

An example is given in Figure 6 to illustrate the internal mappings within a CMAC module. The $S \rightarrow M$ mapping is shown in the leftmost part of the figure.

In Figure 6, two input variables s_1 and s_2 are represented with unity resolution in the range of 0 to 8. The range of each input variable is described using three quantising functions. For example, the range of s_1 is described by functions q_1, q_2, and q_3. q_1 divides the range into intervals A, B, C and D. q_2 gives intervals E, F, G, and H and q_3 provides intervals I, J, K and L. That is,

$$q_1 = \{A, B, C, D\}$$

$$q_2 = \{E, F, G, H\}$$

$$q_3 = \{I, J, K, L\}$$

For every value of s_1, there exists a set of elements, m_1, which are the intersection of the functions q_1 to q_3, such that the value of s_1 uniquely defines set m_1 and vice versa. For example, value $s_1 = 5$ maps to set $m_1 = \{B, G, K\}$ and vice versa. Similarly, value $s_2 = 4$ maps to set $m_2 = \{b, g, j\}$ and vice versa.

The $S \rightarrow M$ mapping gives CMAC two advantages: the first is that a single precise variable s_i can be transmitted over several imprecise information channels. Each channel carries only a small part of the information of s_i. This increases the reliability of the information transmission. The other advantage is that small changes in the value of s_i have no influence on most of the elements in m_i. This leads to the property of input generalisation which is important in an environment where random noise exists.

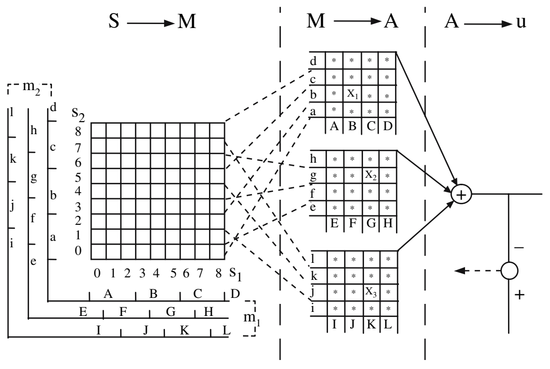


Figure 6. Internal mappings within a CMAC module

(b) Address computing ($M \rightarrow A$ mapping)

A is a set of address vectors associated with weight tables. A is obtained by combining the elements of m_i. For example, in Figure 6, the sets $m_1 = \{B, G, K\}$ and $m_2 = \{b, g, j\}$ are combined to give the set of elements $A = \{a_1, a_2, a_3\} = \{Bb, Gg, Kj\}$.

(c) Output computing ($A \rightarrow U$ mapping)

This mapping involves looking up the weight tables and adding the contents of the addressed locations to yield the output of the network. The following formula is employed:

$$u = \sum_i w_i(a_i) \tag{15}$$

That is, only the weights associated with the addresses a_i in A are summed. For this given example, these weights are:

$$w(Bb) = x_1$$
$$w(Gg) = x_2$$
$$w(Kj) = x_3$$

Thus the output is:

$$u = x_1 + x_2 + x_3 \tag{16}$$

Training a CMAC module consists of adjusting the stored weights. Assuming that f is the function that CMAC has to learn, the following training steps could be adopted:

(i) select a point S in the input space and obtain the current output u corresponding to S;
(ii) let $\overline{u}$ be the desired output of CMAC, that is, $\overline{u} = f(S)$;
(iii) if $|\overline{u} - u| \leq \xi$, where ξ is an acceptable error, then do nothing; the desired value is already stored in CMAC. However, if $|\overline{u} - u| > \xi$, then add to every weight which contributed to u the quantity

$$\Delta = \alpha \frac{\overline{u} - u}{|A|} \tag{17}$$

where $|A|$ = the number of weights which contributed to u and α is the learning rate.

2.5 Group Method of Data Handling (GMDH) Network

Figure 7 shows a GMDH network and the details of one of its neurons. Unlike the feedforward neural networks previously described which have a fixed structure,

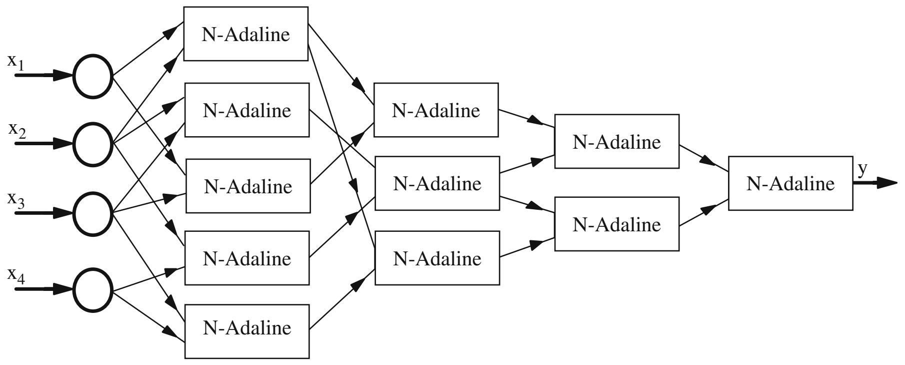


Figure 7a. A trained GMDH network

Note: Each GMDH neuron is an N-Adaline, which is an Adaptive Linear Element with a nonlinear preprocessor

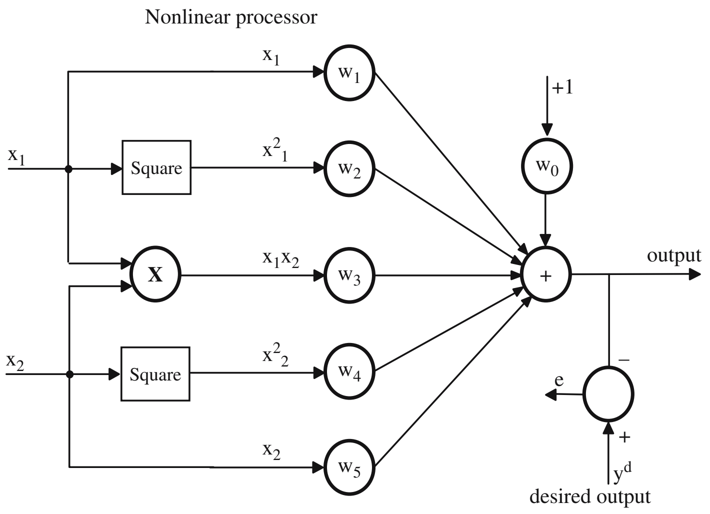


Figure 7b. Details of a GMDH Neuron

a GMDH network has a structure which grows during training. Each neuron in a GMDH network usually has two inputs x_1 and x_2 and produces an output y that is a quadratic combination of these inputs, viz.

$$y = w_o + w_1x_1 + w_2x_1^2 + w_3x_1x_2 + w_4x_2^2 + w_5x_2 \tag{18}$$

Training a GMDH network consists of configuring the network starting with the input layer, adjusting the weights of each neuron, and increasing the number of layers until the accuracy of the mapping achieved with the network deteriorates.

The number of neurons in the first layer depends on the number of external inputs available. For each pair of external inputs, one neuron is used.

Training proceeds with presenting an input pattern to the input layer and adapting the weights of each neuron according to a suitable learning algorithm, such as the delta rule (see for example [Pham and Liu, 1994]), viz.

$$W_{k+1} = W_k + \alpha \frac{X_k}{|X_k|^2}\left(y_k^d - W_k^T X_k\right) \tag{19}$$

where W_k, the weight vector of a neuron at time k, and X_k the modified input vector to the neuron at time k, are defined as

$$W_k = [w_0, w_1, w_2, w_3, w_4, w_5]^T \tag{20}$$

$$X_k = \left[1, x_1, x_1^2, x_1x_2, x_2^2, x_2\right]^T \tag{21}$$

and y_k^d is the desired network output at time k.

Note that, for this description, it is assumed that the GMDH network only has one output. Equation (19) shows that the desired network output is presented to each neuron in the input layer and an attempt is made to train each neuron to produce that output. When the sum of the mean square errors S_E over all the desired outputs in the training data set for a given neuron reaches the minimum for that neuron, the weights of the neuron are frozen and its training halted. When the training has ended for all neurons in a layer, the training for the layer stops. Neurons that produce S_E values below a given threshold when another set of data (known as the selection data set) is presented to the network are selected to grow the next layer. At each stage, the smallest S_E value achieved for the selection data set is recorded. If the smallest S_E value for the current layer is less than that for the previous layer (that is, the accuracy of the network is improving), a new layer is generated, the size of which depends on the number of neurons just selected. The training and selection processes are repeated until the S_E value deteriorates. The best neuron in the immediately preceding layer is then taken as the output neuron for the network.

2.6 Hopfield Network

Figure 8 shows one version of a Hopfield network. This network normally accepts binary and bipolar inputs (+1 or −1). It has a single "layer" of neurons, each connected to all the others, giving it a recurrent structure, as mentioned earlier. The *training* of a Hopfield network takes only one step, the weights w_{ij} of the network being assigned directly as follows:

$$w_{ij} = \begin{cases} \frac{1}{N}\sum_{c=1}^{P} x_i^c x_j^c, & i \neq j \\ 0, & i = j \end{cases} \tag{22}$$

where w_{ij} is the connection weight from neuron i to neuron j, and x_i^c (which is either +1 or −1) is the ith component of the training input pattern for class c, P

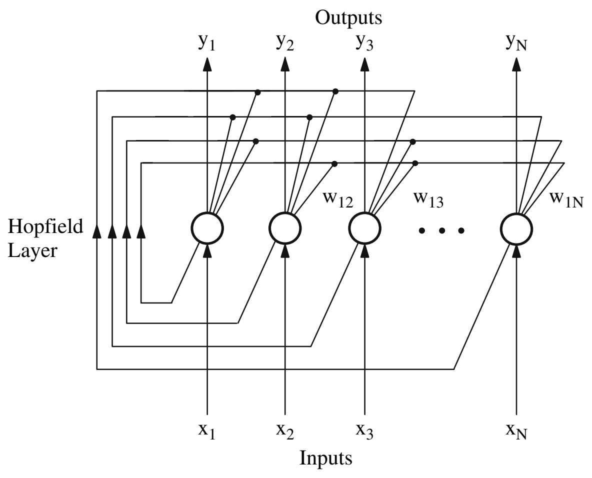


Figure 8. A Hopfield network

the number of classes and N the number of neurons (or the number of components in the input pattern). Note from Equation (22) that $w_{ij} = w_{ji}$ and $w_{ii} = 0$, a set of conditions that guarantee the stability of the network. When an unknown pattern is input to the network, its outputs are initially set equal to the components of the unknown pattern, viz.

$$(23) \qquad y_i(0) = x_i, \quad 1 \le i \le N$$

Starting with these initial values, the network iterates according to the following equation until it reaches a minimum *energy* state, i.e. its outputs stabilise to constant values:

$$(24) \qquad y_i\,(k+1) = f\left[\sum_{j=1}^{N} w_{ij} y_i\,(k)\right], 1 < i \le N$$

where f is a hard limiting function defined as

$$(25) \qquad f(x) = \begin{cases} -1, & x < 0 \\ 1, & x > 0 \end{cases}$$

2.7 Elman and Jordan Nets

Figures 9a and b show an Elman net and a Jordan net, respectively. These networks have a multi-layered structure similar to the structure of MLPs. In both nets, in addition to an ordinary hidden layer, there is another special hidden layer sometimes called the context or state layer. This layer receives feedback signals from the

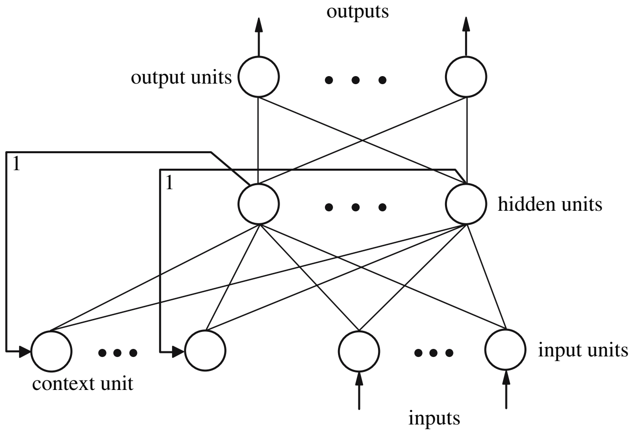


Figure 9a. An Elman network

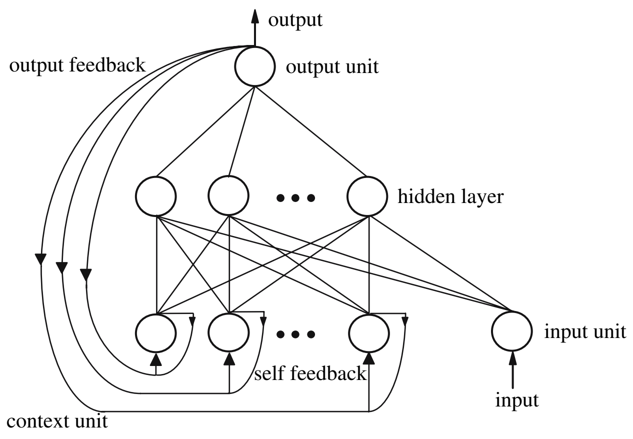


Figure 9b. A Jordan network

ordinary hidden layer (in the case of an Elman net) or from the output layer (in the case of a Jordan net). The Jordan net also has connections from each neuron in the context layer back to itself. With both nets, the outputs of neurons in the context layer, are fed forward to the hidden layer. If only the forward connections are to be adapted and the feedback connections are preset to constant values, these networks can be considered ordinary feedforward networks and the BP algorithm used to train them. Otherwise, a GA could be employed [Pham and Karaboga, 1993b; Karaboga, 1994]. For improved versions of the Elman and Jordan nets, see [Pham and Liu, 1992; Pham and Oh, 1992].

2.8 Kohonen Network

A Kohonen network or a self-organising feature map has two layers, an input buffer layer to receive the input pattern and an output layer (see Figure 10). Neurons in the output layer are usually arranged into a regular two-dimensional array. Each output neuron is connected to all input neurons. The weights of the connections form the components of the reference vector associated with the given output neuron.

Training a Kohonen network involves the following steps:

(i) initialise the reference vectors of all output neurons to small random values;

(ii) present a training input pattern;

(iii) determine the winning output neuron, i.e. the neuron whose reference vector is closest to the input pattern. The Euclidean distance between a reference vector and the input vector is usually adopted as the distance measure;

(iv) update the reference vector of the winning neuron and those of its neighbours. These reference vectors are brought closer to the input vector. The adjustment is greatest for the reference vector of the winning neuron and decreased for reference vectors of neurons further away. The size of the neighbourhood of a neuron is reduced as training proceeds until, towards the end of training, only the reference vector of a winning neuron is adjusted.

In a well-trained Kohonen network, output neurons that are close to one another have similar reference vectors. After training, a labelling procedure is adopted where input patterns of known classes are fed to the network and class labels are assigned to output neurons that are activated by those input patterns. As with the LVQ network, an output neuron is activated by an input pattern if it wins the competition against other output neurons, that is, if its reference vector is closest to the input pattern.

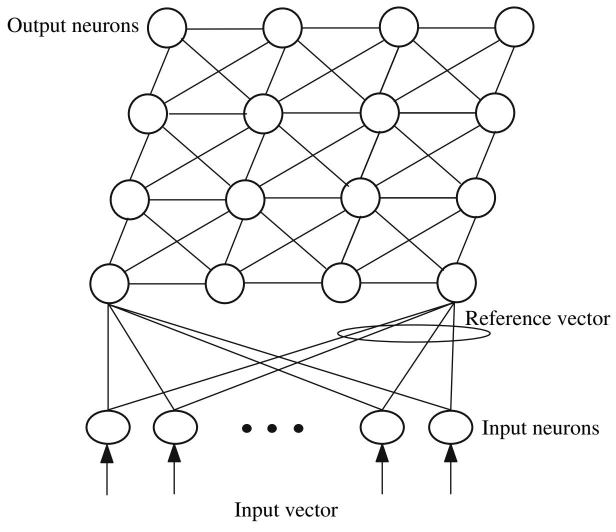


Figure 10. A Kohonen network

2.9 ART Networks

There are different versions of the ART network. Figure 11 shows the ART-1 version for dealing with binary inputs. Later versions, such as ART-2 can also handle continuous-valued inputs.

ART-1

As illustrated in Figure 11, an ART-1 network has two layers, an input layer and an output layer. The two layers are fully interconnected, the connections are in both the forward (or bottom-up) direction and the feedback (or top-down) direction. The vector W_i of weights of the bottom-up connections to an output neuron i forms an exemplar of the class it represents. All the W_i vectors constitute the long-term memory of the network. They are employed to select the winning neuron, the latter again being the neuron whose W_i vector is most similar to the current input pattern. The vector V_i of the weights of the top-down connections from an output neuron i is used for *vigilance* testing, that is, determining whether an input pattern is sufficiently close to a stored exemplar. The vigilance vectors V_i form the short-term memory of the network. V_i and W_i are related in that W_i is a normalised copy of V_i, viz.

$$(26) \qquad W_i = \frac{V_i}{\varepsilon + \sum V_{ji}}$$

where ε is a small constant and V_{ji}, the jth component of V_i (i.e. the weight of the connection from output neuron i to input neuron j).

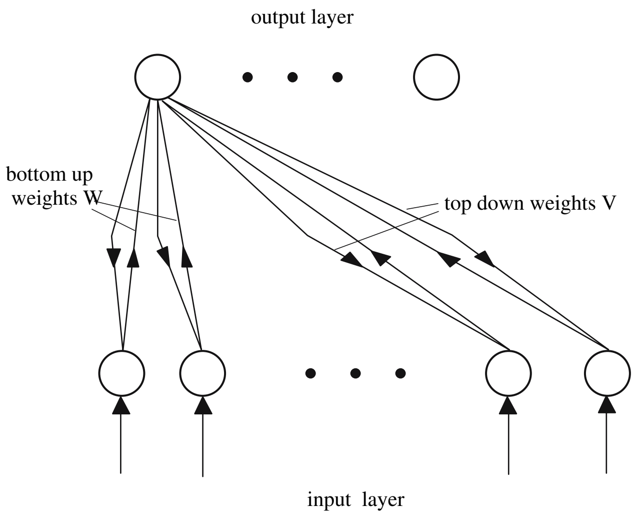


Figure 11. An ART-1 network

Training an ART-1 network occurs continuously when the network is in use and involves the following steps:

(i) initialise the exemplar and vigilance vectors W_i and V_i for all output neurons, setting all the components of each V_i to 1 and computing W_i according to Equation (26). An output neuron with all its vigilance weights set to 1 is known as an *uncommitted* neuron in the sense that it is not assigned to represent any pattern classes;

(ii) present a new input pattern x;

(iii) enable all output neurons so that they can participate in the competition for activation;

(iv) find the winning output neuron among the competing neurons, i.e. the neuron for which $x.\ W_i$ is largest; a winning neuron can be an uncommitted neuron as is the case at the beginning of training or if there are no better output neurons;

(v) test whether the input pattern x is sufficiently similar to the vigilance vector V_i of the winning neuron. Similarity is measured by the fraction r of bits in x that are also in W_i, viz.

$$r = \frac{x.V_i}{\sum x_i} \qquad (27)$$

x is deemed to be sufficiently similar to V_i if r is at least equal to *vigilance threshold* $\rho(0 < \rho \leq 1)$;

(vi) go to step (vii) if $r \geq \rho$ (i.e. there is *resonance*); else disable the winning neuron temporarily from further competition and go to step (iv) repeating this procedure until there are no further enabled neurons;

(vii) adjust the vigilance vector V_i of the most recent winning neuron by logically ANDing it with x, thus deleting bits in V_i that are not also in x; compute the bottom-up exemplar vector W_i using the new V_i according to Equation (26); activate the winning output neuron;

(viii) go to step (ii).

The above training procedure ensures that if the same sequence of training patterns is repeatedly presented to the network, its long-term and short-term memories are unchanged (i.e. the network is *stable*). Also, provided there are sufficient output neurons to represent all the different classes, new patterns can always be learnt, as a new pattern can be assigned to an uncommitted output neuron if it does not match previously stored exemplars well (i.e. the network is *plastic*).

ART-2

The architecture of an ART-2 network [Carpenter and Grossberg, 1987; Pham and Chan, 1998; 2001] is depicted in Figure 12. In this particular configuration, the "feature representation" field ($F1$) consists of 4 loops. An input pattern will be circulated in the lower two loops first. Inherent noise in the input pattern will be suppressed (this is controlled by the parameters a and b and the feedback function $f(\cdot)$) and prominent features in it will be accentuated. Then the enhanced input pattern will be passed to the upper two $F1$ loops and will excite the neurons in the "category representation" field ($F2$) via the bottom-up weights. The "established class" neuron in $F2$ that receives the strongest stimulation will fire. This neuron will read out a "top-down expectation" in the form of a set of top-down weights sometimes referred to as class templates. This top-down expectation will be compared against the enhanced input pattern by the vigilance mechanism. If the vigilance test is passed, the top-down and bottom-up weights will be updated and, along with the enhanced input pattern, will circulate repeatedly in the two upper $F1$ loops until stability is achieved. The time taken by the network to reach a stable state depends on how close the input pattern is to passing the vigilance test. If it passes the test comfortably, i.e. the input pattern is quite similar to the top-down expectation, stability will be quick to achieve. Otherwise, more iterations are required. After the top-down and bottom-up weights have been updated, the current firing neuron will become an established class neuron. If the vigilance test fails, the current firing neuron will be disabled. Another search within the remaining established class neurons in the $F2$ layer will be conducted. If none of the established class neurons has a top-down expectation similar to the input pattern, an unoccupied $F2$ neuron will be assigned to classify the input pattern. This procedure repeats itself until either all the patterns are classified or the memory capacity of $F2$ has been exhausted.

The basic ART-2 training algorithm can be summarised as follows:

(i) initialising the top-down and bottom-up long term memory traces;
(ii) presenting an input pattern from the training data set to the network;
(iii) triggering the neuron with the highest total input in the category representation field;
(iv) checking the match between the input pattern and the exemplar in the top-down filter (long term memory) using a vigilance parameter;
(v) starting the learning process if the mismatch is within the tolerance level defined by the vigilance parameter and then going to step (viii); otherwise, moving to the next step;
(vi) disabling the current active neuron in the category representation field and returning to step (iii); go to step (vii) if all the established classes have been tried;
(vii) establishing a new class for the given input pattern;
(viii) repeating (ii) to (vii) until the network stabilises or a specified number of iterations are completed.

In the recall mode, only steps (ii), (iii), (iv) and (viii) will be utilised.

Dynamics of ART-2: The dynamics of the ART-2 network illustrated in Figure 12 is controlled by a set of mathematical equations. They are as follows:

$$w'_i = I_i + au'_i \tag{28}$$

$$x'_i = \frac{w'_i}{\|W'\|} \tag{29}$$

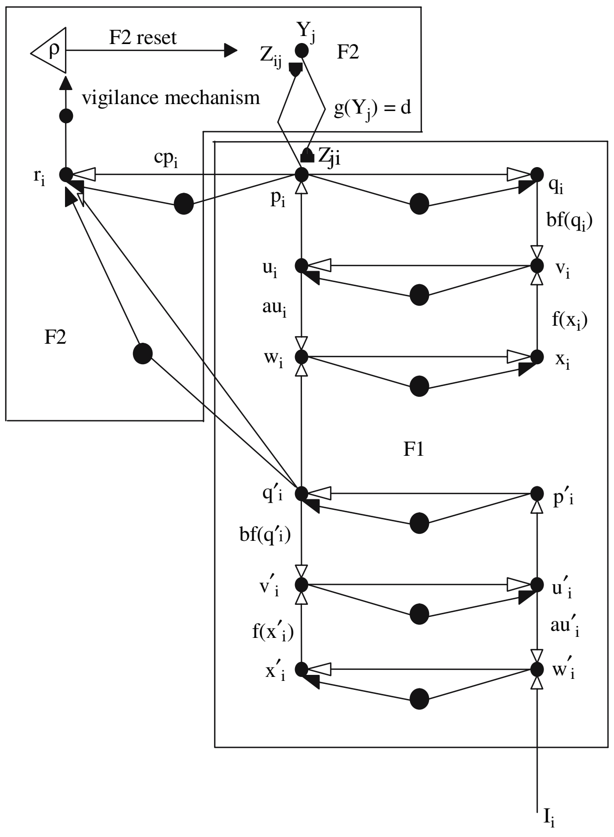


Figure 12. Architecture of an ART-2 network

(30) $$v'_i = f(x'_i) + bf(q'_i)$$

(31) $$u'_i = \frac{v'_i}{\|V'\|}$$

(32) $$p'_i = u'_i$$

(33) $$q'_i = \frac{p'_i}{\|P'\|}$$

(34) $$w_i = q'_i$$

(35) $$x_i = \frac{w_i}{\|W\|}$$

(36) $$v_i = f(x_i) + bf(q_i)$$

(37) $$u_i = \frac{v_i}{\|V\|}$$

(38) $$p_i = u_i + \sum_j g\left(Y_j\right) z_{ji}$$

(39) $$q_i = \frac{p_i}{\|P\|}$$

The symbol $\|X\|$ represents the L_2 *norm* of the vector X. If $X = [x_1, x_2 \ldots .x_n]$, then $\|X\| = \sqrt{x_1^2 + x_2^2 + \ldots .. + x_n^2}$. The output of the jth neuron in the classification layer is denoted by $g(Y_j)$. The L_2 norm is used in the equations for the purpose of normalising the input data. The function $f(\cdot)$ used in Equations (30) and (36) is a non-linear function, the purpose of which is for suppressing the noise in the input pattern down to a prescribed level. The definition of $f(\cdot)$ is

(40) $$f(x) = \begin{cases} 0 & if\ 0 \leq x < \theta \\ x & if\ x \geq \theta \end{cases}$$

where θ is a user defined parameter, it has a value between 0 and 1.

Learning Mechanism of ART-2: When an input pattern is applied to the ART-2 network, it will pass through the 4 loops comprising $F1$ and then stimulate the classification neurons in $F2$. The total excitation received by the jth neuron in the classification layer is equal to T_j where

(41) $$T_j = \sum_i p_i z_{ij}$$

The neuron which is stimulated by the strongest total input signal will fire by generating an output with the constant value d. Therefore, for the winning neuron, $g(Y_j)$ equals d. When a winning neuron is determined, all the other neurons will be prohibited from firing. The value d will be used to multiply the top-down expectation of the firing class before the top-down expectation pattern is read out for comparison in the vigilance test. When the winning neuron fires, all the other neurons are inhibited from firing so it can be inferred that when there is a firing neuron (say j), Equation (38) becomes:

(42) $$p_i = u_i + dz_{ji}$$

otherwise if there is no winning neuron, it can be simplified as:

(43) $$p_i = u_i$$

The top-down expectation pattern is merged with the enhanced input pattern at point r_i before they enter the vigilance test (see Figure 12). r_i is defined by

(44) $$r_i = \frac{q_i' + cp_i}{\|Q'\| + \|cP\|}$$

The vigilance test is failed and the firing neuron will be reset if the following condition is true:

$$\frac{\rho}{\|R\|} > 1 \tag{45}$$

where ρ is the vigilance parameter.

On the other hand, if the vigilance test is passed (in other words, the current input pattern can be accepted as a member of the firing neuron), the top-down and the bottom-up weights are updated so that the special features present in the current input pattern can be incorporated into the class exemplar represented by the firing neuron. The updating equations are as follows:

$$\frac{d}{dt} z_{ji} = d\left[p_i - z_{ji}\right] \tag{46}$$

$$\frac{d}{dt} z_{ij} = d\left[p_i - z_{ij}\right] \tag{47}$$

The bottom-up weights are denoted by Z_{ij} and the top-down weights by Z_{ji}. According to the recommendations in [Carpenter and Grossberg, 1987], all the top-down weights should be initialised with the value 0 at the beginning of the learning process. This can be expressed by the following equation:

$$Z_{ji}(0) = 0 \tag{48}$$

This measure is designed to prevent a neuron from being reset when it is allocated to classify an input pattern for the first time. The bottom-up weights are initialised using the equation:

$$Z_{ji}(0) = \frac{1}{(1-d)\sqrt{M}} \tag{49}$$

where M is the number of neurons in the input layer. This number is equal to the dimension of the input vectors. This arrangement ensures that after all the neurons with the top-down expectations similar to the input pattern have been searched, it would be easy for the input pattern to access a previously uncommitted neuron.

2.10 Spiking Neural Network

Experiments with biological neural systems have shown that they use the timing of electrical pulses or "spikes" to encode and transmit information. Spiking neural networks, also known as pulsed neural networks, are attempts at modelling the operation of biological neural systems more closely than is the case with other artificial neural networks.

An example of spiking neural network is shown in Figure 13. Each connection between neurons i and j could contain multiple connections associated with a weight value and delay [Natschläger and Ruf, 1998].

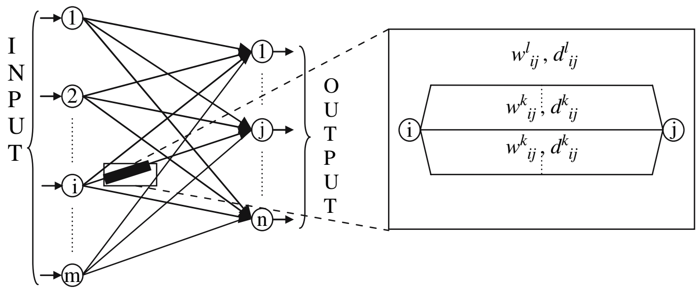
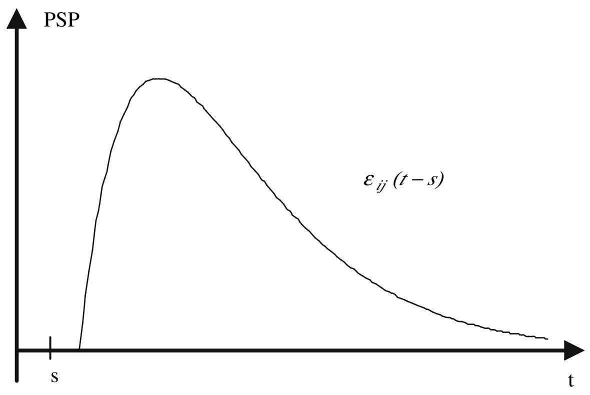
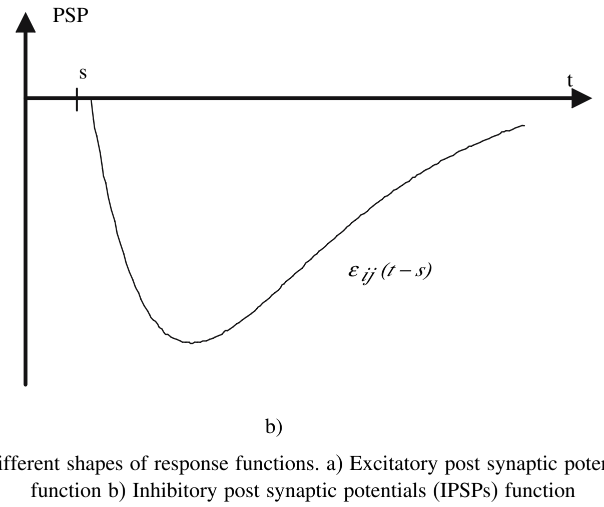


Figure 13. Spiking neural network topology showing a single connection composed of multiple weights w^k_{ij} with corresponding delays d^k_{ij}



a)



b)

Figure 14. Different shapes of response functions. a) Excitatory post synaptic potentials (EPSPs) function b) Inhibitory post synaptic potentials (IPSPs) function

In the leaky integrate-and-fire model proposed by Maass [Maass, 1997], a neuron is regarded as a homogeneous unit that generates spikes when the total excitation exceeds a threshold value.

Consider a network that consists of a finite set V of such spiking neurons, a set $E \subseteq V \times V$ of synapses, a set of weights $W_{u,v} \geq 0$, a response function $\varepsilon_{u,v} : R^+ \rightarrow R$ for each synapse $\langle u, v\rangle \in E$ where $R^+ := \{x \in R : x \geq 0\}$ and a threshold function $\theta_v : R^+ \rightarrow R$ for each neuron $v \in V$. If $F_u \subseteq R^+$ is the set of firing times of a neuron u, then the potential at the trigger zone of each neuron v at time t is given by:

$$(50) \qquad P_v(t) = \sum\nolimits_{u:\langle u,v\rangle \in E} \sum\nolimits_{s \in F_u : s<t} w_{u,v}{}^* \varepsilon_{u,v}(t-s)$$

In the simplest model of a spiking neuron, a neuron v fires whenever its potential $P_v(t)$ reaches a certain threshold $\theta_v(t)$. This potential $P_v(t)$ is the sum of the so-called excitatory post synaptic potentials (EPSPs) and inhibitory post synaptic potentials (IPSPs), which result from the firing of other neurons u that are connected through a synapse to neuron v. The firing of a presynaptic neuron u at time s contributes to the potential $P_v(t)$ at time t an amount that is modeled by the term $w_{u,v}{}^* \varepsilon_{u,v}(t-s)$, where $w_{u,v}$ is the weight of the connection between neurons u and v, and $\varepsilon_{u,v}(t-s)$ is the response function. Some biologically realistic shapes of the post synaptic potentials (PSPs) are shown in Figure 14. The change in threshold as a function of time is illustrated in Figure 15.

Learning can be achieved in spiking neural networks as in traditional neural networks for tasks such as classification, pattern recognition and function approximation [Lannella and Back 2001; Bohte et al., 2002a; 2002b]. Different learning

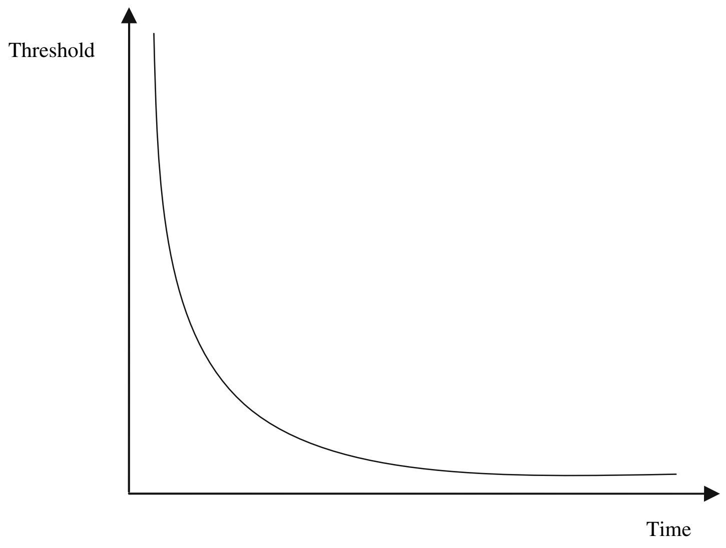


Figure 15. Firing threshold of a neuron

algorithms have been proposed for the training of spiking neural networks [Maass and Bishop, 98; Gerstner and Kistler, 2002].

A supervised learning algorithm has been proposed [Bohte et al., 2002b], where it is shown that a feedforward network of spiking neurons can be trained for classification tasks by means of error backpropagation. Unsupervised learning can also be achieved for spiking neural networks [Bohte et al., 2002a] with self organisation as for Radial Basis Function (RBF) networks.

3. SUMMARY

This chapter has presented the main types of existing neural networks and has described examples of each type. For an overview of the different systems engineering applications of these neural networks, see the chapter on Soft Computing and its Applications in Engineering and Manufacture for example.

4. ACKNOWLEDGEMENTS

This work was carried out within the ALFA project "Novel Intelligent Automation and control systems 11" (NIACS 11), the ERDF (Objective One) projects "Innovation in Manufacturing Centre" (IMC), "Innovative Technologies for Effective Enterprises" (ITEE) and "Supporting Innovative Product Engineering and Responsive Manufacturing" (SUPERMAN) and within the project "Innovative Production Machines and Systems" (I*PROMS).

REFERENCES

Albus J S, (1975a), "A new approach to manipulator control: cerebellar model articulation control (CMAC)", *Trans. ASME, J. of Dynamics Syst., Meas. and Contr.*, **97**, 220–227.

Albus J S, (1975b), "Data storage in the cerebellar model articulation controller (CMAC)", *Trans. ASME, J. of Dynamics Syst., Meas. and Contr.*, **97**, 228–233.

Albus J S, (1979a), "A model of the brain for robot control", *Byte*, 54–95.

Albus J S, (1979b), "Mechanisms of planning and problem solving in the brain", *Math. Biosci.*, **45**, 247–293.

An P E, Brown M, Harris C J, Lawrence A J and Moore C J, (1994), "Associative memory neural networks: adaptive modelling theory, software implementations and graphical user", *Engng. Appli. Artif. Intell.*, **7** (1), 1–21.

Bohte S M, La Poutre H and Kok J N, (2002a), "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks", *IEEE Trans. on Neural Networks*, **13** (2), 415–425.

Bohte S M, La Poutre H and Kok J N, (2002b), "Error-back propagation in temporally encoded networks of spiking neurons", *Neuro Computing*, 17–37.

Broomhead D S and Lowe D, (1988), "Multivariable functional interpolation and adaptive networks", *Complex Systems*, **2**, 321–355.

Carpenter G A and Grossberg S, (1987), "ART2: Self-organisation of stable category recognition codes for analog input patterns", *Appl. Optics*, ***26** (23)*, 4919–4930.

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

Cichocki A and Unbahauen R, (1993), *Neural Networks for Optimisation and Signal Processing*, Chichester: Wiley.

Elman J L, (1990), "Finding structure in time", *Cognitive Science*, **14**, 179–211.

Gerstner W and Kistler W M, (2002), *Spiking Neuron Models: Single Neurons, Populations and Plasticity*, Cambridge University Press, UK.

Goldberg D, (1989), *Genetic Algorithms in Search, Optimisation and Machine Learning*, Reading, MA: Addison-Wesley.

Hassoun M H, (1995), *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, MA.

Haykin S, (1999), *Neural Networks: A Comprehensive Foundation*, 2nd Edition, Upper Saddle River, NJ: Prentice Hall.

Hecht-Nielsen R, (1990), *Neurocomputing*, Reading, MA: Addison-Wesley.

Holland J H, (1975), *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press.

Hopfield J J, (1982), "Neural networks and physical systems with emergent collective computational abilities", *Proc. National Academy of Sciences*, **79**, 2554–2558.

Iannella N and Back A D, (2001), Spiking neural network architecture for nonlinear function approximation, *Neural Networks, Special Issue,* **14**(6), 922–931.

Jordan M I, (1986), "Attractor dynamics and parallelism in a connectionist sequential machines", *Proc. 8th Annual Conf. of the Cognitive Science Society*, 531–546.

Karaboga D, (1994), *Design of Fuzzy Logic Controllers Using Genetic Algorithms*, PhD thesis, University of Wales, Cardiff, UK.

Kohonen T, (1989), *Self-Organising and Associative Memory* (3rd ed.), Berlin: Springer-Verlag.

Lannella N and Back A D, (2001), Spiking neural network architecture for nonlinear function approximation, *Neural Networks, Special Issue*, **14**(16), 922-931.

Maass W, (1997), "Networks of spiking neurons: The third generation of neural network models", *Neural Networks*, **10**, 1659–1671.

Maass W and Bishop CM, (1998), *Pulsed Neural Networks*, Cambridge: MIT Press.

Moody J and Darken C J, (1989), "Fast learning in networks of locally-tuned processing units", *Neural Computation*, **1** (2), 281–294.

Natschläger T and Ruf B, (1998), "Spatial and temporal pattern analysis via spiking neurons", *Network: Computation in Neural systems*, **9** (3), 319–332.

Pham D T and Chan A J, (1998), "Control chart pattern recognition using a new type of self-organising neural network", *Proc. of the Institution of Mechanical Engineers*, **212** (Part I), 115–127.

Pham D T and Chan A J, (2001), "Unsupervised adaptive resonance theory neural networks for control chart pattern recognition", *Proc. of the Institution of Mechanical Engineers*, **215** (Part B), 59–67.

Pham D T and Karaboga D, (1993), "Dynamic system identification using recurrent neural networks and genetic algorithms", *Proc. 9th Int. Conf. on Mathematical and Computer Modelling*, San Francisco.

Pham D T and Liu X, (1992), "Dynamic system modelling using partially recurrent neural networks", *Journal of Systems Engineering*, **2** (2), 90–97.

Pham D T and Liu X, (1994), "Modelling and prediction using GMDH networks of Adalines with nonlinear preprocessors", *Int. J. Systems Science*, **25** (11), 1743–1759.

Pham D T and Oh S J, (1992), "A recurrent backpropagation neural network for dynamic system identification", *Journal of Systems Engineering*, **2** (4), 213–223.

Pham D T and Oztemel E, (1994), "Control chart pattern recognition using learning vector quatization networks", *Int. J. Production Research*, **32** (3), 721–729.

Rumelhart D and McClelland J, (1986), *Parallel distributed processing: exploitations in the micro-structure of cognition*, volumes **1** and **2**, Cambridge: MIT Press.

Widrow B and Hoff M E, (1960), "Adaptive switching circuits", *Proc. 1960 IRE WESCON Convention Record*, Part 4, IRE, New York, 96–104.