UNT

Department

of Computer Science and Engineering

CSCE 5300: Introduction to Big Data and Data Science

Chapter 6 (6.1): State-of-the-Art Big Data Analytics Architectures and Tools

analytics architectures and tools

20 October 2024

Qutline

- **6.1.1. Neural networks**
- **6.1.2. Network architectures**
- **6.1.3. Learning processes**
- **6.1.4. Simulation example**
- **6.1.5. Concluding remarks**

6.1.1. Neural Networks

- **Artificial neural networks** (or simply neural networks) motivated by the human brain that computes in completely different way from the conventional digital computer.

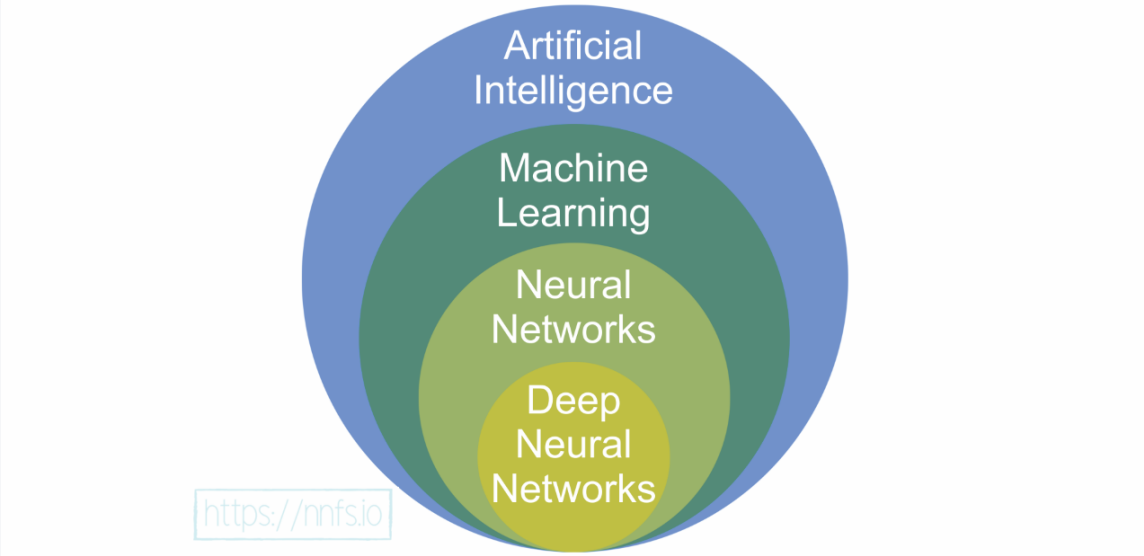
- The brain is a highly ***nonlinear, complex, and parallel computer*** (information-processing system).

- It has the capability to arrange its structural constituents, known as *neurons*, so as to fulfill certain computations (e.g., perception, pattern recognition, and control) many times faster than the fastest digital computer today [1], [2].

- [1] Simon Haykin, *Neural Networks and Learning Machines,* Third Edition, Pearson, New Jersey, NJ, 2009.
- [2] Michael A. Arbib, *The Handbook of Brain Theory and Neural Networks,* Second Edition, The MIT Press, Cambridge, Massachusetts, 2003.

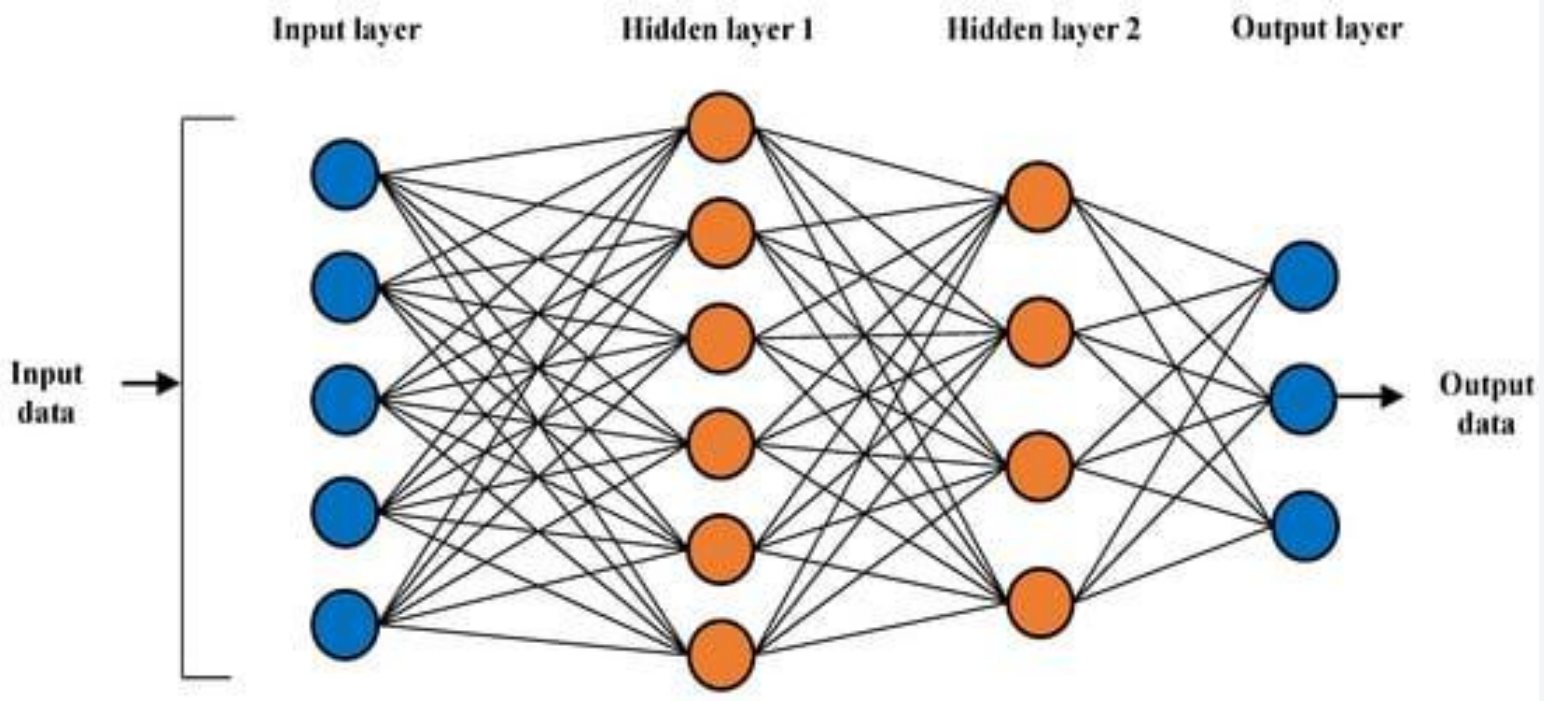
6.1.1. Neural Networks

- Fig. 6.1.



6.1.1. Neural Networks

- Fig. 6.2.



6.1.1. Neural Networks

- Consider, for instance, human ***vision***, which is an information-processing task.

- •It is the function of the visual system to give a ***representation*** of the environment around us and, more important, to supply the information we require to ***interact*** with the environment.

- •To be specific, the brain accomplishes perceptual recognition problems (e.g., recognizing a familiar face embedded in an unfamiliar scene) in approximately 100–200 ms, whereas tasks of much lesser complexity take a great deal longer on a powerful computer.

6.1.1. Neural Networks

- In its general form, a ***neural network*** is a machine that is created to *mimic* the way in which the brain performs a specific task; the network is usually implemented using electronic components or is simulated in software on a digital computer.
- In this chapter, we focus on an important class of neural networks that fulfill useful computations through a process of ***learning***.
- To achieve good performance, neural networks employ a massive interconnection of simple computing units referred to as “neurons”.

- We may thus suggest the following definition of a neural network:

6.1.1. Neural Networks

- ***A neural network is a essentially parallel distributed processor made up of simple processing elements that has a propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:***
- **1.** ***Knowledge is acquired by the network from its environment through a learning process.***
- **2.** ***Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge****.*

- The procedure used to fulfill the learning process is called a *learning algorithm*, the function of which is to update the synaptic weights of the network in an orderly fashion to attain a required design objective.

6.1.1. Neural Networks

- The modification of synaptic weights gives the traditional approach for the design of neural networks.

- •Such an method is the closest to linear adaptive filter theory, which is already well established and successfully applied in many fields.

- •However, it is also possible for a neural network to change its own topology, which is motivated by the fact that neurons in the human brain can die and new synaptic connections may grow.

6.1.1. Neural Networks

- •It is apparent that a neural network obtains its computing power through, first, its massively parallel distributed structure and, second, its ability to learn and therefore generalize.

- •***Generalization*** refers to the neural network's production of reasonable outputs for inputs not encountered within training (learning).

- •These two information processing capabilities make it possible for neural networks to find good approximate solutions to complex (large-scale) tasks that are ***intractable***.

6.1.1. Neural Networks

- •In practice, however, neural networks cannot give the solution by working individually.
- •Rather, they require to be integrated into a consistent system engineering method.
- •In particular, a complex task of interest is *decomposed* into a number of relatively simple tasks, and neural networks are appointed a subset of the tasks that *match* their inherent capabilities.
- •It is important to recognize, however, that we have a long way to go (if ever) before we can build a computer architecture that simulates the human brain.

6.1.1. Neural Networks

- **Neural networks suggest the following useful properties and capabilities:**
- **1. *Nonlinearity.***
- An artificial neuron can be linear or nonlinear. A neural network, fabricated up of an interconnection of nonlinear neurons, is nonlinear.

- The nonlinearity is of a special type in the sense that it is *distributed* throughout the network.

- Nonlinearity is a important property, particularly if the underlying physical mechanism responsible for generation of the input signal is nonlinear.

6.1.1. Neural Networks

- **2. *Input–Output Mapping.***
- A paradigm of learning, called ***learning with a teacher*, or *supervised learning***, involves updating of the synaptic weights of a neural network by applying a set of labeled ***training examples***.
- Each example consists of a unique ***input signal*** and a corresponding ***desired response***.

- The network synaptic weights are modified to reduce the difference between the required response and the actual output of the network produced by the input signal according to an appropriate statistical criterion.

6.1.1. Neural Networks

- The learning of the network is repeated for many examples in the set, until it reaches a steady state, where there are no further significant changes in the synaptic weights.

- Thus the network learns from the examples by creating an ***input–output mapping*** for the problem at hand.

6.1.1. Neural Networks

- •Consider, for instance, a ***pattern classification*** task, where the requirement is to assign an input signal representing a physical object or event to one of several prespecified categories (classes).

- •In a nonparametric method to this problem, the requirement is to “estimate” any decision boundaries in the input signal space for the pattern-classification problem using a set of examples, and to do so *without* invoking a probabilistic distribution model.

- •A similar point of view is implicit in the supervised learning paradigm, which assumes a close analogy between the input–output mapping fulfilled by a neural network and nonparametric statistical inference.

6.1.1. Neural Networks

- **3. *Adaptivity.***
- Neural networks have a built-in ability to ***adapt*** their synaptic weights to changes in the surrounding environment.

- In particular, a neural network learned to function in a specific environment may be ***retrained*** to deal with minor changes in the operating environmental conditions.

- When it is functioning in a ***nonstationary*** environment, a neural network may be built to change its synaptic weights in real time.

6.1.1. Neural Networks

- •The natural architecture of a neural network for signal processing, pattern classification, and control applications, coupled with the adaptive capability of the network, makes it a useful tool in adaptive pattern classification, adaptive signal processing, and adaptive control.
- •As a general rule, it may be said that the more adaptive we create a system, all the time ensuring that the system remains stable, the more robust its performance will likely be when the system is required to operate in a nonstationary environment.
- •It should be emphasized, however, that adaptivity does not always gives robustness; indeed, it may do the very opposite.
- •For instance, an adaptive system with short-time constants can change rapidly and therefore tend to respond to spurious disturbances, causing a drastic degradation in system efficiency.

6.1.1. Neural Networks

- •To realize the full advantages of adaptivity, the principal time constants of the system should be long enough for the system to ignore spurious disturbances, and yet short enough to respond to meaningful variations in the environment; the problem described here is referred to as the ***stability–plasticity dilemma***.

- **4. *Evidential Response.***
- For pattern classification, a neural network can be created to provide information not only about which specific pattern to *select*, but also about the *confidence* in the decision made.
- This latter information may be used to reject some patterns, and thereby improve the classification efficiency of the network.

6.1.1. Neural Networks

- **5. *Contextual Information.***
- Knowledge is represented by the structure and activation state of a neural network.

- Every neuron in the network is affected by the global activity of all other neurons of the network.

- **6. *Fault Tolerance.***
- A neural network, implemented in hardware, has the potential to be inherently ***fault tolerant***, or capable of stable computation, in the sense that its efficiency degrades slowly under adverse operating conditions.

- However, due to the distributed structure of the network, the damage has to be extensive before the overall response of the network is degraded.

6.1.1. Neural Networks

- •Thus, in principle, a neural network has a graceful degradation in performance rather than catastrophic failure.

- •There is some empirical evidence for stable computation, but usually it is uncontrolled.

- •In order to be assured that the neural network is, in fact, fault tolerant, it may be required to take corrective measures in designing the algorithm applied to train the network.

6.1.1. Neural Networks

- ***7. VLSI Implementability.***
- The essentially parallel nature of a network makes it fast for the computation of some tasks.

- This same feature makes a network well equipped for implementation using ***very-large-scale-integrated*** technology.

- One particular beneficial virtue of VLSI is that it gives a means of capturing truly complex behavior in a hierarchical fashion.

6.1.1. Neural Networks

- **8. *Uniformity of Analysis and Design.***
- Neural networks enjoy universality as information processors.

- This feature manifests itself in different ways:
- • Neurons, in one form or another, show an ingredient *common* to all neural networks.
- • This commonality makes it possible to ***share*** theories and training algorithms in different applications of neural networks.
- • Modular networks can be designed through a ***seamless integration of modules***.

6.1.1. Neural Networks

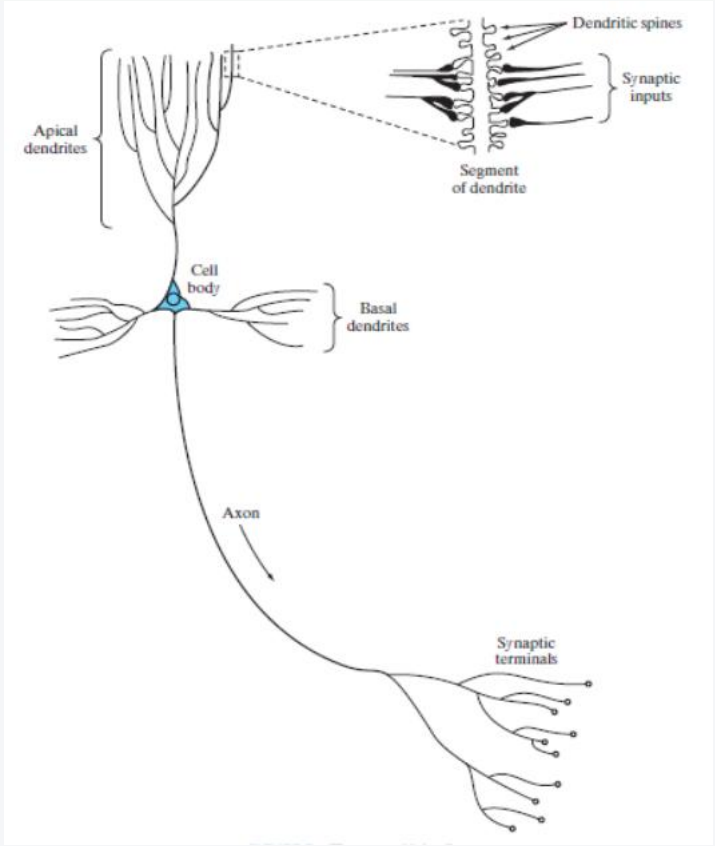
- **9. *Neurobiological Analogy.***
- The design of a network is motivated by analogy with the brain, which is living proof that fault-tolerant parallel processing is not only possible, but also fast and powerful.

- Fig. 6.3 shows the shape of a ***pyramidal cell***, which is one of the common types of cortical neurons.

- Like many other types of neurons, it obtains most of its inputs through dendritic spines (Fig. 6.3);

- The pyramidal cell can obtain 10,000 or more synaptic contacts, and it can project onto thousands of target cells.

6.1.1. Neural Networks



- Fig. 6.3.

6.1.1. Neural Networks

- •The majority of neurons encode their outputs as a series of voltage pulses.
- •These pulses, known as ***action potentials*****, or *spikes***, originate at or close to the cell body of neurons and then propagate across the neurons at constant velocity and amplitude.
- •The reasons for the use of action potentials for communication between neurons are based on the physics of axons.
- •The axon of a neuron is very long and thin and is characterized by high electrical resistance and large capacitance.
- •Both of these elements are distributed along the axon.

6.1.1. Neural Networks

- •The axon can be modeled as resistance-capacitance (RC) transmission line, thus the common use of “cable equation” as the terminology for describing signal propagation along an axon.

- •Analysis of this propagation mechanism shows that when a voltage is applied at one end of the axon, it decays exponentially with distance, dropping to an insignificant level by the time it reaches the other end.

- •The action potentials provide a way to circumvent this transmission problem.

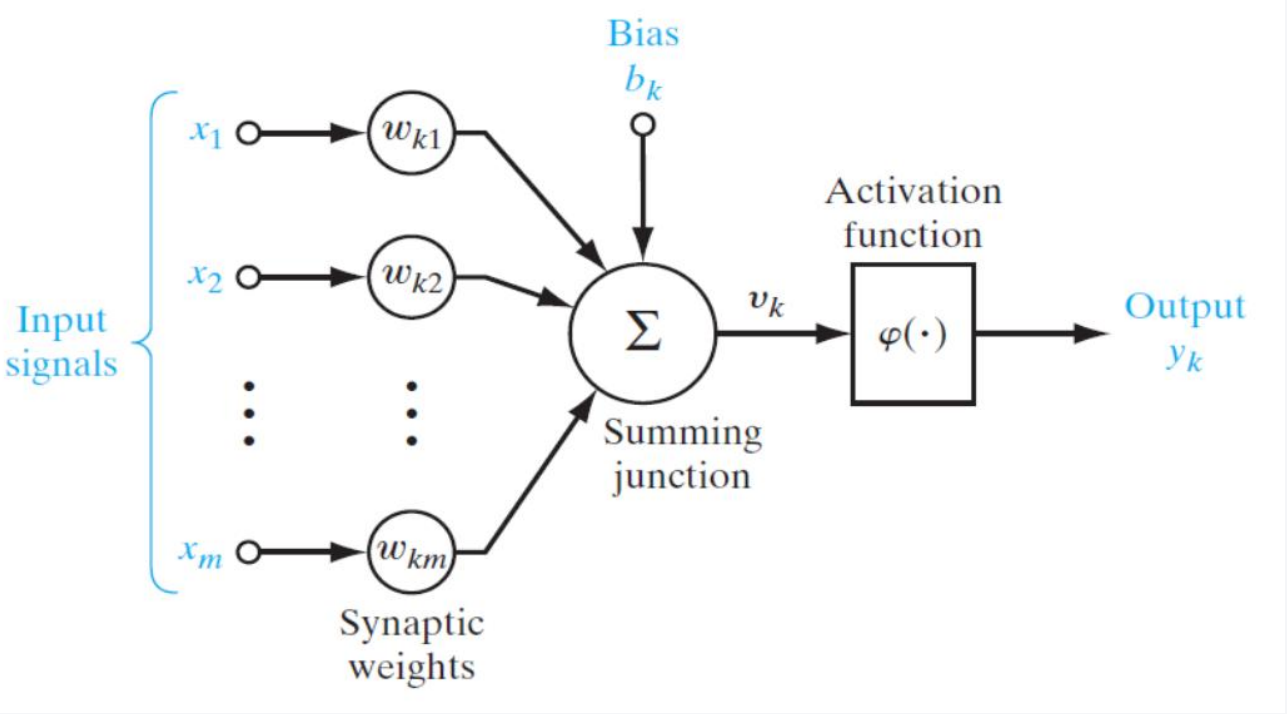
6.1.1. Neural Networks

- A ***neuron*** is an information-processing cell that is fundamental to the operation of a
- neural network.

- The diagram of Fig. 6.4 presents the ***model*** of a neuron.

- Here, we identify three basic cells of the neural model:

6.1.1. Neural Networks



- Fig. 6.4

6.1.1. Neural Networks

- **1.** A set of ***synapses*, or *connecting links***, each of which is characterized by a *weight*.
- In particular, a signal *xj* at the input of synapse *j* connected to neuron *k* is multiplied by the synaptic weight w*kj.*

- The first subscript in w*kj indicates* the neuron in question, and the second subscript indicates the input end of the synapse to which the weight refers.

- Unlike the weight of a synapse in the brain, the synaptic weight of a neuron may lie in a range that includes negative as well as positive values.

6.1.1. Neural Networks

- **2.** An ***adder*** for summing the inputs, weighted by the respective synaptic strengths of the neuron; the operations described here constitute a ***linear combiner***.

- **3.** An ***activation function*** for limiting the amplitude of the output of a neuron.
- The activation function is referred to as a ***squashing function***, in that it squashes the permissible amplitude range of the output to finite value.

- Typically, the normalized amplitude range of the output of a neuron is written as the closed interval [0,1] or [-1,1].

6.1.1. Neural Networks

- The neural model of Fig. 6.4 also has an external *bias*, marked by *bk.*
- The bias *bk* has the effect of rising or decreasing the net input of the activation function, depending on if it is positive or negative.
- We may describe the neuron *k shown* in Fig. 5.45 by the following pair of equations:
- $$u_k = \sum_{j=1}^{m} w_{kj} x_j \qquad (6.1)$$
- and
- $$y_k = \varphi(u_k + b_k) \qquad (6.4)$$

6.1.1. Neural Networks

- where *x*1, *x*2, ..., *xm* are the inputs;
- w*k*1, w*k*2, ..., w*km* are the synaptic weights of neuron *k*;
- *uk* (not shown in Fig. 6.4) is the ***output*** due to the input signals;
- *bk* is the bias;
- „(·) is the ***activation function***; and *yk* is the output of the neuron.

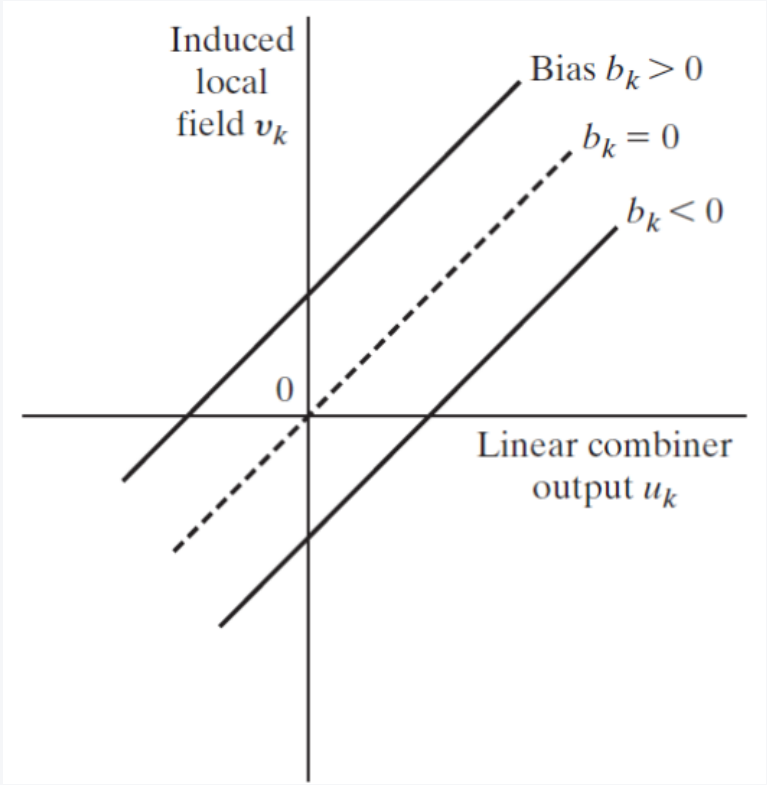
- The use of bias *bk* has the effect of applying an ***affine transformation*** to the output *uk* in the model of Fig. 6.4, given by

- $$v_k = u_k + b_k \qquad (6.5)$$

6.1.1. Neural Networks

- Depending on if the bias *bk* is positive or negative, the relationship between the **or *activation potential***, v*k* of neuron *k* and the output *uk* is modified in the manner illustrated in Fig. 6.5, where v*k*=*bk* at *uk*=0.

6.1.1. Neural Networks



- Fig. 6.5

6.1.1. Neural Networks

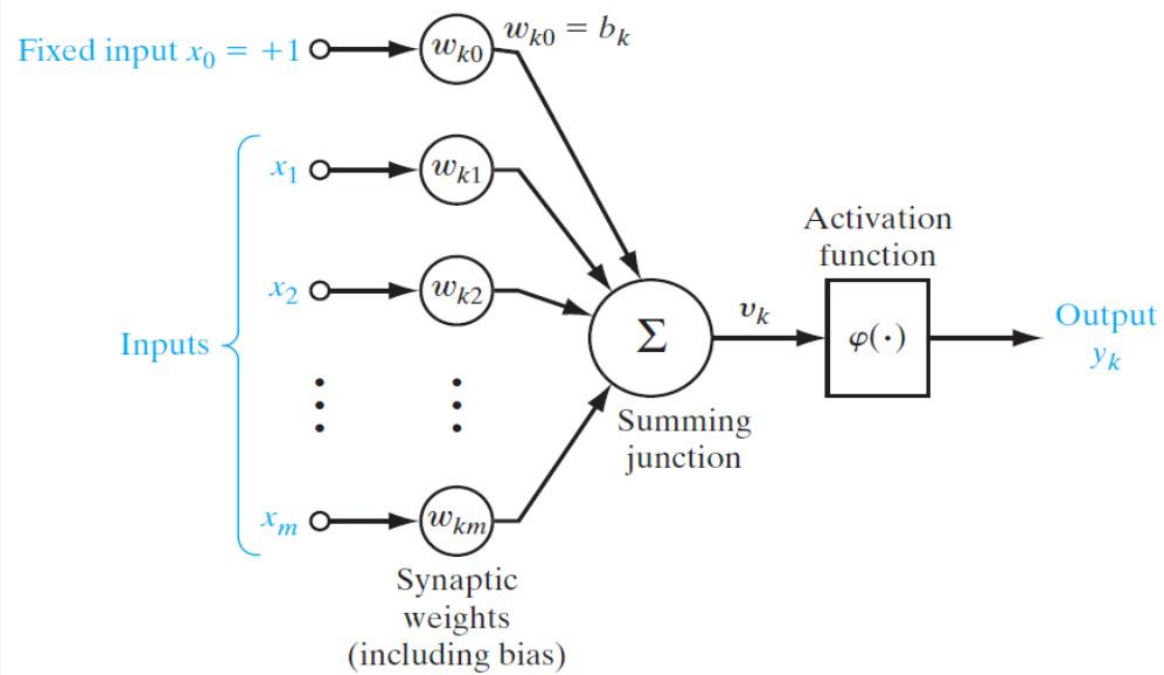
- The bias *bk* is external parameter of neuron *k*.
- Equivalently, we may formulate the combination of Eqs. (6.1) to (6.5) by
- $$v_k = \sum_{j=0}^{m} w_{kj} x_j \qquad (6.6)$$
- and
- $$y_k = \varphi(v_k) \qquad (6.7)$$

6.1.1. Neural Networks

- In Eq. (6.6), we added a new synapse.
- Its input is
- $$x_0 = +1 \tag{6.8}$$
- and its weight is
- $$w_{k0} = b_k \tag{6.9}$$

6.1.1. Neural Networks

- We may therefore reformulate the model of neuron *k* as presented in Fig. 6.6, where wk0 accounts for the bias bk.



- Fig. 6.6

6.1.1. Neural Networks

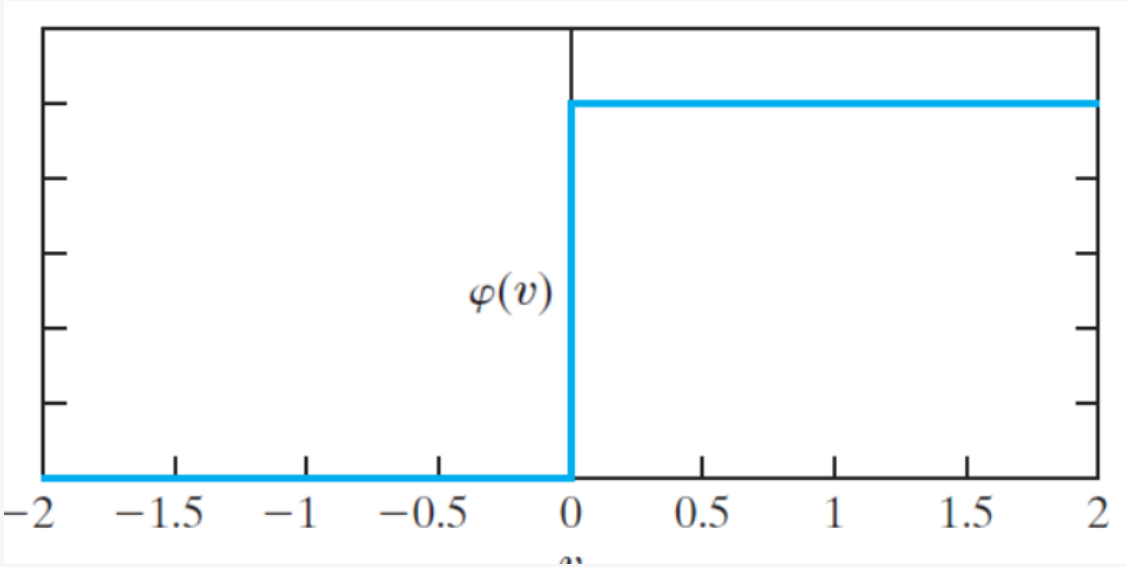
- The effect of the bias is accounted for by doing two things:
- (1) adding a new input fixed at 1,
- and (2) adding a new synaptic weight equal to *bk*.

- Although the models of Figs. 6.5 and 6.6 are different, they are mathematically equivalent.

6.1.1. Neural Networks

- **Types of Activation Function**
- The activation function, denoted by (v), determines the output of a neuron in terms of the induced local field v.
- In what follows, we determine two basic types of activation functions:
- **1. *Threshold Function.*** For this type of activation function, described in Fig. 6.7a,
- we have
- $$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \qquad (6.10)$$

6.1.1. Neural Networks



- Fig. 6.7a.

6.1.1. Neural Networks

- This form of a threshold function is commonly referred to as a ***Heaviside function***.
-
- Correspondingly, the output of neuron *k* employing such a function is expressed as
- $$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases} \qquad (6.11)$$
- where vk is the induced local field of the neuron; that is,
- $$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \qquad (6.12)$$

6.1.1. Neural Networks

- Such a neuron is referred to as the ***McCulloch–Pitts model*** (McCulloch and Pitts (1943)).

- The output of a neuron receives the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise.

6.1.1. Neural Networks

- **2. *Sigmoid Function.*** The sigmoid function, whose graph is “S”-shaped, is the most common form of activation function used in the designing of neural networks.

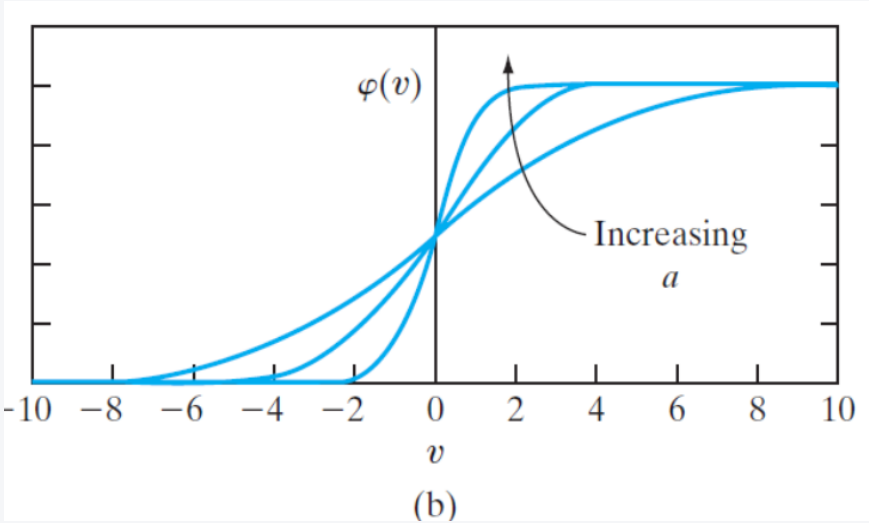
- An example of such function is the *logistic function* given by

- $$\varphi(v) = \frac{1}{1 + \exp(-av)} \qquad (6.13)$$

6.1.1. Neural Networks

- where *a* is the ***slope*** of the sigmoid function.
- By changing the parameter *a*, we get sigmoid functions of different slopes, as shown in Fig. 6.7b.

- Fig. 6.7b.



6.1.1. Neural Networks

- The slope at the origin equals $a/4$.

- As the slope parameter approaches infinity, the sigmoid function becomes a threshold function.
- Whereas a threshold function supposes the value of 0 or 1, a sigmoid function supposes a continuous range of values from 0 to 1.

- The sigmoid function is differentiable, whereas the threshold function is not.

6.1.1. Neural Networks

- The functions defined in Eqs. (6.11) and (5.11) range from 0 to 1.
- It is sometimes required to have the activation function range from -1 to 1, in which case, the activation function is an odd function of the induced local field.
- Specifically, the threshold function of Eq. (6.56) is now defined as
- $$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases} \qquad (6.14)$$
- which is referred to as the ***signum function***.

6.1.1. Neural Networks

- For the corresponding form of a sigmoid function, we can use the ***hyperbolic tangent function*** given by
- $$\varphi(v) = \tanh(v) \qquad (6.15)$$
- Allowing an activation function of the sigmoid type to suppose negative values as prescribed by Eq. (6.15) can yield practical advantages over the logistic function of Eq. (6.13).

- **ReLu:**ReLu stands for Rectified Linear Units. It takes real-valued input and thresholds it to 0 (replaces negative values to 0).
- $$f(x) = max(0, x) \qquad (6.16)$$

6.1.1. Neural Networks

- **Stochastic Model of a Neuron**
- The model described in Fig. 6.6 is deterministic in that its input–output behavior is precisely defined for all inputs.
- For many applications, it is desirable to base the analysis on a stochastic neural model.
- In an analytical approach, the activation function of the McCulloch–Pitts model is given a probabilistic interpretation.
- In particular, a neuron is permitted to reside in only one of two states: +1 or -1.
- The decision for a neuron to *fire* (i.e., switch its state from “off” to “on”) is probabilistic.

6.1.1. Neural Networks

- Let *x* mark the state of the neuron and *P*(v) mark the *probability* of firing, where v is the induced local field of the neuron.
- We can then write
- $$x = \begin{cases} +1 & \text{with probability } P(v) \\ -1 & \text{with probability } 1 - P(v) \end{cases} \qquad (6.17)$$
- A standard choice for *P*(v) is the sigmoid function
- $$P(v) = \frac{1}{1 + \exp(-v/T)} \qquad (6.18)$$

6.1.1. Neural Networks

- where *T* is a ***pseudotemperature*** applied to control the noise level and therefore the uncertainty in firing.

- However, *T* is *not* the physical temperature of a network.

- *T is* a parameter that controls the thermal fluctuations showing the effects of synaptic noise.

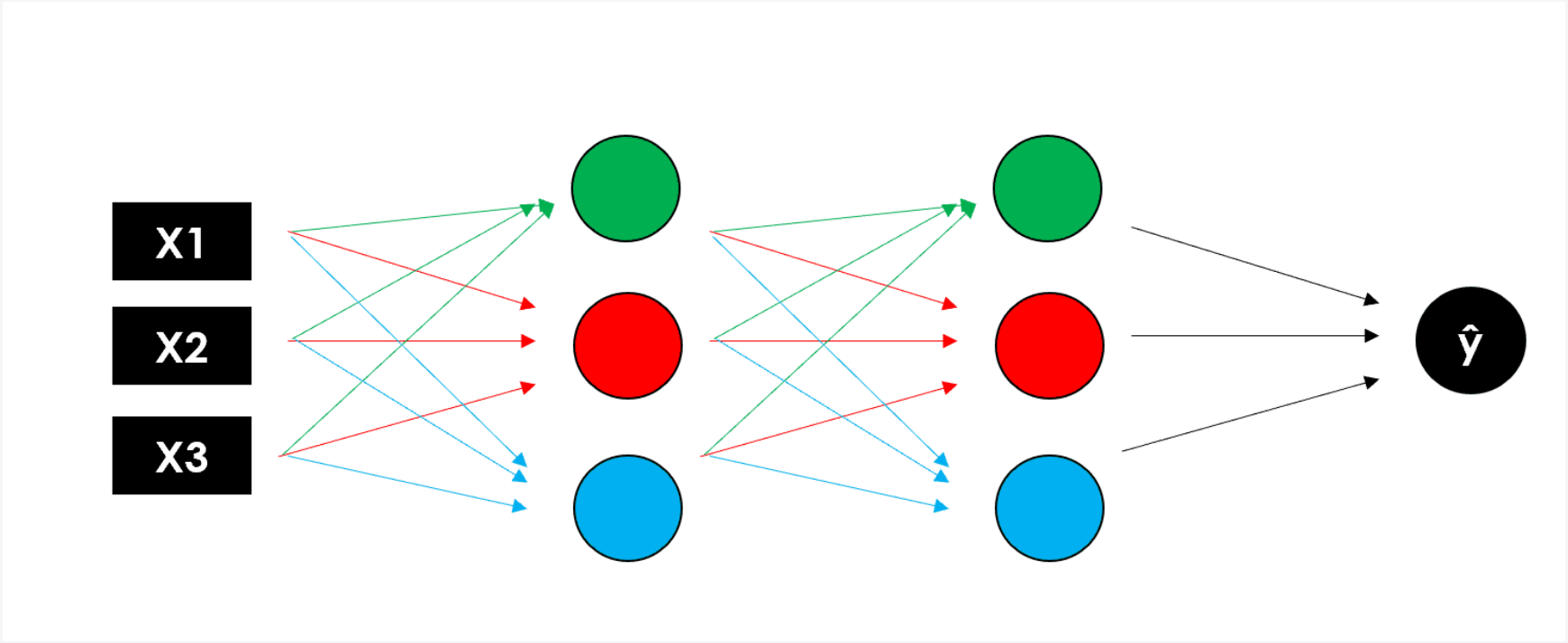
6.1.1. Neural Networks

- •The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm applied to train the network.

- •We may therefore speak of learning algorithms (rules) applied in the design of neural networks as being ***structured***.

- •In general, we can identify three fundamentally different classes of network architectures:

6.1.1. Neural Networks



6.1.1. Neural Networks

- **Feed-Forward Networks**
- Such networks have practical applications in many different areas, in particular, for image classification, speech recognition, object detection, etc.
- Let’s try to understand the basic unit behind all these states of art techniques.

- A single neuron transforms given input into some output.
- Depending on the given input and weights assigned to each input, decide whether the neuron fired or not.

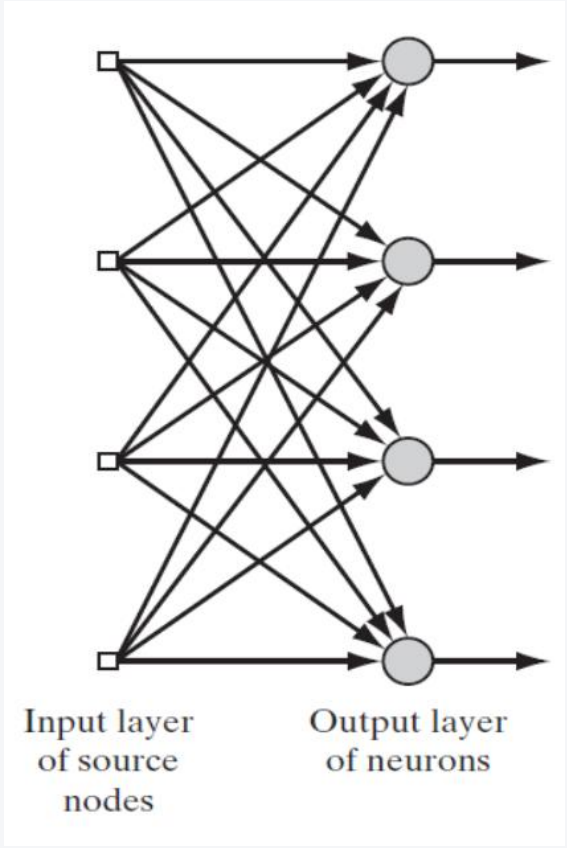
6.1.1. Neural Networks

- The end goal is to find the optimal set of weights for this neuron that produces correct results.
- Do this by training the neuron with several different training examples.
- At each step calculate the error in the output of the neuron, and back-propagate the gradients.
- The step of calculating the output of a neuron is called *forward propagation* while the calculation of gradients is called *back propagation*.

6.1.2. Network Architectures

- **Single-Layer Feed-Forward Networks**
- In a ***layered*** network, the neurons are connected in the form of layers.
- In the simplest form of a layered network, we have an ***input layer*** of source nodes that projects directly onto an ***output layer*** of neurons.
- In other words, this network is a ***feed-forward*** type.
- It is shown in Fig. 6. 6 for the case of four nodes in both the input and output layers.
- Such a network is called a ***single-layer network***, with the designation “single-layer” referring to the output layer of neurons.
- We do not count the input layer of source nodes since no computation is performed there.

6.1.2. Network Architectures



- Fig. 6.8

6.1.2. Network Architectures

- **Multilayer Feed-Forward Networks**
- The second class of a feed-forward network distinguishes itself by the presence of one or more ***hidden layers***, whose computation nodes are called ***hidden neurons***; the term “hidden” refers to the fact that this part of the network is not seen directly from either the input or output of the network.

- The function of hidden neurons is to intervene between the external input and the network output.

6.1.2. Network Architectures

- The output signals of the second layer are used as inputs to the third layer, and so on for the network.

- The neurons in each layer of the network have as their inputs the output signals of the preceding layer.

- The set of output signals of the neurons in the output layer of the network constitutes the overall response of the network to the activation supplied by the source nodes in the input layer.

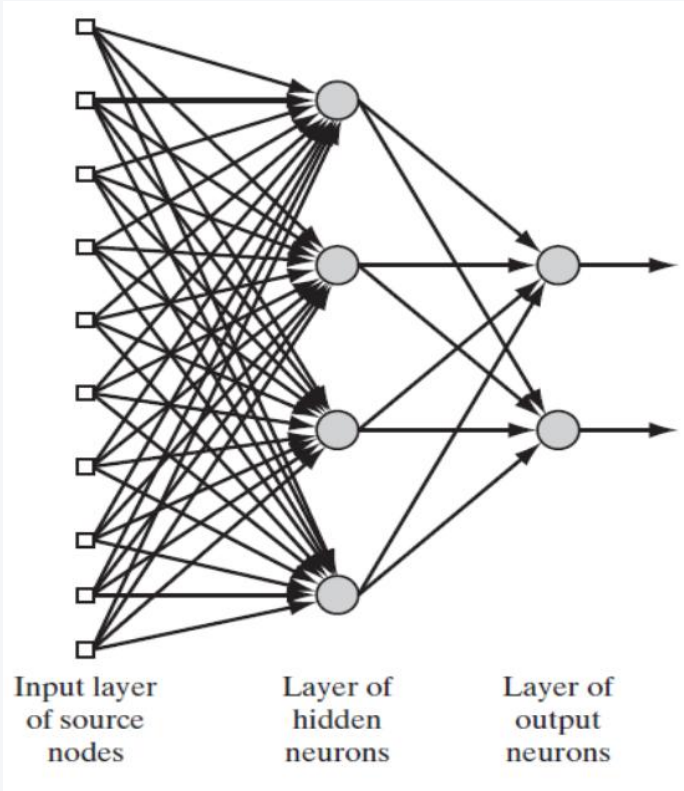
6.1.2. Network Architectures

- The graph in Fig. 6. 7 shows the layout of a multilayer feed-forward network (perceptron) for the case of a single hidden layer.

- The network in Fig. 6.9 is called to as a 10–4–2 network because it has 10 source nodes, 4 hidden neurons, and 2 output neurons.

- As another example, a feed-forward network with *m* source nodes, *h*1 neurons in the first hidden layer, *h*2 neurons in the second hidden layer, and *q* neurons in the output layer is named to as an *m*–*h*1–*h*2–*q* network.

6.1.2. Network Architectures



- Fig. 6.9

6.1.2. Network Architectures

- The network in Fig. 6.9 is said to be ***fully connected*** in the sense that each node in every layer of the network is connected to each other node in the adjacent forward layer.
- If some of the communication links are missing from the network, the network is ***partially connected***.
- This neuron takes as input x1,x2,….,x3 (and a +1 bias term), and outputs f(summed inputs+bias), where f(.) called the activation function.
- The main function of Bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives).
- Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it.
- There are several activation functions you may encounter in practice:

6.1.2. Network Architectures

- **Limitations of Perceptrons:**
  (i) The output values of a perceptron can take on only one of two values (0 or 1) due to the hard-limit transfer function.
- (ii) Perceptrons can only classify linearly separable sets of vectors. If a straight line or a plane can be drawn to separate the input vectors into their correct categories, the input vectors are linearly separable.

- If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly.

6.1.2. Network Architectures

- The Boolean function XOR is not linearly separable (Its positive and negative instances cannot be separated by a line or hyperplane).
- Hence a single layer perceptron can never compute the XOR function.
- This is a big drawback that once resulted in the stagnation of the field of neural networks.
- But this has been solved by multi-layer.

6.1.2. Network Architectures

- **Recurrent Networks**
- A ***recurrent neural network*** distinguishes itself from a feed-forward network in that it has at least one ***feedback*** loop.

- The simplest discrete-time artificial neuron with feedback connection is described by the equation

- $$x_j(k\tau + \tau) = \Psi\left[\sum_{i=1}^{n} w_{ji}\, x_i(k\tau) + \Theta_j\right] \qquad (6.17)$$

6.1.2. Network Architectures

- or
- $$x_j^{(k+1)} = \Psi\left(\sum_{i=1}^{n} w_{ji}\, x_i^{(k)} + \Theta_j\right), \qquad (6.18)$$
- where
- $$x_j(k\tau) = x_j^{(k)}$$
- is the output of the j-th neuron at the discrete-time instant *tk=ktau, wji* are real numbers representing the synaptic weights; psi*j (.)* represents a nonlinear activation function which is commonly the signum function, Gj is the external input (bias); and *n* is the number of neurons.

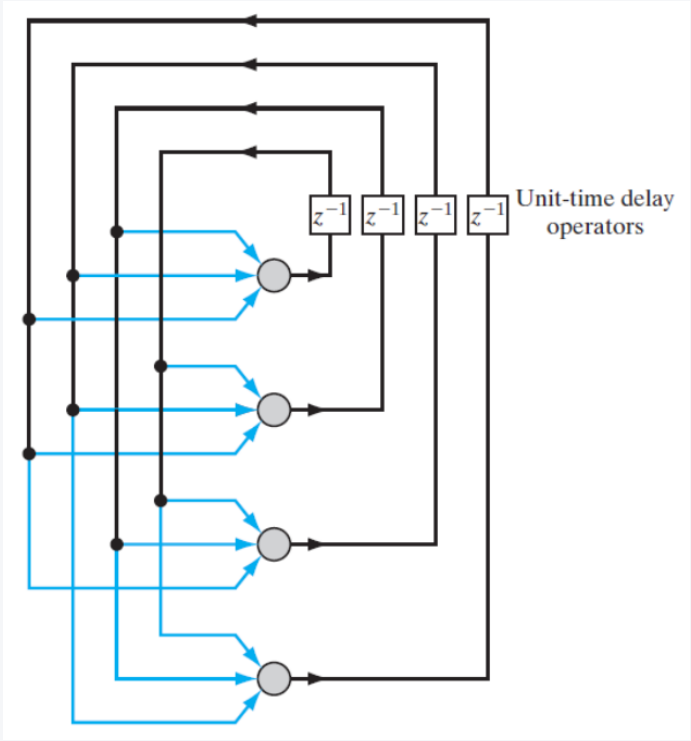
6.1.2. Network Architectures

- A recurrent network may consist of one layer of neurons with every neuron feeding its output signal back to the inputs of all the other neurons, as shown in the graph of Fig. 6.10.

- In the structure shown in this figure, there are *no* self-feedback connections in the network; self-feedback refers to a situation where the output of a neuron is fed back into its own input.

- The recurrent network shown in Fig. 6.10 also has *no* hidden neurons.

6.1.2. Network Architectures



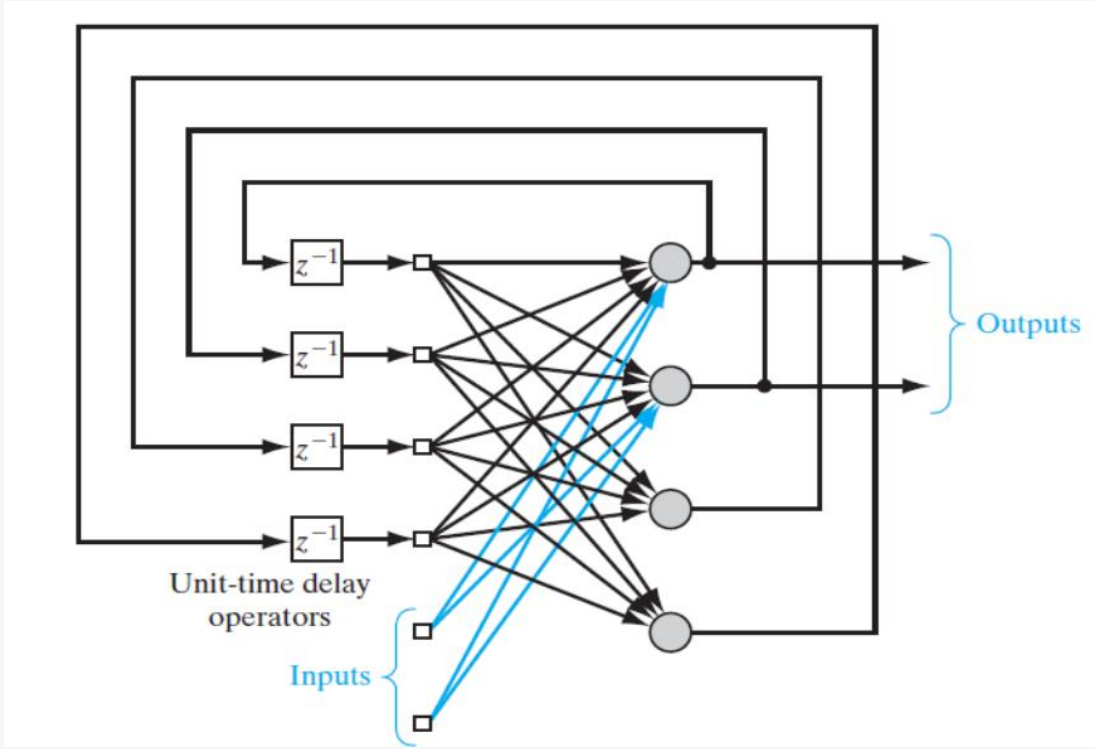
- Fig. 6.10

6.1.2. Network Architectures

- In Fig. 6.11 we show another class of recurrent networks with hidden neurons.
- The feedback connections presented in Fig. 6.11 originate from the hidden neurons as well as from the output neurons.
- The availability of feedback connections, be it in the recurrent structure of Fig. 6.10 or in that of Fig. 6.11, has an impact on the learning capability of the network and on its performance.

- Moreover, the feedback connections involve the use of particular branches composed of unit-time delay elements (denoted by z-1), that result in a nonlinear dynamic behavior, supposing that the network contains nonlinear units.

6.1.2. Network Architectures



- Fig. 6.11

6.1.3. Learning Processes

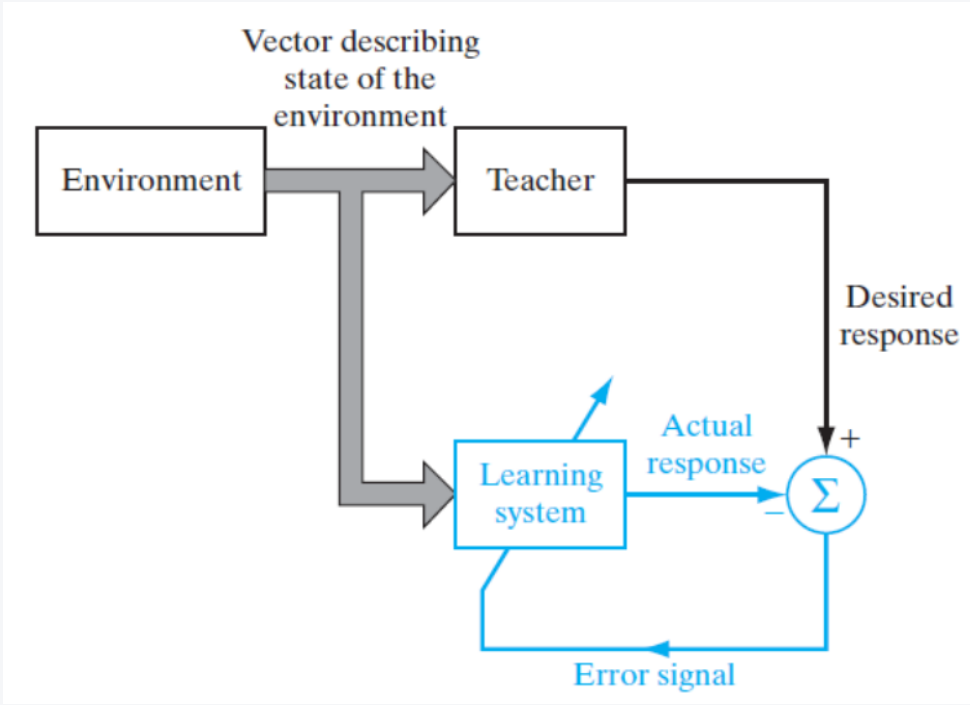
- In a broad sense, we may divide the learning processes through which networks operate as follows: learning with a teacher and learning without a teacher.

- The latter form of learning may be sub-divide into unsupervised learning and reinforcement learning.

6.1.3. Learning Processes

- **Learning with a Teacher**
- ***Learning with a teacher*** is also called as ***supervised learning***.
- Fig. 6.12 shows a diagram that shows this form of learning.
- The environment is ***unknown*** to the neural network.
- Suppose that the teacher and the network are both exposed to a training vector drawn from the same environment.
- The teacher can provide the network with a desired response for that training vector.

6.1.3. Learning Processes



- Fig. 6.12

6.1.3. Learning Processes

- The network parameters are updated under the combined influence of the training vector and the error signal.
- The ***error signal*** is determined as the difference between the desired response and the actual response of the network.
- This adjustment is performed iteratively in a step-by-step mode with the aim of eventually making the network ***emulate*** the teacher.
- Knowledge of the environment available to the teacher is transferred to the network through training and stored in the form of “fixed” synaptic weights, representing ***long-term memory***.
- When this condition is reached, we can then dispense with the teacher and let the network deal with the environment by itself.

6.1.3. Learning Processes

- From Fig. 6.12, we see that the supervised-learning process constitutes a closed-loop feedback system, but the unknown environment is outside the loop.

- As a performance measure for the system, we can think in terms of the ***mean-square error*** over the training sample, defined as a function of the free parameters of the system.

- The true error is ***averaged*** over all possible input–output examples.

6.1.3. Learning Processes

- For the system to improve performance over time and therefore learn from the teacher, the functioning point has to move down successively toward a minimum point of the error surface; the minimum point may be a local or global minimum.

- A supervised learning system is capable to do this with the useful information it has about the ***gradient*** of the error surface corresponding to the behavior of the system.

- The gradient of the error at any point is a vector that points in the direction of ***steepest descent***.

6.1.3. Learning Processes

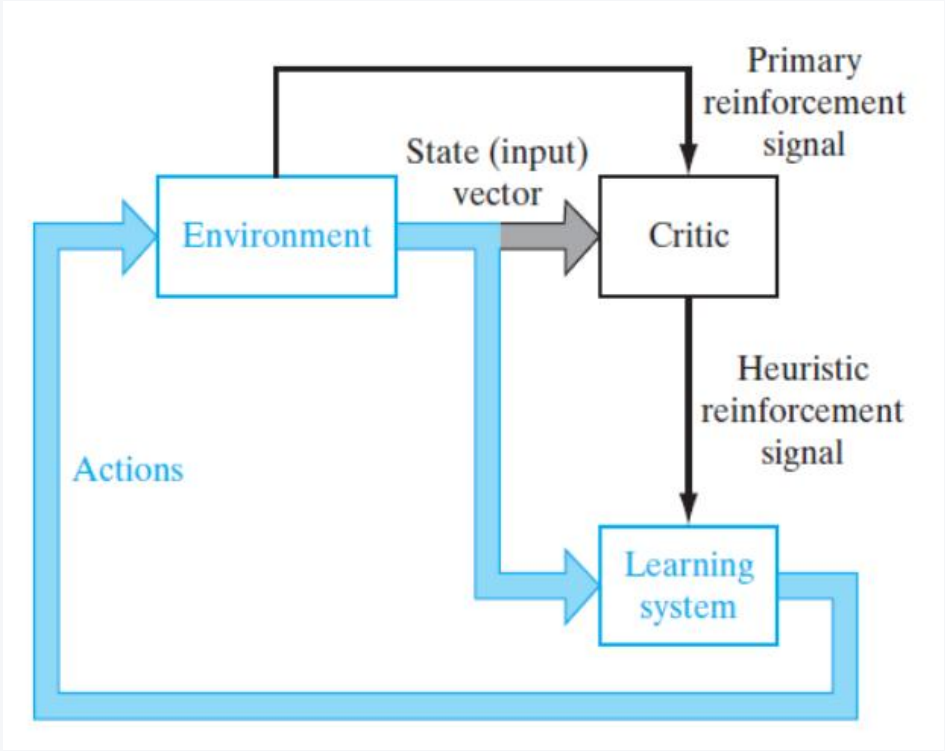
- **Learning without a Teacher**
- In supervised learning, the learning process takes place without a teacher.

- That is, there are no labeled examples of the function to be learned by the network.
- Under this second paradigm, two subcategories are identified:

6.1.3. Learning Processes

- **1. Reinforcement Learning**
- In ***reinforcement learning***, the learning of an input–output mapping is fulfilled through continued interaction with the environment to minimize a performance.
- Fig. 6.13 shows the diagram of one form of a reinforcement-learning system designed around a ***critic*** that converts a ***primary reinforcement signal*** received from the environment into a higher quality reinforcement signal named the ***heuristic reinforcement signal***.
- The system is built to learn under ***delayed reinforcement***, which means that the system observes a sequence of stimuli also received from the environment, which eventually result in the generation of the heuristic reinforcement signal.

6.1.3. Learning Processes



- Fig. 6.13

6.1.3. Learning Processes

- The purpose of reinforcement learning is to minimize a ***cost-to-go function***, determined as the expectation of the cumulative cost of ***actions*** taken over a sequence of steps.

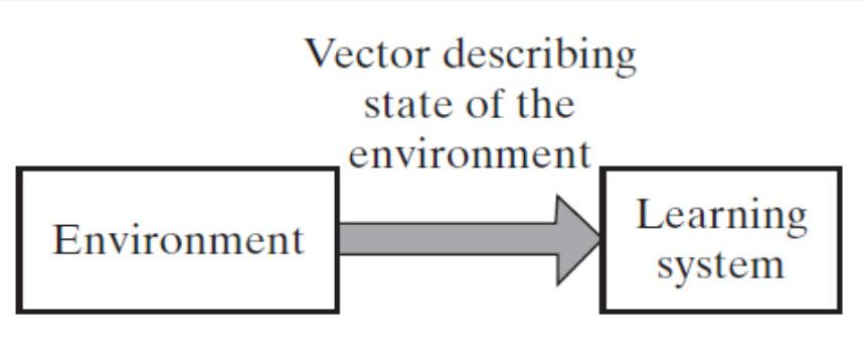
- The goal of the ***learning system*** is to ***discover*** these actions and feed them back to the environment.

6.1.3. Learning Processes

- **2. Unsupervised Learning**
- In ***unsupervised*, or *self-organized*, *learning***, there is no teacher to oversee the learning process, as shown in Fig. 6.14.
- Provision is made for a ***task-independent measure*** of the quality of representation that the network is necessary to train, 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, the network develops the capability to form internal representations for encoding features of the input.

6.1.3. Learning Processes



- Fig. 6.14

6.1.3. Learning Processes

- To fulfill unsupervised learning, we can use a competitive-learning rule.
- For example, we can use a network that consists of two layers—an input layer and a competitive layer.

- The input layer receives the input data.
- The competitive layer consists of neurons that compete with each other to respond to features contained in the input data.
- In its simplest form, the network functions according to a “winner-takes-all” strategy. In such a case, the neuron with the greatest input “wins” the competition and turns on; all the other neurons in the network then turn off.

6.1.3. Learning Processes

- **Pattern Recognition**
- ***Pattern recognition*** is defined as ***the process whereby a received pattern/signal is assigned to one of a prescribed number of classes.***

- A network performs pattern recognition by first undergoing a learning session during which the network is repeatedly presented with a set of inputs along with the category to which each particular pattern belongs.

6.1.3. Learning Processes

- Later, the network is presented with a new pattern that has not been seen before, but which belongs to the same set of patterns used to learn the network.
- The network is capable to identify the class of that particular pattern since the information it has extracted from the learning data.
- Pattern recognition fulfilled by a network is statistical in nature, with the patterns being represented by points in a multidimensional ***decision space***.
- The decision space is divided into ranges, each one of which is connected with a class.
- The decision boundaries are defined by the learning process.

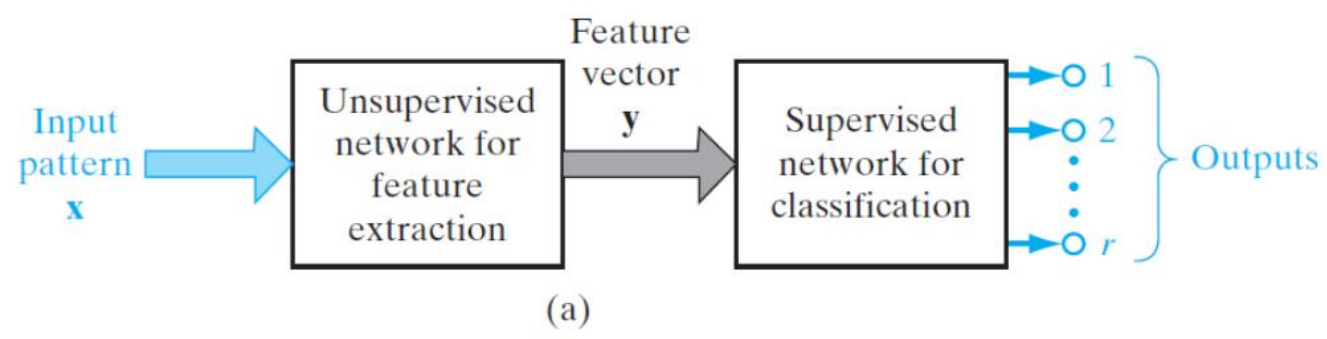
- The designing these boundaries is fulfilled statistically by the inherent variability that exists within and among classes.

6.1.3. Learning Processes

- Pattern-recognition machines using neural networks can take one of two forms:
- • The machine is divided into two parts, an unsupervised network for ***feature extraction*** and a supervised network for ***classification***, as presented in Fig. 6.15a.

- A pattern is represented by a set of *m* observables, which can be viewed as a point **x** in an *m*-dimensional ***observation space***.

6.1.3. Learning Processes



- Fig. 6.15a.

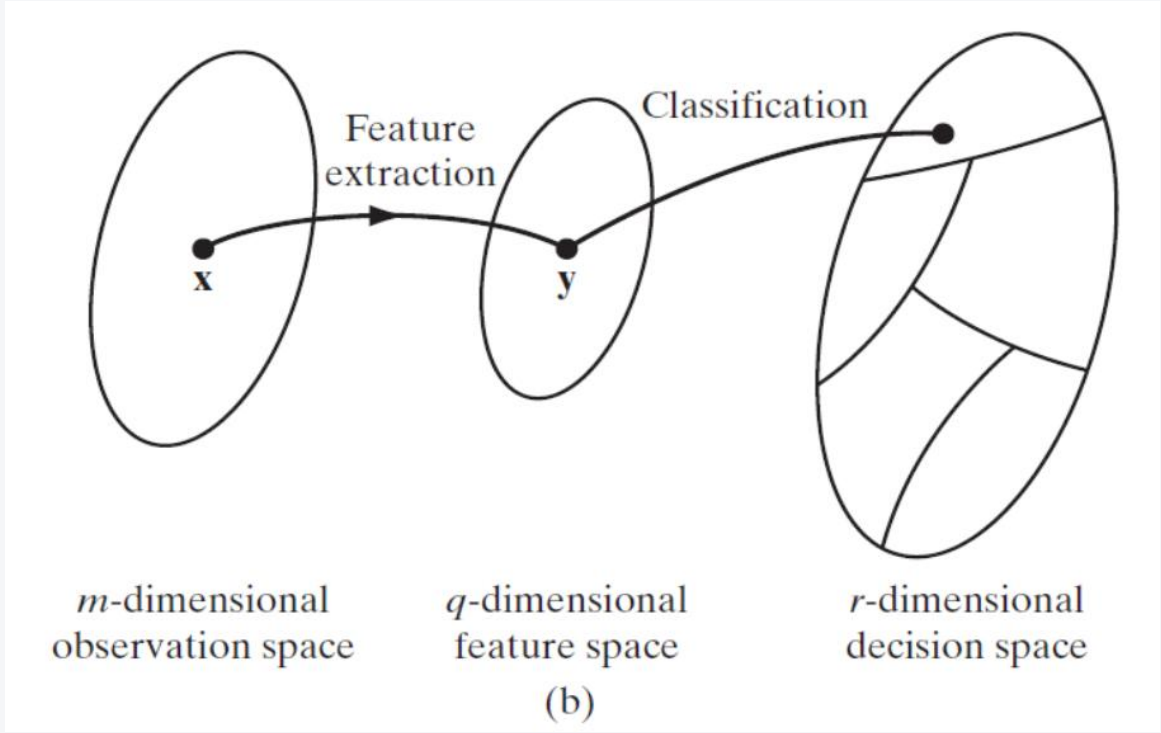
6.1.3. Learning Processes

- Feature extraction is described by a transformation that maps the point **x** into an point **y** in a q-dimensional ***feature space*** with $q < m$, as shown in Fig. 6.15b.

- This transformation can be viewed as one of dimensionality reduction, the use of which is justified on the grounds that it simplifies the classification task.

- The classification is itself described as a transformation that maps the point **y** into one of the classes in an r-dimensional decision space, where r is the quantity of classes.

6.1.3. Learning Processes



- Fig. 6.15b.

6.1.3. Learning Processes

- **Function Approximation**
- Consider a nonlinear input–output mapping given by the functional relationship
- $$\mathbf{d} = \mathbf{f}(\mathbf{x}) \tag{6.19}$$
- where the vector **x** is the input and the vector **d** is the output.
- The vector valued function **f**(·) is supposed to be unknown.
- Because of the lack of knowledge about the function **f**(·), we are given the set of labeled examples:
- $$\mathcal{T} = \{(\mathbf{x}_i, \mathbf{d}_i)\}_{i=1}^{N} \tag{6.40}$$

6.1.3. Learning Processes

- The requirement is to design a network that approximates the unknown function **f**(·) such that the function **F**(·) describing the input–output mapping realized by the network, is close to **f**(·) in a Euclidean sense over all inputs, as given by

- $$\|\mathbf{F}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\| < \varepsilon \quad \text{for all } \mathbf{x} \tag{6.41}$$

- where epsilon is a small positive number.

- Provided that the size *N* of the training sample is large and the network has an adequate quantity of free parameters, the approximation error can be made small enough for the task.

6.1.3. Learning Processes

- The capability of a network to approximate an unknown input–output mapping can be exploited in two important ways:

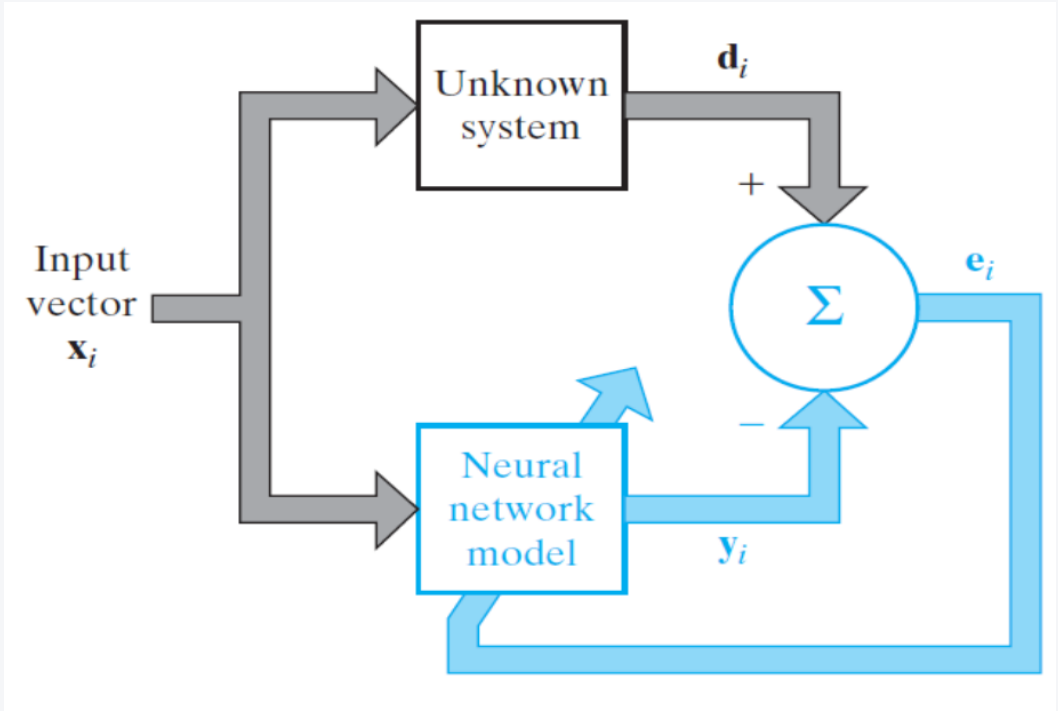
- **(i)** ***System identification.*** Let Eq. (6.19) describe the input–output relation of an unknown memoryless ***multiple input–multiple output (MIMO) system***; by a "memoryless" system, we mean a system that is time invariant.
- We can then use the set of labeled examples in Eq. (6.19) to learn a network as a model of the system.
- Let the vector **y***i* denote the actual output of the network generated in response to an input vector **x***i*.

6.1.3. Learning Processes

- The difference between **d***i* and the network output **y***i* *gives* the error signal vector **e***i*, as shown in Fig. 6.16.

- This error signal is, in turn, used to update the free parameters of the network to reduce the squared difference between the outputs of the unknown system and the network in a statistical sense, and is calculated over the entire learning sample *T*.

6.1.3. Learning Processes



- Fig. 6.16

6.1.3. Learning Processes

- **(ii) *Inverse modeling.*** Assume next we are given a known memoryless MIMO system whose input–output relation is described by Eq. (6.19).
- The requirement in this case is to design an *inverse model* that produces the vector **x** in response to the vector **d**.
- The inverse system may be described by

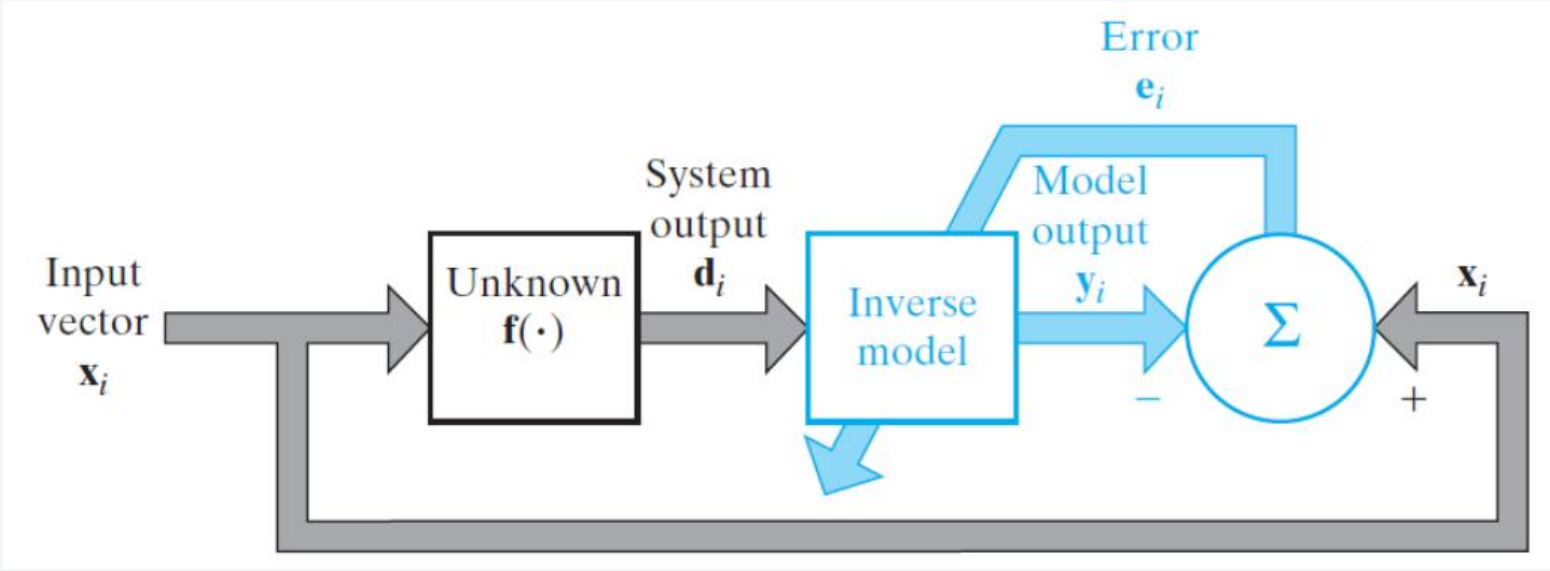
$$\mathbf{x} = \mathbf{f}^{-1}(\mathbf{d}) \tag{6.42}$$

- where the vector-valued function **f**-1(·) marks the inverse of **f**(·).
- Note, that **f**-1(·) is not the reciprocal of **f**(·); rather, the use of superscript -1 is a flag to indicate an inverse.

6.1.3. Learning Processes

- In practice, the vector-valued function **f**(·) is much too complex and inhibits a straightforward formulation of the inverse function **f**-1(·).
- Given the set of labeled examples in Eq. (6.42), we can design a network approximation of **f**-1(·) using the scheme shown in Fig. 6.17.
- In the case described here, the roles of **x***i* and **d***i* are interchanged: The vector **d***i* is used as the input, and **x***i* is treated as the desired output.
- Let the error signal vector **e***i* marks the difference between **x***i* and the actual output **y***i* of the network produced in response to **d***i*.

6.1.3. Learning Processes



- Fig. 6.17

6.1.3. Learning Processes

- This error signal vector is applied to adjust the free parameters of the network to minimize the squared difference between the outputs of the unknown inverse system and the network in a statistical sense and is computed over the complete learning set.

- Inverse modeling is a more difficult training task than system identification, as there may not be a unique solution for it.

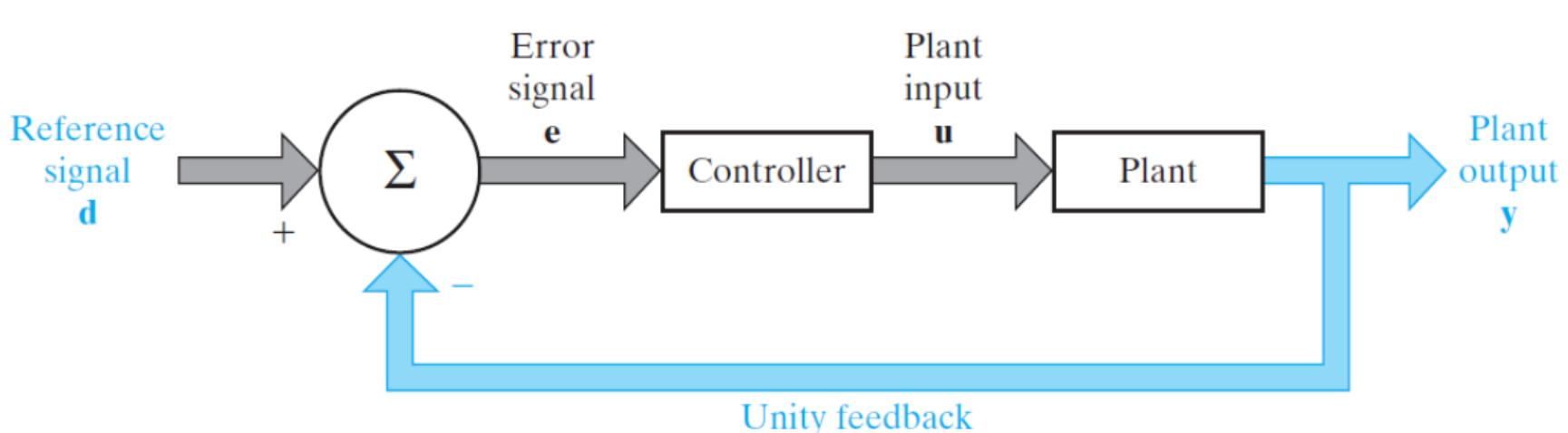
6.1.3. Learning Processes

- **Control**
- The control of a ***plant*** is another learning problem that is well suited for neural networks; by a “plant” we mean a process or critical part of a system that is to be controlled.

- Consider the ***feedback control system*** presented in Fig. 6.18.

- The system uses unity feedback around a plant to be controlled; that is, the plant output is fed back directly to the input.

6.1.3. Learning Processes



- Fig. 6.18

6.1.3. Learning Processes

- Thus, the plant output **y** is subtracted from a ***reference signal*** **d** supplied from an external source.

- The error signal **e** so generated is applied to a neural *controller* for the purpose of updating its free parameters.

- The objective of the controller is to supply appropriate inputs to the plant to make its output **y** track the reference signal **d**.

6.1.3. Learning Processes

- In Fig. 6.18, the error signal **e** has to propagate through the neural controller before reaching the plant.

- Consequently, to fulfill adjustments on the free parameters of the plant according to an error-correction learning algorithm, we need to know the ***Jacobian***, made up of a matrix of partial derivatives as given by

- $$\mathbf{J} = \left\{ \frac{\partial y_k}{\partial u_j} \right\}_{j,k} \qquad (6.43)$$

6.1.3. Learning Processes

- where *yk* is an element of the plant output **y** and *uj* is an element of the plant input **u**.
- Unfortunately, the partial derivatives $\partial y_k / \partial u_j$ for the different *k* and *j* depend on the operating point of the plant and are therefore not known.
- We may use one of two methods to account for them:

- **(i) *Indirect learning.*** Using actual input–output measurements on the plant, we first design a neural model to produce a copy of it.
- This model is, in turn, applied to provide an estimate of the Jacobian **J**.
- The partial derivatives constituting this Jacobian are used in the error-correction learning algorithm for calculating the adjustments to the free parameters of the neural controller.

6.1.3. Learning Processes

- **(ii) *Direct learning***. The signs of the partial derivatives $\partial y_k / \partial u_j$ are generally known and usually remain constant over the dynamic change range of the plant.
- This proposes that we can approximate these partial derivatives by their individual signs.
- Their absolute values are given a distributed representation in the free parameters of the neural controller.
- The neural controller is enabled to train the adjustments to its free parameters directly from the plant.

6.1.4. Simulation Example

- The python code of training and testing single neuron neural network and corresponding output results can be presented as follows:
- *CSCE 5300 – Single Neuron Neural Network*

- # Python program to implement a
- # single neuron neural network
-
- # import all necessary libraries
- from numpy import exp, array, random, dot, tanh
-
- # Class to create a neural
- # network with single neuron
- class NeuralNetwork():

6.1.4. Simulation Example

```
def __init__(self):

    # Using seed to make sure it'll
    # generate same weights in every run
    random.seed(1)

    # 3x1 Weight matrix
    self.weight_matrix = 2 * random.random((3, 1)) - 1

# tanh as activation function
def tanh(self, x):
    return tanh(x)
```

6.1.4. Simulation Example

```
# derivative of tanh function.
    # Needed to calculate the gradients.
    def tanh_derivative(self, x):
        return 1.0 - tanh(x) ** 2

    # forward propagation
    def forward_propagation(self, inputs):
        return self.tanh(dot(inputs, self.weight_matrix))

    # training the neural network.
    def train(self, train_inputs, train_outputs,
                        num_train_iterations):
```

6.1.4. Simulation Example

```
# Number of iterations we want to
    # perform for this set of input.
    for iteration in range(num_train_iterations):
        output = self.forward_propagation(train_inputs)

        # Calculate the error in the output.
        error = train_outputs - output

        # multiply the error by input and then
        # by gradient of tanh function to calculate
        # the adjustment needs to be made in weights
        adjustment = dot(train_inputs.T, error *
                    self.tanh_derivative(output))
```

6.1.4. Simulation Example

```
# Adjust the weight matrix
            self.weight_matrix += adjustment

# Driver Code
if __name__ == "__main__":

    neural_network = NeuralNetwork()

    print ('Random weights at the start of training')
    print (neural_network.weight_matrix)
```

6.1.4. Simulation Example

```
train_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    train_outputs = array([[0, 1, 1, 0]]).T

    neural_network.train(train_inputs, train_outputs, 10000)

    print ('New weights after training')
    print (neural_network.weight_matrix)

    # Test the neural network with a new situation.
    print ("Testing network on new examples ->")
    print (neural_network.forward_propagation(array([1, 0, 0])))
```

6.1.4. Simulation Example

- **Output :**
  Random weights at the start of training
- [[-0.16595599]
- [0.44064899]
- [-0.99977125]]
- New weights after training
- [[6.19428067]
- [0.19482422]
- [0.34317086]]
- Testing network on new examples ->
- [0.99995873]

6.1.5. Concluding Remarks

- **Advantage of Artificial Neural Networks**
- Problem in ANNs can have instances that are represented by many attribute-value pairs.
- ANNs used for problems having the target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- ANN learning methods are quite robust to noise in the training data. The training examples may contain errors, which do not affect the final output.
- It is used generally used where the fast evaluation of the learned target function may be required.
- ANNs can bear long training times depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

6.1.5. Concluding Remarks

- **Limitations of Artificial Neural Networks**
- Neural networks are black boxes, meaning we cannot know how much each independent variable is influencing the dependent variables.

- It is computationally very expensive and time consuming to train with traditional CPUs.

- Neural networks depend a lot on training data.

6.1.5. Concluding Remarks

- **Applications of Neural Networks**
- Facial Recognition.
- Stock Market Prediction.
- Social Media.
- Aerospace.
- Defense.
- Healthcare.
- Signature Verification and Handwriting Analysis
- and many others

6.1.5. Concluding Remarks

- Important property of neural networks stands out that of ***learning***, which is categorized as follows:
- **(i) *supervised learning***, which needs the availability of a target or desired response for the implementation of a specific input–output mapping by minimizing a cost function;
- **(ii) *unsupervised learning***, the realization of which relies on the provision of a task-independent measure of the quality of representation that the network is required to train in a self-organized manner;
- **(iii) *reinforcement learning***, in which input–output mapping is fulfilled through the continued interaction of a learning system with its environment so as to reduce a scalar index of performance.

6.1.5. Concluding Remarks

- Supervised learning relies on the availability of a learning sample of ***labeled examples***, with each example consisting of an input signal (stimulus) and the corresponding desired (target) response.

- •In practice, we find that the collection of labeled examples is a time-consuming and expensive problem, especially when we are dealing with large-scale learning problems; typically, we therefore find that labeled examples are in short supply.

- •On the other hand, unsupervised learning relies on unlabeled examples, consisting simply of a set of input signals or stimuli, for which there is usually a plentiful supply.

6.1.5. Concluding Remarks

- There is a great deal of interest in another category of learning: ***semisupervised learning***, which employs a learning sample that consists of labeled as well as unlabeled examples.

- •The challenge in semi-supervised training, discussed in a subsequent chapter, is to design a training system that scales reasonably well for its implementation to be practically feasible when dealing with large-scale pattern classification problems.

6.1.5. Concluding Remarks

- •Reinforcement learning is between supervised learning and unsupervised learning.
- •It operates through continuing interactions among a learning system (agent) and the environment.

- •The learning system fulfills an action and learns from the response of the environment to that action.

- •In effect, the role of the teacher in supervised learning is replaced by a critic, for example, that is integrated into the training machinery.