

2 Basic structures and properties of Artificial Neural Networks

2.1 A simplistic model of a biological neuron

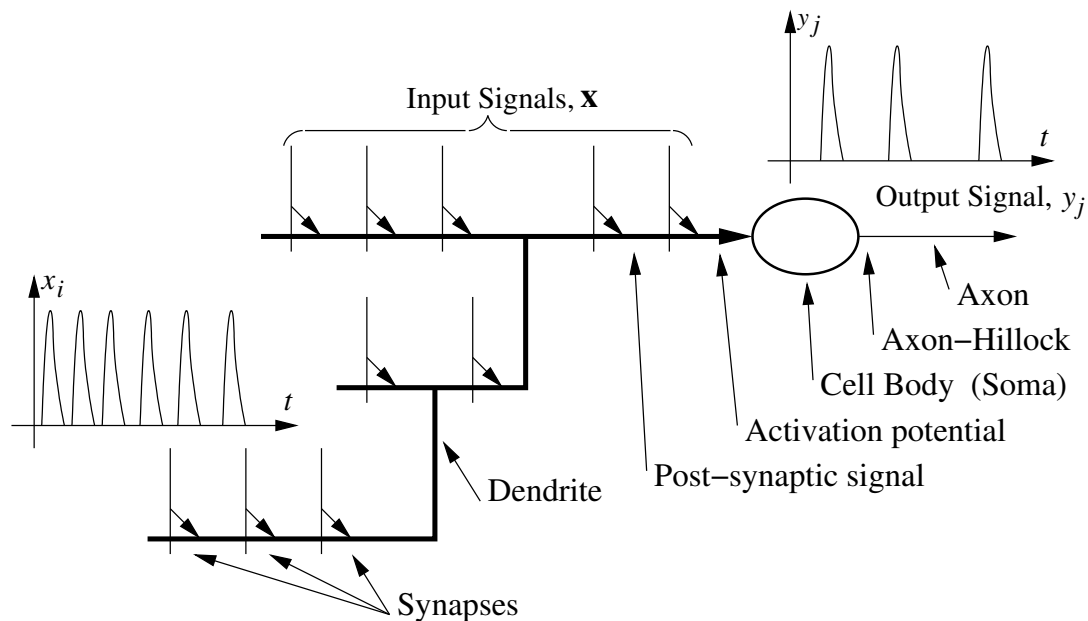


Figure 2–1: Conceptual structure of a biological neuron

Basic characteristics of a biological neuron:

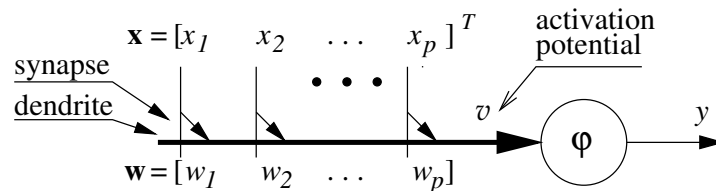
- data is coded in a form of instantaneous frequency of pulses
- synapses are either excitatory or inhibitory
- Signals are aggregated (“summed”) when travel along dendritic trees
- The cell body (neuron output) generates the output pulse train of an average frequency proportional to the total (aggregated) post-synaptic activity (activation potential).

2.2 Models of artificial neurons

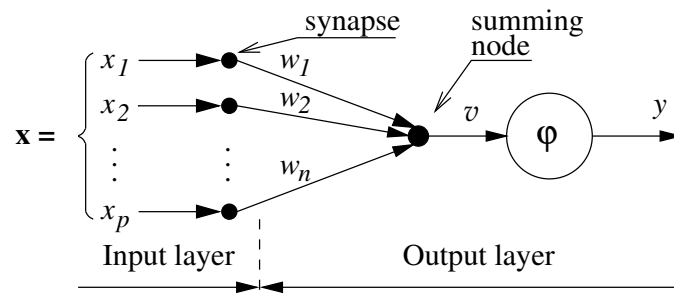
Artificial neural networks are nonlinear information (signal) processing devices which are built from interconnected elementary processing devices called neurons.

An artificial neuron is a p -input single-output signal processing element which can be thought of as a simple model of a non-branching biological neuron. Graphically, an artificial neuron is represented in one of the following forms:

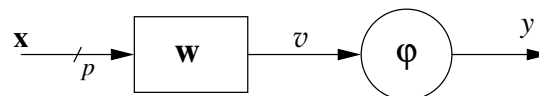
a. Dendritic representation



b. Signal flow graph



c. Block-diagram representation



$$v = w_1 \cdot x_1 + \cdots + w_p \cdot x_p = \mathbf{w} \cdot \mathbf{x} ; \quad y = \varphi(v)$$

Figure 2–2: Three basic graphical representations of a single p -input (p -synapse) neuron

From a **dendritic representation** of a single **neuron** we can identify p **synapses** arranged along a linear **dendrite** which aggregates the synaptic activities, and a neuron body or axon-hillock generating an output signal.

The **pre-synaptic activities** are represented by a p -element **column vector** of input signals

$$\mathbf{x} = [x_1 \ \dots \ x_p]^T$$

In other words the space of input patterns is p -dimensional.

Synapses are characterised by adjustable parameters called weights or synaptic strength parameters. The weights are arranged in a p -element **row vector**:

$$\mathbf{w} = [w_1 \ \dots \ w_p]$$

In a **signal flow** representation of a neuron p synapses are arranged in a layer of input **nodes**. A dendrite is replaced by a single summing node. Weights are now attributed to **branches** (connections) between input nodes and the summing node.

Passing through synapses and a dendrite (or a summing node), input signals are aggregated (combined) into the **activation potential**, which describes the total **post-synaptic activity**. The activation potential is formed as a linear combination of input signals and synaptic strength parameters, that is, as an **inner product** of the weight and input vectors:

$$v = \sum_{i=1}^p w_i x_i = \mathbf{w} \cdot \mathbf{x} = \begin{bmatrix} w_1 & w_2 & \dots & w_p \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \quad (2.1)$$

Subsequently, the activation potential (the total post-synaptic activity) is passed through an **activation function**, $\varphi(\cdot)$, which generates the output signal:

$$y = \varphi(v) \quad (2.2)$$

The activation function is typically a saturating function which normalises the total post-synaptic activity to the standard values of output (axonal) signal.

The **block-diagram** representation encapsulates basic operations of an artificial neuron, namely, aggregation of pre-synaptic activities, eqn (2.1), and generation of the output signal, eqn (2.2)

A single synapse in a dendritic representation of a neuron can be represented by the following block-diagram:

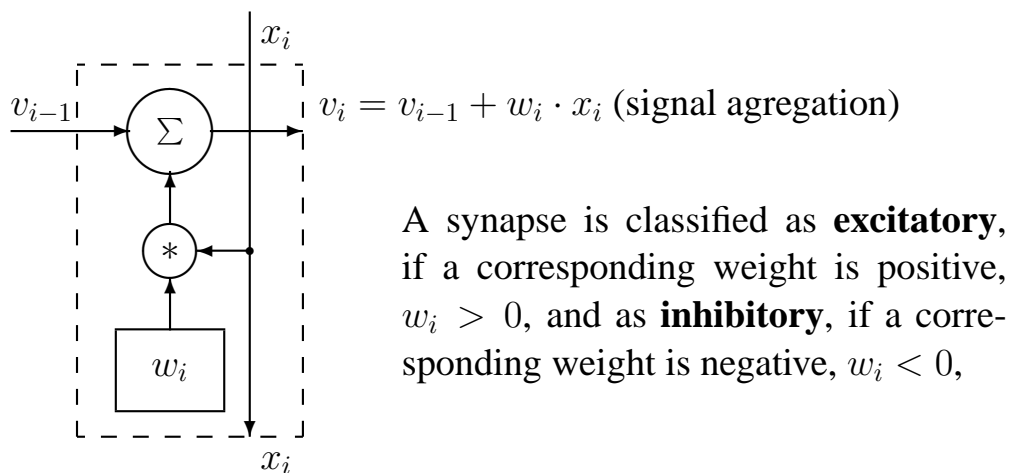


Figure 2–3: A synapse in a dendritic representation of a neuron

In the synapse model of Figure 2–3 we can identify: a storage for the synaptic weight, augmentation (multiplication) of the pre-synaptic signal with the weight parameter, and the dendritic aggregation of the post-synaptic activities.

It is sometimes convenient to add an additional parameter called **threshold**, θ or **bias** $b = -\theta$. It can be done by fixing one input signal to be constant. Then we have

$$x_p = +1, \text{ and } w_p = b = -\theta$$

With this addition, the activation potential is calculated as:

$$\hat{v} = \sum_{i=1}^p w_i x_i = v - \theta, \quad v = \sum_{i=1}^{p-1} w_i x_i$$

where \hat{v} is the augmented activation potential.

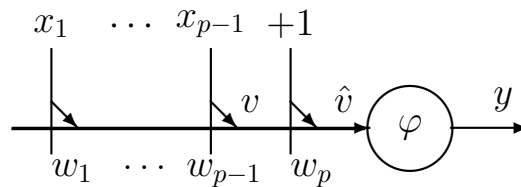


Figure 2–4: A single neuron with a biasing input

2.3 Types of activation functions

Typically, the activation function generates either **unipolar** or **bipolar** signals.

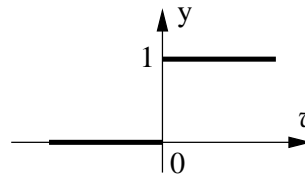
A linear function: $y = v$.

Such linear processing elements, sometimes called ADALINEs, are studied in the theory of linear systems, for example, in the “traditional” signal processing and statistical regression analysis.

A step function

unipolar:

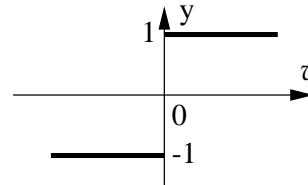
$$y = \varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$



Such a processing element is traditionally called **perceptron**, and it works as a threshold element with a binary output.

bipolar:

$$y = \varphi(v) = \begin{cases} +1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$



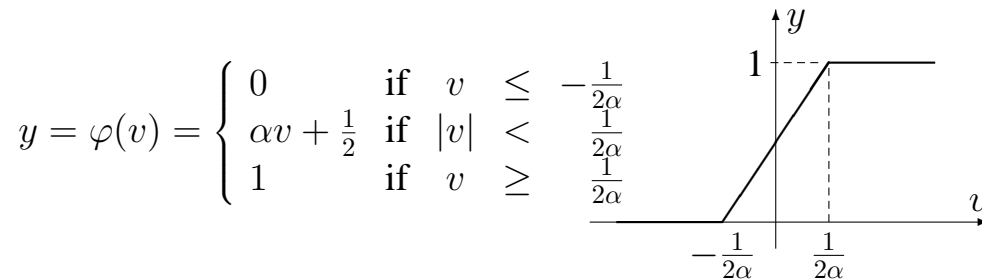
A step function with bias

The bias (threshold) can be added to both, unipolar and bipolar step function. We then say that a neuron is “fired”, when the synaptic activity exceeds the threshold level, θ . For a unipolar case, we have:

$$y = \varphi(v) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq \theta \\ 0 & \text{if } \mathbf{w} \cdot \mathbf{x} < \theta \end{cases}$$

(The McCulloch-Pitts perceptron — 1943)

A piecewise-linear function



- For small activation potential, v , the neuron works as a linear combiner (an ADALINE) with the gain (slope) α .
- For large activation potential, v , the neuron saturates and generates the output signal either 0 or 1.
- For large gains $\alpha \rightarrow \infty$, the piecewise-linear function is reduced to a step function.

Sigmoidal functions

unipolar:

$$\varphi(v) = \frac{1}{1 + e^{-v}} = \frac{1}{2}(\tanh(v/2) + 1)$$

bipolar:

$$\varphi(v) = \tanh(\beta v)$$

The parameter β controls the slope of the function.

The hyperbolic tangent (bipolar sigmoidal) function is perhaps the most popular choice of the activation function specifically in problems related to function mapping and approximation.

Radial-Basis Functions

Radial-basis functions arise as optimal solutions to problems of interpolation, approximation and regularisation of functions. The optimal solutions to the above problems are specified by some integro-differential equations which are satisfied by a wide range of nonlinear differentiable functions (Ch. 5, [?]).

Typically, Radial-Basis Functions $\varphi(\mathbf{x}; \mathbf{t}_i)$ form a family of functions of a p -dimensional vector, \mathbf{x} , each function being centered at point \mathbf{t}_i .

A popular simple example of a Radial-Basis Function is a symmetrical multivariate Gaussian function which depends only on the distance between the current point, \mathbf{x} , and the center point, \mathbf{t}_i , and the variance parameter σ_i :

$$\varphi(\mathbf{x}; \mathbf{t}_i) = G(\|\mathbf{x} - \mathbf{t}_i\|) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{t}_i\|^2}{2\sigma_i^2}\right)$$

where $\|\mathbf{x} - \mathbf{t}_i\|$ is the norm of the distance vector between the current vector \mathbf{x} and the centre, \mathbf{t}_i , of the symmetrical multidimensional Gaussian surface. The spread of the surface is controlled by the variance parameter σ_i .

Two concluding remarks:

- In general, the smooth activation functions, like sigmoidal, or Gaussian, for which a continuous derivative exists, are typically used in networks performing a function approximation task, whereas the step functions are used as parts of pattern classification networks.
- Many learning algorithms require calculation of the derivative of the activation function see the relevant assignments/practical.

2.4 A layer of neurons

Neurons of Figure 2–2 can be arrange into a **layer** of neurons. This can be thought of as a p -input **single layer neural network** consisting of m neurons. Similarly to a single neuron, the network can be represented in all three basic forms: **dendritic**, **signal-flow**, and **block-diagram** form as presented in Figure 2–5.

From the **dendritic representation** of the neural network in Figure 2–5 it is readily seen that a layer of neurons is described by a $m \times p$ matrix W of synaptic weights. Each row of the **weight matrix** is associated with one neuron. Operations performed by the network can be described as follows:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1p} \\ w_{21} & \cdots & w_{2p} \\ \vdots & \cdots & \vdots \\ w_{m1} & \cdots & w_{mp} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} ; \quad \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \varphi(v_1) \\ \varphi(v_2) \\ \vdots \\ \varphi(v_m) \end{bmatrix}$$

or equivalently in a matrix form as:

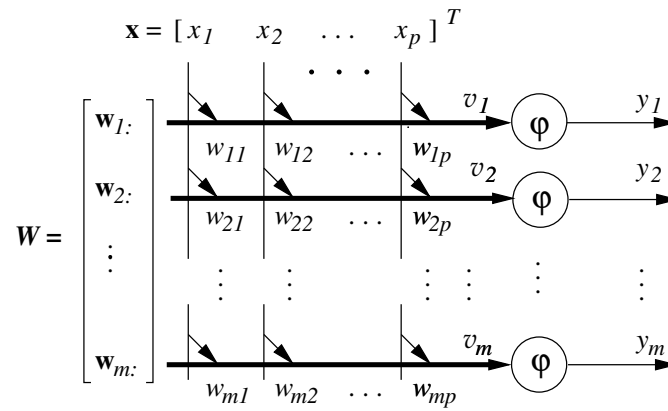
$$\mathbf{v} = W \cdot \mathbf{x} ; \quad \mathbf{y} = \varphi(W \cdot \mathbf{x}) = \varphi(\mathbf{v})$$

where \mathbf{v} is a vector of activation potentials.

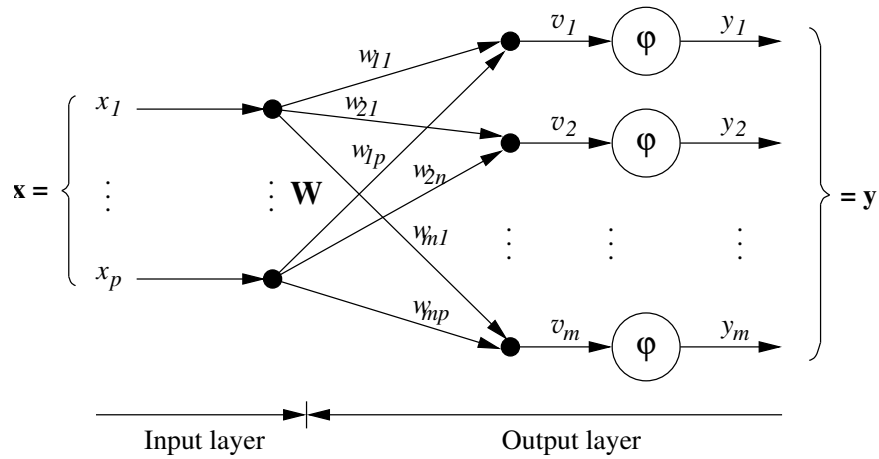
From the **signal-flow graph** it is visible that each weight parameter w_{ij} (synaptic strength) is now related to a connection between nodes of the input layer and the output layer. Therefore, the name **connection strengths** for the weights is also justifiable.

The **block-diagram** representation of the single layer neural network is the most compact one, hence most convenient to use.

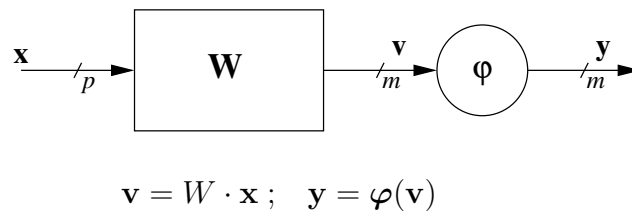
a. Dendritic representation



b. Signal-flow graph



c. Block-diagram

Figure 2-5: Three basic graphical representations of a p -input m -neuron single layer neural network

2.5 Multi-layer feedforward neural networks

Connecting in a serial way layers of neurons presented in Figure 2–5 we can build multi-layer feedforward neural networks.

The most popular neural network seems to be the one consisting of two layers of neurons as presented in Figure 2–6. In order to avoid a problem of counting an input layer, the architecture of Figure 2–6 is referred to as a **single hidden layer** neural network.

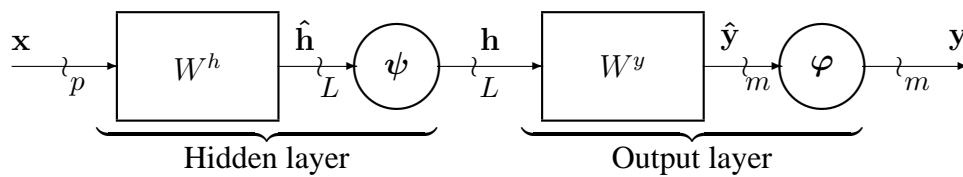


Figure 2–6: A block-diagram of a single-hidden-layer feedforward neural network

There are L neurons in the hidden layer (hidden neurons), and m neurons in the output layer (output neurons).

Input signals, \mathbf{x} , are passed through synapses of the hidden layer with connection strengths described by the **hidden weight matrix**, W^h , and the L **hidden activation signals**, $\hat{\mathbf{h}}$, are generated.

The hidden activation signals are then normalised by the functions ψ into the L **hidden signals**, \mathbf{h} .

Similarly, the hidden signals, \mathbf{h} , are first, converted into m **output activation signals**, $\hat{\mathbf{y}}$, by means of the output weight matrix, W^y , and subsequently, into m **output signals**, \mathbf{y} , by means of the functions φ . Hence

$$\mathbf{h} = \psi(W^h \cdot \mathbf{x}) , \quad \mathbf{y} = \varphi(W^y \cdot \mathbf{h})$$

If needed, one of the input signals and one of the hidden signals can be constant. Functions ψ and φ can be identical.

We will be discussing such a network in detail in the subsequent lectures.

2.6 Static and Dynamic Systems — General Concepts

Static systems

Neural networks considered in previous sections belong to the class of **static** systems which can be fully described by a set of m -functions of p -variables as in Figure 2–7.

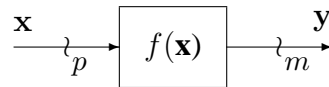


Figure 2–7: A static system: $y = f(x)$

The defining feature of the static systems is that they are **time-independent** — current outputs depends only on the current inputs in the way specified by the mapping function, f . This function can be very complex.

Dynamic systems — Recurrent Neural Networks

In the dynamic systems, the current output signals depend, in general, on current and past input signals. There are two equivalent classes of dynamic systems: continuous-time and discrete-time systems.

The dynamic neural networks are referred to as **recurrent neural networks**.

2.6.1 Continuous-time dynamic systems

Continuous-time dynamic systems operate with signals which are functions of a continuous variable, t , interpreted typically as **time**. A **spatial variable** can be also used.

Continuous-time dynamic systems are described by means of **differential equations**. The most convenient yet general description uses only **first-order** differential equations in the following form:

$$\dot{\mathbf{y}}(t) = f(\mathbf{x}(t), \mathbf{y}(t)) \quad (2.3)$$

where

$$\dot{\mathbf{y}}(t) \stackrel{\text{df}}{=} \frac{d\mathbf{y}(t)}{dt}$$

is a vector of time derivatives of output signals.

In order to model a dynamic system, or to obtain the output signals, the **integration** operation is required. The dynamic system of eqn (2.3) is illustrated in Figure 2–8.

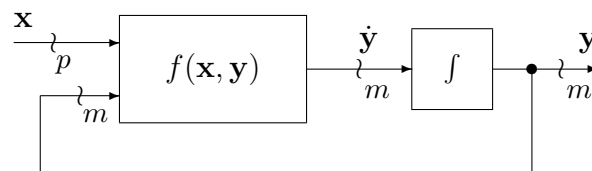


Figure 2–8: A continuous-time dynamic system: $\dot{\mathbf{y}}(t) = f(\mathbf{x}(t), \mathbf{y}(t))$

It is evident that **feedback** is inherent to dynamic systems.

2.6.2 Discrete-time dynamic systems

Discrete-time dynamic systems operate with signals which are functions of a discrete variable, n , interpreted typically as time, but a discrete spatial variable can be also used.

Typically, the discrete variable can be thought of as a **sampled** version of a continuous variable:

$$t = n \cdot t_s ; \quad t \in \mathcal{R} , \quad n \in \mathcal{N}$$

and t_s is the **sampling time**

Analogously, discrete-time dynamic systems are described by means of **difference equations**. The most convenient yet general description uses only **first-order** difference equations in the following form:

$$\mathbf{y}(n+1) = f(\mathbf{x}(n), \mathbf{y}(n)) \quad (2.4)$$

where $\mathbf{y}(n+1)$ and $\mathbf{y}(n)$ are the predicted (future) value and the current value of the vector \mathbf{y} , respectively.

In order to model a discrete-time dynamic system, or to obtain the output signals, we use the **unit delay** operator, $D = z^{-1}$ which originates from the z-transform used to obtain analytical solutions to the difference equations. Using the delay operator, we have

$$z^{-1}\mathbf{y}(n+1) = \mathbf{y}(n)$$

which leads to the structure as in Figure 2–9.

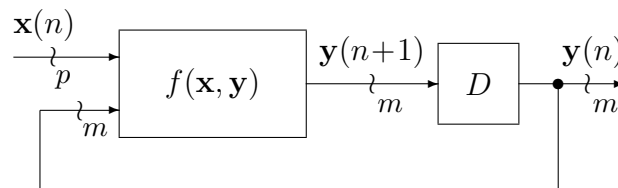


Figure 2–9: A discrete-time dynamic system: $\mathbf{y}(n+1) = f(\mathbf{x}(n), \mathbf{y}(n))$

Notice that **feedback** is also present in the discrete dynamic systems.

2.6.3 Example: A continuous-time generator of a sinusoid

As a simple example of a continuous-time dynamic system let us consider a simple linear system which generates a sinusoidal signal. Eqn (2.3) takes on the following form:

$$\dot{\mathbf{y}}(t) = A \cdot \mathbf{y}(t) + B \cdot \mathbf{x}(t) \quad (2.5)$$

where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} ; \quad A = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix} ; \quad B = \begin{bmatrix} 0 \\ b \end{bmatrix} ; \quad \mathbf{x} = \delta(t)$$

$\delta(t)$ is the unit impulse which is non-zero only for $t = 0$ and is used to describe the initial condition.

In order to show that eqn (2.5) really describes the sinusoidal generator we re-write this equation for individual components. This yields:

$$\begin{aligned} \dot{y}_1 &= \omega y_2 \\ \dot{y}_2 &= -\omega y_1 + b \delta(t) \end{aligned} \quad (2.6)$$

Differentiation of the first equation and substitution of the second one gives the second-order linear differential equation for the output signal y_1 :

$$\ddot{y}_1 + \omega^2 y_1 = \omega b \delta(t)$$

Taking the Laplace transform and remembering that $\mathcal{L}\delta(t) = 1$, we have:

$$y_1(s) = b \frac{\omega}{s^2 + \omega^2}$$

Taking the inverse Laplace transform we finally have

$$y_1(t) = b \sin(\omega t)$$

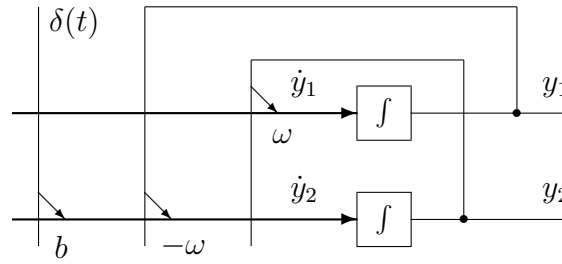


Figure 2–10: A continuous-time sinusoidal generator

The internal structure of the generator can be obtained from eqns (2.7) and illustrated using the dendritic representation as in Figure 2–10. The generator can be thought of as a simple example of a linear recurrent neural network with the fixed weight matrix of the form:

$$W = [B \ A] = \begin{bmatrix} 0 & 0 & \omega \\ b & -\omega & 0 \end{bmatrix}$$

The weights were designed appropriately rather than “worked out” during the learning procedure.

2.6.4 Example: A discrete-time generator of a sinusoid

It is possible to build a discrete-time version of the sinusoidal generator using difference equations of the general form as in eqn (2.4):

$$\mathbf{y}(n+1) = A \cdot \mathbf{y}(n) + B \cdot \mathbf{x}(n) \quad (2.7)$$

where

$$A = \begin{bmatrix} \cos \Omega & \sin \Omega \\ -\sin \Omega & \cos \Omega \end{bmatrix}; \quad B = \begin{bmatrix} 0 \\ b \end{bmatrix}; \quad \mathbf{x}(n) = \delta(n)$$

This time we take the z-transform directly of eqn (2.7), which gives:

$$(zI - A)\mathbf{y}(z) = B; \quad \text{where } I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{and } \mathcal{Z}\delta(n) = 1$$

Hence

$$\mathbf{y}(z) = (zI - A)^{-1}B$$

and subsequently

$$\mathbf{y}(z) = \left(\begin{bmatrix} z - \cos \Omega & \sin \Omega \\ -\sin \Omega & z - \cos \Omega \end{bmatrix} \begin{bmatrix} 0 \\ b \end{bmatrix} \right) / (z^2 - 2z \cos \Omega + 1)$$

Extracting the first component, we have

$$y_1(z) = \frac{b \sin \Omega}{z^2 - 2z \cos \Omega + 1}$$

Taking the inverse z-transform finally yields

$$y_1(n+1) = b \sin(\Omega n)$$

which means that the discrete-time dynamic system described by eqn (2.7) generates a sinusoidal signal.

The structure of the generator is similar to the previous one and is presented in Figure 2–11.

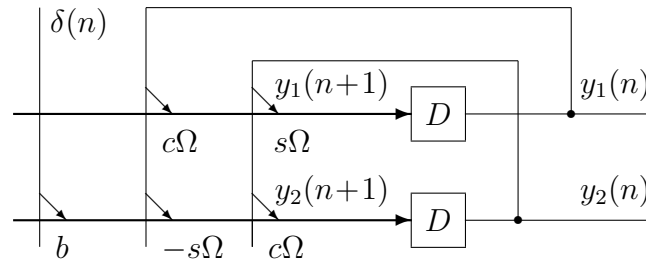


Figure 2–11: A discrete-time sinusoidal generator

This time the weight matrix is:

$$W = [B \ A] = \begin{bmatrix} 0 & c\Omega & s\Omega \\ b & -s\Omega & c\Omega \end{bmatrix}$$

where

$$s\Omega = \sin \Omega, \quad \text{and} \quad c\Omega = \cos \Omega$$

2.7 Introduction to learning

In the previous sections we concentrated on the **decoding** part of a neural network assuming that the **weight matrix**, W , is given. If the weight matrix is satisfactory, during the decoding process the network performs some useful task it has been design to do.

In simple or specialised cases the weight matrix can be pre-computed, but more commonly it is obtained through the **learning** process.

Learning is a dynamic process which modifies the weights of the network in some desirable way. As any dynamic process learning can be described either in the continuous-time or in the discrete-time framework.

Consider a neural network as in Figure 2–12

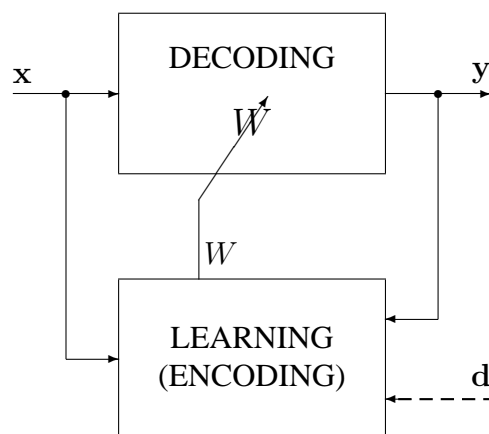


Figure 2–12: A neural network with a learning part

The learning can be described either by differential equations (continuous-time)

$$\dot{W}(t) = L(W(t), \mathbf{x}(t), \mathbf{y}(t), \mathbf{d}(t)) \quad (2.8)$$

or by the difference equations (discrete-time)

$$W(n+1) = L(W(n), \mathbf{x}(n), \mathbf{y}(n), \mathbf{d}(n)) \quad (2.9)$$

where \mathbf{d} is an external teaching/supervising signal used in **supervised learning**. This signal is not present in networks employing **unsupervised learning**.

The discrete-time learning law is often used in a form of a **weight update** equation:

$$\begin{aligned} W(n+1) &= W(n) + \Delta W(n) \\ \Delta W(n) &= L(W(n), \mathbf{x}(n), \mathbf{y}(n), \mathbf{d}(n)) \end{aligned} \quad (2.10)$$