

Chapter 1

Introduction

1.1. Background

Human beings are now living in an era of unparalleled changes especially in the world of science and technology. We have witnessed the impact of DNA research from the way of diagnosing cancer, creating new drugs and to the way of tracing our human ancestors back to over eighty thousand years ago. We have also witnessed the birth of the Information Technology era which has changed our life from the personal entertainment habits to our learning and even the way of how our business is run. All these amazing things, which are attributed to the immense development of computing technology, happened in merely few decades. Undoubtedly, computers have become an integral part of our lives that most users have taken them for granted but may not know that the structure of computer is based on John Von Neumann's view on how computational process be organized. Von Neumann architecture is a very organized way of processing computation. The computing is achieved by using a CPU operating a series of instructions including: fetch, decode, execute, and writeback. This type of serial operation has been working extremely well for virtually all applications, until the challenge posed by the emerging Artificial Intelligent technology.

Neural Computing is basically a parallel distributed processing. It has the ability of performing supervised and/or unsupervised learning to adapt the information environment. The architecture of neural network is in fact based on the way of how our human nerve system is connected. Generally, there are about 100 billion numbers of neuron in a human brain. Neurons in our brain are parallel connected to numerous other neurons forming a massive parallel computer like machine. Neural network is designed in a way to seek the style of computing of human

brain. As a result, neural network is powerful enough to solve a variety of problems that are proved to be difficult with conventional digital computational methods. Typical cognitive tasks include recognition of a familiar face, learning to speak and understand a natural language, retrieving contextually appropriate information from memory, and performing demanding classification tasks.

The human thinking system is in parallel which means it operates with numerous of our neurons connected together. In contrast to the conventional crisp mathematical logic, the main characteristics of human thinking process is imprecise, fuzziness, but adaptive. It learns by examples, experience and it exhibits strong adaptation to external changes. Neural networks are designed in a way to mimic most of these characteristics. They are so far exhibiting very encouraging performance.

Learning: Neural network can modify their behaviour in response to the environment. When a given set of inputs with or without desired outputs, they can self-adjust to produce consistent responses.

Generalization: When the network is once trained, its response can be in some extent insensitive to minor variations, which may be caused by noise corruption or slight distortion in a real-world environment, in its inputs.

Massively parallelism: Information is processed in a massive parallel manner.

Fault-tolerance: Once when the network connections are made, the network is able to deliver a robust behaviour. The response of the network as a whole may only be slightly degraded if some of its processing elements are slightly damaged or altered.

1.2. Neuron Model

Neuron is the fundamental cellular unit of a nervous system. A typical biological neuron in human brain is shown in Fig. 1.1. In each neuron it has an output fiber called axon, and a button like terminal called synapse. The axon, which is the output path of a neuron, splits up and connects to many dendrites, which are the input paths of other neurons, through a junction called synapse. Each neuron receives and combines signals from numerous neurons through dendrites similarly connected.

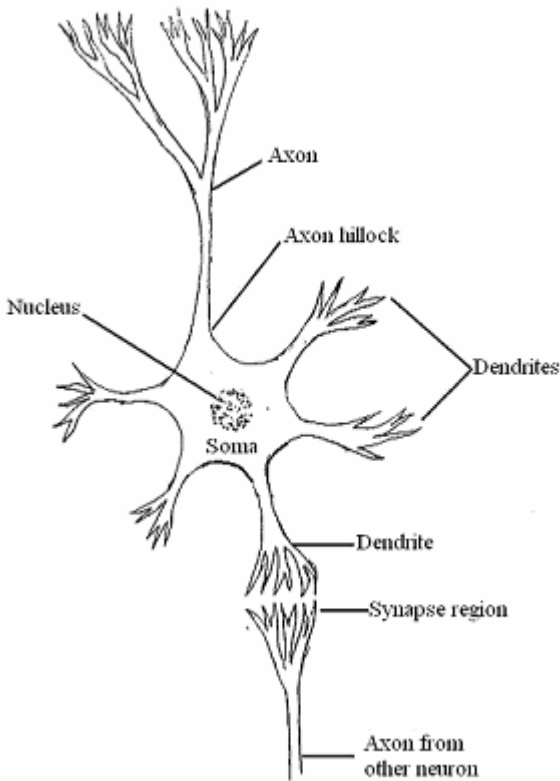


Figure 1.1. A typical biological neuron and its connection

A neuron can receive up to few thousands to about fifteen thousands inputs from the axons of other neurons. Apparently this forms a massive parallel system compared with the digital computer architecture. In each neuron, if the combined signal exceeds a threshold value, it activates the firing of a neuron producing an output signal through the axon. Transmission of information across synapses is in fact a chemical in nature, but it has electrical side effects which we can be measured. Electrical activities in neurons appeared in a shape of pulse with a frequency in the range of 1 KHz. This type of biological behaviour is modeled by an electronic model shown in Fig. 1.2. A simple neuron model is the most fundamental processing element of neural networks. The weights correspond to the synaptic strength of neural connections, i.e., analogous to “memory” and the neuron sums all the weighted inputs,

modifies the signals through a transfer function which is usually non-linear. The transfer function can be a threshold function which only allows signal to be passed if the combined activity level reaches a certain threshold level, or it is a continuous function of the combined input.

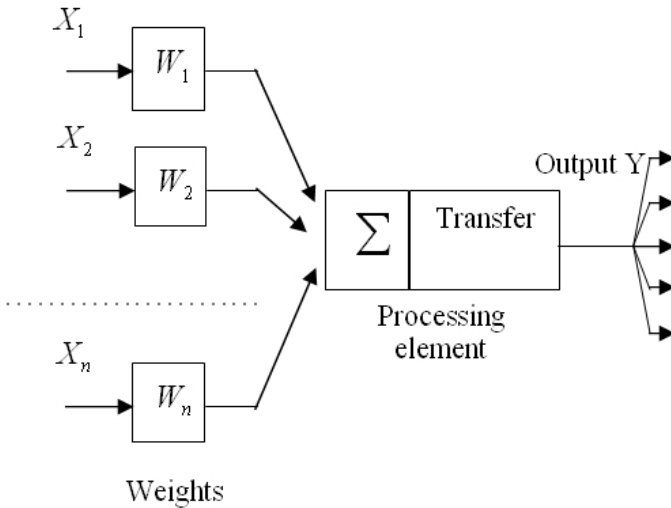


Figure 1.2. Electronic model of a neuron

1.3. Historical Remarks

The first neural network model has to be the McCulloch and Pitts model proposed in 1943. It is a very simple electronic model that can be hardware implemented. It is a multiple inputs summing device that consists of different weightings for each input, and a threshold before the output. The significance of this model at that time was its ability to compute any arithmetic or logical function. It is until about the ends of the fifties that the first type of Perceptron was proposed by Frank Rosenblatt, and Wightman. The first application of the Mark I Perceptron machine was on pattern recognition. In their experiments, the pattern recognition ability of Perceptron model was demonstrated by recognizing different simple characters. Subsequently, numerous neural networks results were reported. Neural hardware business was even established. It was similar to the hype of the IT industries in the end of nineties, the neural hype burst when good idea ran out. By the mid of

sixties, Minsky and Papert's paper mathematically proved that a simple perceptron could not handle the XOR function. This has silenced neural research work for about 2 decades. It was until about the mid of eighties that Neural research has become flourish again. It was achieved with the contribution from a number of renowned scientists including Hopfield, Amari, Grossberg, Kohonen, Fukushima, Oji, etc. They have developed many important neural topologies.

In 1982, Hopfield published two important articles which most people still regard as the inauguration of the current neural network era. Hopfield showed a novel idea that models of physical systems could be effectively used for solving computational problems. Using the idea of an energy function to establish a new type of network architecture, he showed that when a distorted pattern is presented to the network, the pattern is associated with another pattern which belongs to a similar class pattern. Hopfield networks are, thus, sometimes called associative networks. For a discrete-time Hopfield network, the energy of a certain vector with a given initial state will converge to a value having minimum energy. This is used to explain its capability of converging to a similar class of patterns. Also, in the early eighties Carpenter and Grossberg established the well-known adaptive resonance theory (ART) based on their early work on competitive learning. ART was introduced by them over the period of 1976-1986 as a theory of human information processing. Like Kohonen's Self Organising Map, they were working on systems that are capable of organizing themselves. ART has the ability to learn without supervised training and is consistent with cognitive and behavioral models. It was derived based on competitive learning which is capable of finding categories. ART has been widely used as a type of neural network models that perform supervised and unsupervised category learning, and pattern classification.

In 1988 George Cybenko published a very important work proving the universal functional approximation ability of neural networks. In 1989, Funahashi, Hornik, and Stinchcombe also reported their findings on proving multilayer Perceptron network as a universal approximator. Subsequently, neural networks have been widely applied on many different science and engineering areas. The nowadays neural networks have already extended from its simple pattern recognition problem to the very complicated DNA, gene recognition and classification problems. Applications have extended from physical science and engineering to finance, economic and social science.

1.4. Network architecture

Generally, neural networks can be categorized into 2 main types, namely supervised networks and unsupervised networks. The way the network architecture was designed has taken the ability of its training algorithm into account. In most newly proposed network topologies, the design of their corresponding training algorithms are deemed essential. Apparently, a successful network architecture must be supported by an effective and simple enough training algorithm. In this book, the later chapter will detail different efficient training algorithms. In this section, we focus the introduction of their hardware architectures.

1.4.1. Supervised Neural Networks

Supervised neural networks are the mainstream of neural network development. The differentiable characteristic of supervised neural networks lies in the inclusion of a teacher in their learning process. The basic block diagram of supervised learning for all neural network models can be described through Fig. 1.3. For the learning process, the network needs training data examples consisting of a number of input-output pairs. The desired output vector in the training data set serve as a teacher for the network's learning. In the training process the error signals are constructed from the difference between the desired output and system output. Through iterative training procedure the network's weights are adjusted by the error signal in a way that the network output tries to follow the desired output as close as possible.

The learning procedure continues until the error signal is close to zero or below a predefined value. The sum of errors over all the training samples can be considered as a kind of network performance measure, which is a function of free parameters of the system. Such function can be visualized a multidimensional error surface where network free parameters serves as coordinates. During the course of learning the system gradually moves to a minimum point along an error surface. The error surface is determined by the network architecture and the cost function. In later chapter of this book, more details in this aspect will be discussed. The supervised networks' architectures can vary depending on the complexity and nature of data to be handled. Broadly speaking, they can be sub-divided into following three fundamental classes.

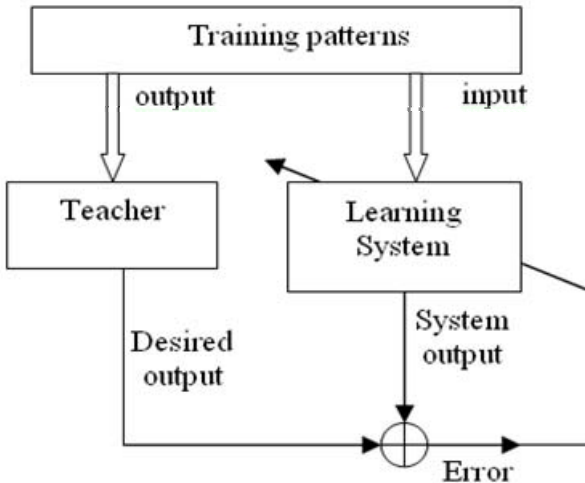


Figure 1.3. Overview of supervised learning

1.4.1.1. McCulloch and Pitts Model

In 1943, neurophysiologist McCulloch and his associate Pitts used a type of electrical circuit to model a simple neuron. This is the first ever type of electronic model used to model a neuron. The McCulloch and Pitts (MCP) model has a profound impact on the later Perceptron model. The MCP model basically consists of a summing amplifier and variable resistors as shown in Fig. 1.4. The weights (W_1, W_2) in Fig. 1.4 are adjusted through varying the values of the resistors. A threshold device, which is made of voltage comparator, generates an output 1 if the summation of signals exceeds the threshold value, T , or the output is 0 if the signal is less than the threshold value, T .

To illustrate how the MCP model works, we use a simple “AND” logic problem as example. The general form of MCP model is

$$\text{Output} = 1, \text{ if } X_1W_1 + X_2W_2 > T \quad (1.1)$$

We make use of the truth table and the following inequalities are obtained.

$$0 < T, \quad 0 + W_2 < T, \quad W_1 + 0 < T, \quad W_1 + W_2 > T$$

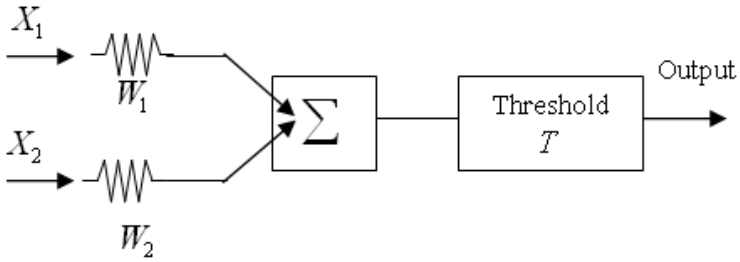


Figure 1.4. The general form of MCP model

Table 1.1

X_1	X_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

The task is to determine the values of T , W_1 and W_2 so that the output satisfies the logic “AND” function. We can choose $W_1 = 0.5$, $W_2 = 0.8$ and $T = 1$ as a solution. Fig. 1.5 shows that the job is equivalent of finding a line separating the 3 “0” from the “1”. Apparently, the 3 lines and many other lines can satisfy the requirement.

To find the separating line, we need to find the gradient and the intercepts of the line. By replacing “>” with “=”,

$$W_1 X_1 + W_2 X_2 = T \tag{1.2}$$

$$X_2 = \frac{T}{W_2} - \frac{W_1 X_1}{W_2}$$

Thus $a = \frac{1}{0.8} = 1.25$, and $gradient = \frac{-5}{8}$, $b = \frac{1}{0.5} = 2$

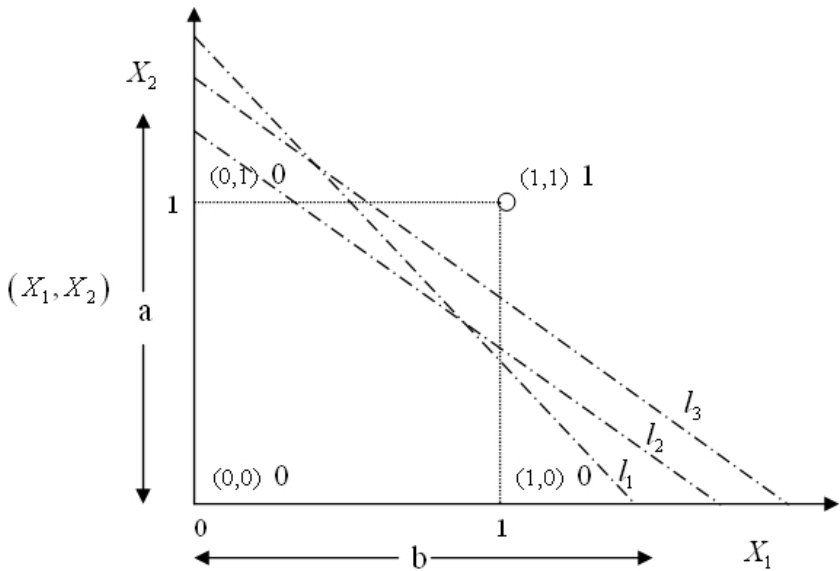


Figure 1.5.

There are infinite numbers of solutions that can satisfy the requirement. Clearly, it is a very loose way in finding a solution which appears to be one of the beauty of neural network. The Application of a “AND” logic function may appear to be too simple to illustrate its concept of finding a hyper-plane for discrimination. We now consider the following logic function.

Table 1.2

X_1	X_2	X_3	Output
0	0	0	1
0	1	1	0
0	0	1	0
0	1	0	1
1	0	0	0
1	1	1	0
1	0	1	0
1	1	0	0

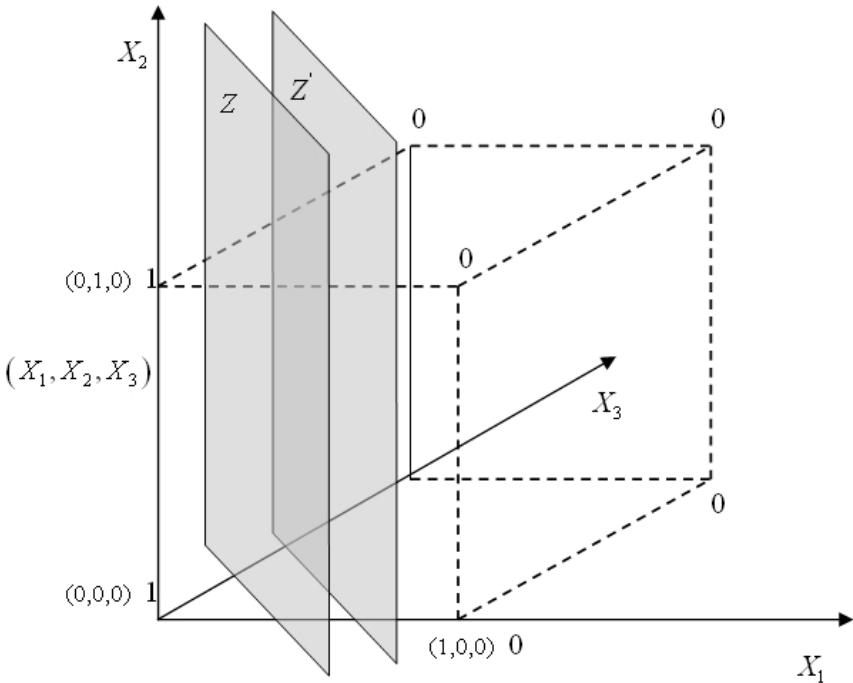


Figure 1.6.

In this example, we need to find a plane “Z” to separate the 6 “0” from the 2 “1”. Similarly, it is clear that there are many hyperplanes “Z” can do the separation job. The complexity of the problem can be visualized when higher dimension input vector are considered. A training rule is needed to find the possible hyperplane to separate “1” from the “0”.

In the above examples, whether it is a 3-dimensional cube or a 2-dimensional plane, they are all linearly separable cases. This means that we can use a linear line or a hyperplane to separate the “0” from the “1”. In real world, problems may usually be linear non-separable. We use the famous “ex-OR” as example to illustrate the concept of linear non-separable. Fig. 1.7 shows that it is impossible to use a straight line to separate the “0” from the “1”. This brings out the main shortcoming of a simple Perceptron model and the need of using multilayer Perceptron network which will all be briefed in later sections of this chapter.

Table 1.3

X_1	X_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

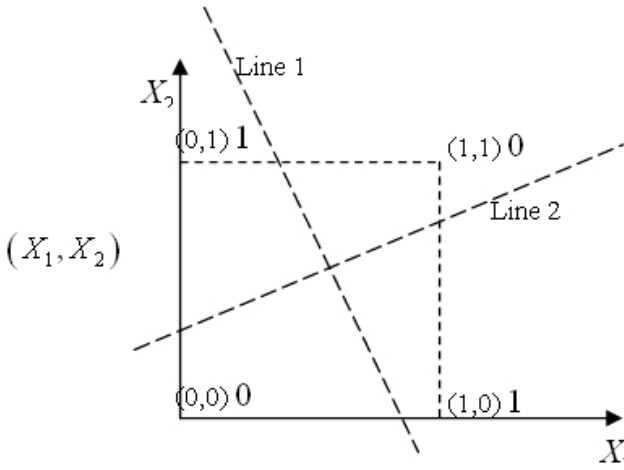


Figure 1.7.

In this case none of line “1”, line “2” and all other linear lines can possible separate the 2 “0” from the 2 “1”.

1.4.1.2. The Perceptron Model

The Perceptron model is the simplest type of neural network developed by Frank Rosenblatt in 1962. This type of simple network is rarely used now but it is significant in terms of its historical contribution to neural networks. A very simple form of Perceptron model is shown in Fig. 1.8. It is very much similar to the MCP model discussed in the last section. It has more than 1 inputs connected to the node summing the linear combination of the inputs connected to the node. Also, the resulting sum goes through a hard limiter which produces an output of +1 if the input of the hard limiter is positive. Similarly, it produces an

output of -1 if the input is negative. It was first developed to classify a set of externally inputs into 2 classes of C_1 or C_2 with an output +1 signifies C_1 or C_2 . Despite its early success, the single layer Perceptron network was proved by Minsky and Papert’s work that it is unable to classify linear non-separable problem.

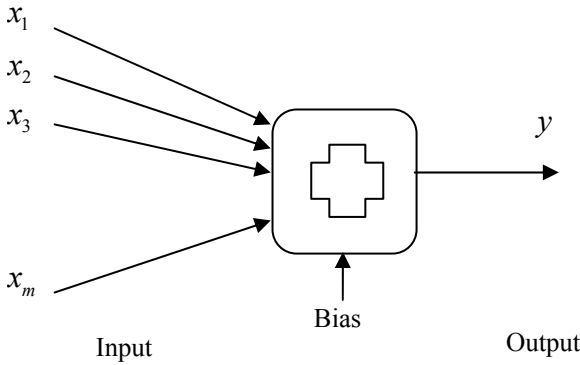


Figure 1.8. Single-layer Perceptron model

A 2-layer Perceptron network can be used for classifying linear non-separable problems. First, we consider the classification regions in a 2 dimensional space. Fig. 1.9 shows the 2-D input space in which a line is used to separate the 2 classes C_1 and C_2 .

Similar to Eq. (1.2)

$$W_1X_1 + W_2X_2 - T = 0$$

The region below or on the right of the line is

$$W_1X_1 + W_2X_2 - T > 0 \tag{1.3}$$

Thus the region above or on the left of the line is

$$W_1X_1 + W_2X_2 - T < 0 \tag{1.4}$$

Fig. 1.10 shows that a linear non-separable case can be separated using a 2-layer perceptron model. Fig. 1.10(b) shows a 2-layer perceptron network separating linear non-separable case. This concept has paved the way for the later development of multi-layer feedforward model.

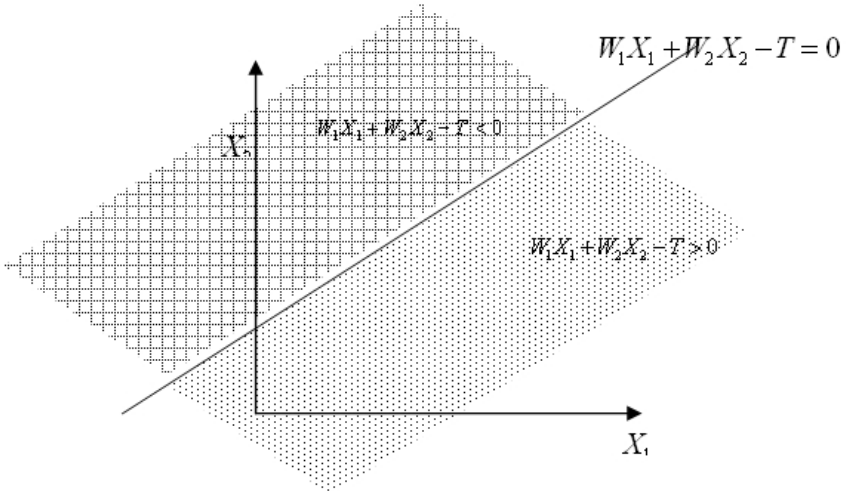


Figure 1.9.

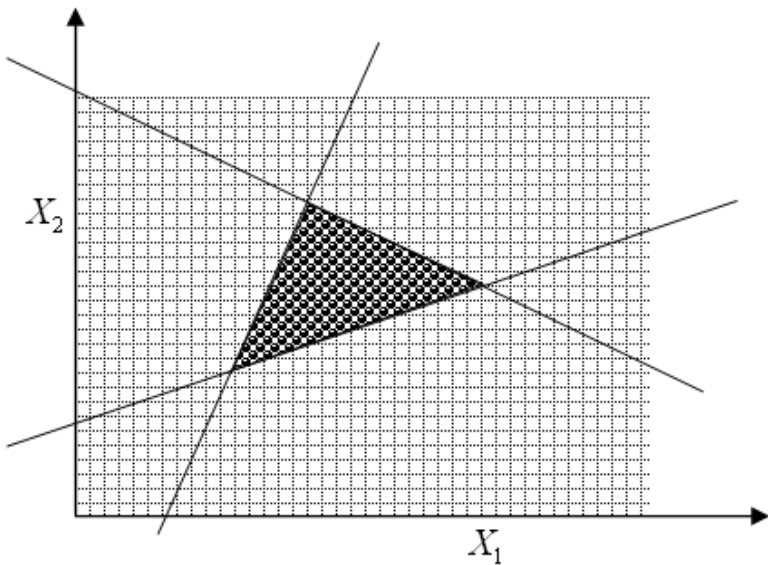


Figure 1.10. (a) It shows we can use 3-linear lines to separate linear non-separable case

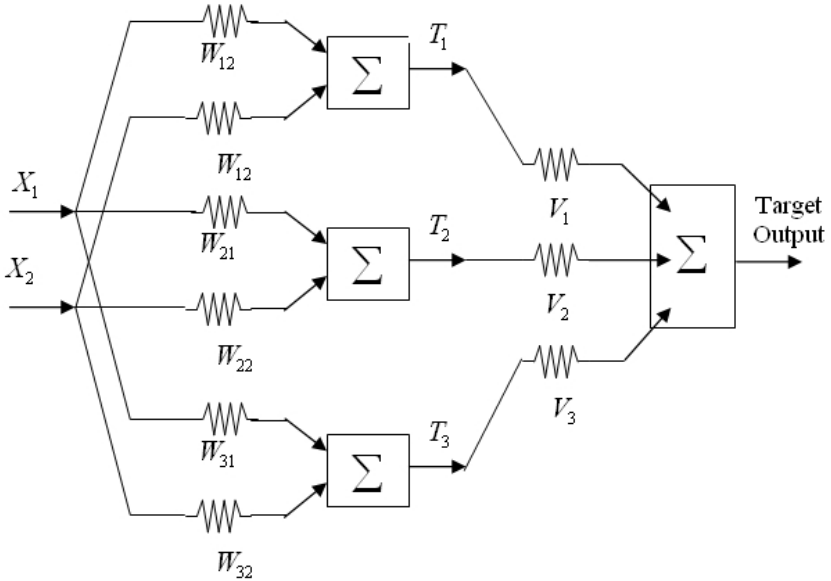


Figure 1.10. (b) A 2-layer perceptron network separating linear non-separable case

1.4.1.3. Multi-layer Feedforward Network

In a simple form, a feedforward network consists of an input layer and a single layer of neurons. Such a single-layer feedforward network is not capable of classifying nonlinearly separable patterns as discussed in the previous sections. Multi-layer feedforward network has become the major and most widely used supervised learning neural network architecture. In the feedforward networks, all connections are acyclic or unidirectional from input to output layer.

Multi-layer network, shown in Fig. 1.11, consists of one or more layers of neuron, called hidden layer, between input and output layer. The neurons in hidden layers are called hidden neuron. The network is called fully connected when every neuron in one layer is connected to every neuron of the next layer. It is able to handle relatively complex tasks and linear non-separable classification. A typical example of such a problem is the well-known Exclusive OR (XOR).

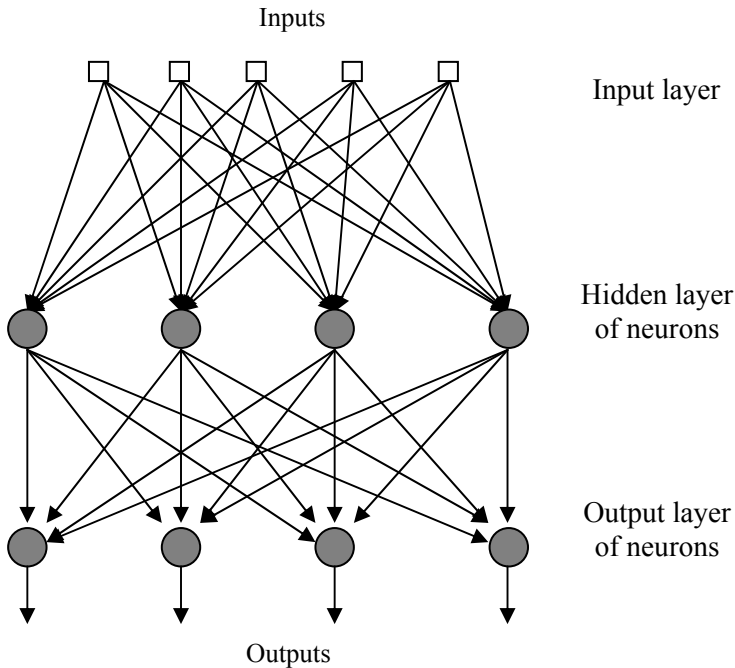


Figure 1.11. Multilayer feedforward network with 1 hidden layer

1.4.1.4. Recurrent Networks

Recurrent network is a special form of network. It can be a single layer or multiple hidden layers network. The basic difference from feedforward networks is that they have one or more feedback loops as shown in Fig. 1.12. The feedback loop can appear in many forms between any two neurons or layers. It typically involves unit delay element denoted by z^{-1} . Recurrent networks exhibit complex dynamics because they consist of a large number of feedforward and feedback connections. This characteristic provides them extra advantages in handling time-series related and dynamical problems over feedforward networks. Recurrent networks are also useful for processing special types of data such as graph-structured data. A recurrent network with a smaller network size may be equivalent to a complicated type of feedforward network architecture.

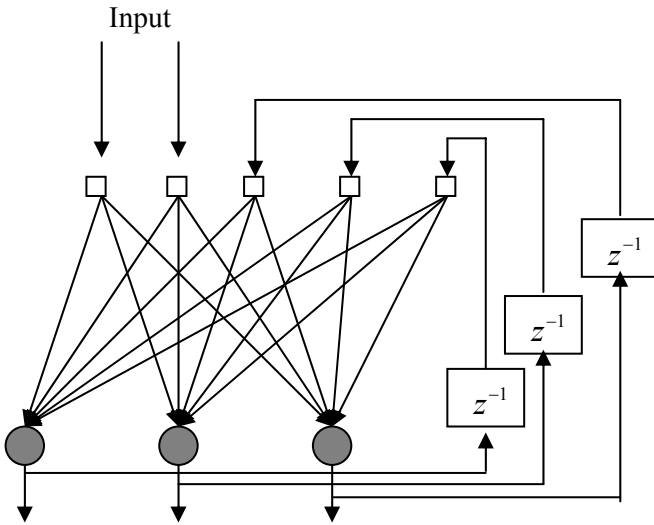


Figure 1.12. Recurrent network

There have been a wide applications of recurrent networks on intelligent control, system identification, and dynamical system applications. In these applications, theoretical study of stability, convergence of the network trajectory to the equilibrium, and their functional approximation ability are regarded important. These studies have widely been reported in many research articles. Some of these work include the proof of a discrete-time trajectory on a closed finite interval can be represented by using a discrete-time recurrent neural network. In the case of continuous-time recurrent networks, there is work showing that a continuous-time dynamical system without input i.e. $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})$ can be approximated by a class of recurrent networks to an arbitrary degree of accuracy. Subsequently, there are work proving that a general dynamical continuous-time systems with control input, i.e. $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{u})$ can be represented by recurrent networks. Different type of dynamical system like $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x}) \cdot \mathbf{u}$ has also been studied. All these works focus on the approximation problem of continuous-time recurrent networks to dynamical time-invariant system $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{u})$. There are also study on dynamical time-variant systems,

i.e. $\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}, \mathbf{u}, t)$, which has wider industrial applications because most dynamical systems may be time-variant. Now the theoretical study of recurrent neural networks on approximating different dynamical systems has reached a very sophisticated stage that recurrent networks have been widely applied to many forecasting, functional approximation, system identification, and control problems. In recurrent network learning algorithm, special types of algorithms were designed to handle control problems. More approaches have been developed using recurrent neural networks based control methods. It has been pointed out that the fundamental shortcomings of current adaptive control techniques, such as nonlinear control laws which are difficult to derive, geometrically increasing complexity with the number of unknown parameters, and general unsuitability for real-time applications, have compelled researchers to look for solutions elsewhere. Recently, it has been shown that recurrent neural networks have emerged as a successful tool in the field of dynamical control systems. Funahashi and Nakamura have proved that any finite-time trajectory of a given n -dimensional dynamical system can be approximately realized by internal states of the output units of a continuous-time recurrent neural network when appropriate network topologies together with appropriate initial conditions are used. When recurrent networks are used to approximate and control an unknown nonlinear system through an on-line learning process, they may be considered as subsystems of an adaptive control system. The weights of the networks need to be updated using a dynamical learning algorithm during the control process. In establishing a convergence control action, many different recurrent training learning algorithms were developed, for instance Chow and Fang derived a 2-D nonlinear system mathematical model to describe the dynamic of a recurrent network and the dynamics of the learning process. The error dynamic equation is expressed in the form of a 2-D Rössler's model with time-varying coefficients. Based on 2-D system theory, a real-time iterative learning algorithm for trajectory tracking was derived.

1.4.2. Unsupervised Neural Networks

Unlike the supervised networks, unsupervised networks do not have a teacher in the training data set. The learning process of unsupervised neural networks is carried out from a self-organizing behavior. In the

course of training, no external factor is used to affect the weights adjustment of the network. The correct outputs are not available during the course of training. For instance, a typical unsupervised network consists of an input layer and a competitive layer. Neurons on the competitive layer compete with each other via a simple competitive learning rule to best represent a given input pattern. Through competitive learning, the network output automatically reflects some statistical characteristics of input data such as data cluster, topological ordering etc.

Self-organizing map (SOM) is the most widely used unsupervised neural networks. Fig. 1.13(b) shows the architecture of a typical SOM network, in which all neurons, arranged on a fixed grid of the output layer, contain a weight vector similar to the input dimension. After the training, each neuron becomes representative of different types of input data. One of the most important characteristic of SOM lies in its topological ordering which means that the neurons that have similar weight (in the input dimension) also close to each other in the SOM output map. This type of SOM map is useful in a many applications including clustering, visualization, quantization and retrieval.

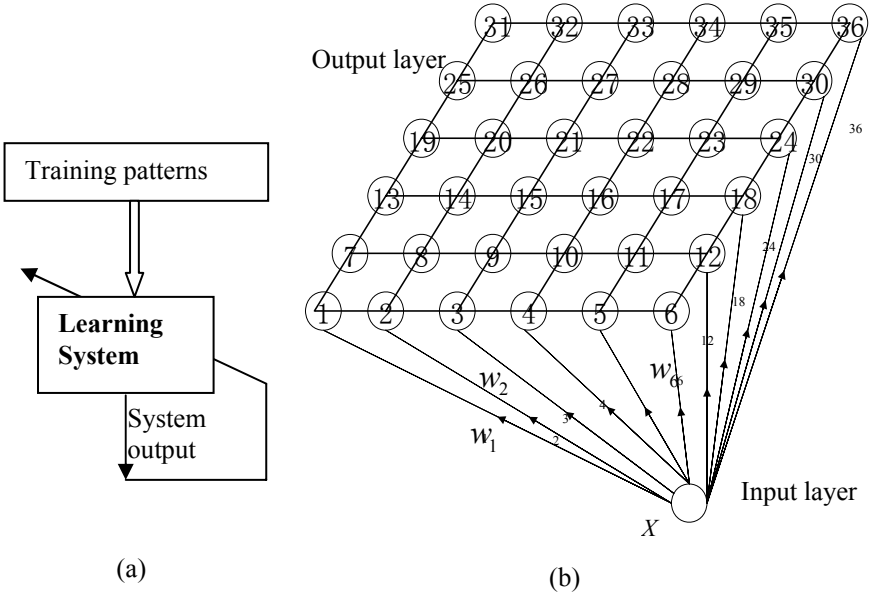


Figure 1.13. (a) Unsupervised learning process (b) SOM network architecture

1.5. Modeling and Learning Mechanism

One of the main functions of neural networks is about their excellent ability to model a complex multi-input multi-output system. Neural networks have widely been considered and used as a kind of soft mathematical modeling method. In a given high dimensional input output dataset, neural networks are able to provide a promising modeling service. Despite the fact that the neural learning procedures may be perceived rather straightforward from the perspective of application, every problem is vastly different because the problems usually consist of new measurements or test data that are not exactly the same as the training data seen amid the training period. For instance, consider a face detection problem in which the training dataset is given with a set of pictures containing faces. A picture can be represented by a vector of its pixel values in an n -dimensional vector. The training set may consist of an m pairs of data in which the label, which is the name of the face, associated with the picture is included. The training process should come up with a function $h(x)$ which would deliver the correct label when given a query image. Usually, there are 2 main problems: (i) the function, $h(x)$, may produce excellent results on the training set, but may perform rather poorly on new unseen pictures. (ii) New unseen pictures may never be identical as the previously seen images in the training set. Apparently, the task of training is not simply to reduce the training error. We are more concern on the generalization ability of the network.

There are many training algorithms developed in the past 10 years, they all share the similar goal of minimizing the training error in the shortest possible time. Despite their employed approaches may ranged from using gradient descent optimization to Least Squared mechanism, we need to analyze the problem from the perspectives of Information theory, and Statistics in order to come up with the best network for a given problem. In order to understand the fundamental of learning mechanism, we will start to illustrate it from a simple linear regression perspective.

1.5.1. Determination of Parameters

We can consider a simple linear regression example in which a linear line is used to model a given dataset as shown in Fig. 1.14. It investigates the electrical property of a new conducting material. The

dots show how electrical current varies with the supply voltage. For a given supply voltage of 50 volts, we are asked to predict the magnitude of current flowing through the new material. This question can simply be answered by using a linear model. The simplest model is in the form

$$y = wx + b \tag{1.5}$$

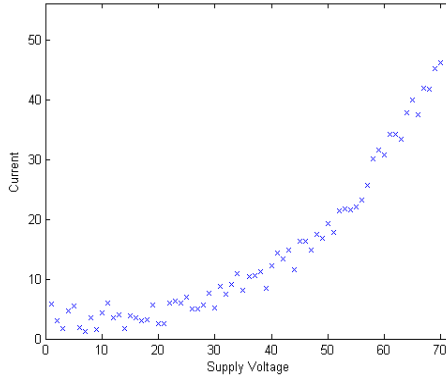


Figure 1.14.

The question becomes the determination of the 2 parameters, w and b . It is a question of finding the best straight line, which will be used as a predictor, drawn through the experimental data. In this simple linear regression question, the two parameters can be solved conventionally by using the Least Squared method.

$$w = \frac{\sum_i x_i d_i - (\sum_i x_i) (\sum_i d_i) / N}{\sum_i x_i^2 - (\sum_i x_i)^2 / N} \tag{1.6a}$$

$$b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i X_i \sum_i x_i d_i}{N \sum_i x_i^2 - (\sum_i x_i)^2} \tag{1.6b}$$

where d_i, x_i correspond to the measured current response, and voltage supply respectively, and N is the total number of data. In solving the linear regression problem, the 2 parameters are obtained by using the

above equation. One can imagine that complexity of the problem when more parameters are involved. In neural network training mechanism, we have to rely on a more sophisticated optimization approach on solving the numerous parameters.

We use the same linear regression problem to illustrate the concept of a cost function approach. To make a “good predictor”, we define a cost function E over the model parameters. One of the most widely used cost function for E is the sum squared error given

$$E = \frac{1}{2} \sum_i (d_i - y_i)^2 \quad (1.7)$$

The above equation is quadratic because of the square term. The performance surface is in a form of a convex surface, in which the minimum locates at the bottom. Fig. 1.15 shows how the sum squared error varies with the 2 parameters.

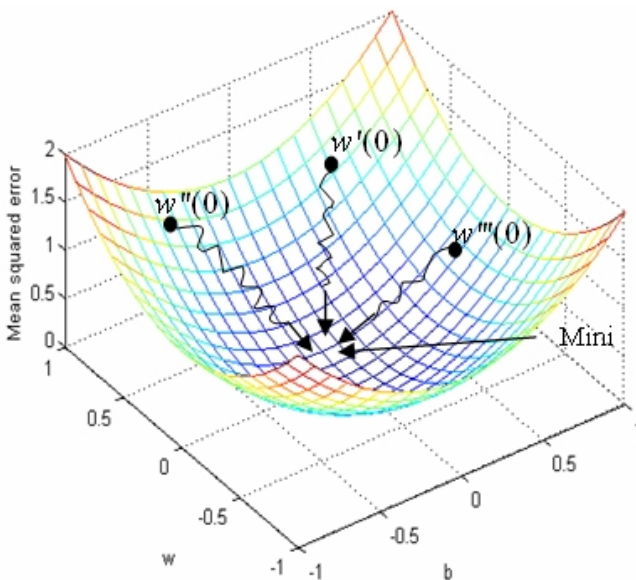


Figure 1.15.

The cost function E provides us an objective measure of predictive error for a specific choice of parameters. The best possible parameters

can be obtained by minimizing the cost function or finding the minimum of the performance surface. We use gradient descent type optimization algorithm to find the minimum. This can easily be found as long as we are able to evaluate the gradient of the performance surface and follow its downhill path all the way along the surface. Different initial settings do not affect the final solution except providing different downhill paths. We can simply imagine that we are somewhere on or near the summit of a mountain. The shortest way to go downhill will simply be to follow the steepest gradient of the terrain. But the selection of our step size is important when we go downhill because too large a step size will apparently be dangerous. In mathematical sense, too large a step size will likely cause oscillation around the minimum although it may speed up the optimization at the beginning. All these issues will be discussed in detail in later chapter of this book. The gradient descent optimization procedures are

1. Initialize the parameters randomly.
2. Evaluate the gradient of the error function with respect to each model parameter.
3. Adjust the model parameters by a certain step size in the direction of the steepest gradient.
4. Repeat steps 2 and 3 until the minimum is found.

Interestingly, the linear model shown in Eq. (1.5) can be implemented by the simplest form of a neural network shown Fig. 1.16. This simple network consists of a bias neuron, an input neuron, and a linear output neuron. In most feedforward neural network implementation, the bias neuron is set to have a constant input 1.

$$y_2 = x_1 W_{21} + (1)(W_{20}) \quad (1.8)$$

The weights W_{21} and W_{20} are determined using the gradient descent algorithm. When more weights are involved, the problem becomes a high dimensional one which cannot be depicted using a 3 dimensional space. But the overall mechanism can be perceived identical to the above described. It is worth noting that although the bias neurons and their weights are generally omitted from most architecture layout diagrams, all neurons consist of a bias weight with a constant input 1. The learning algorithm will determine the weight of the bias together with other weights.

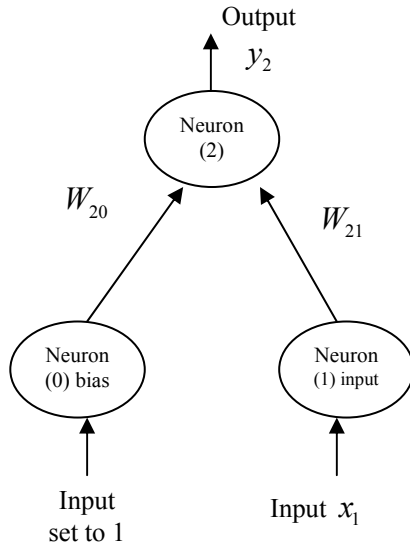


Figure 1.16.

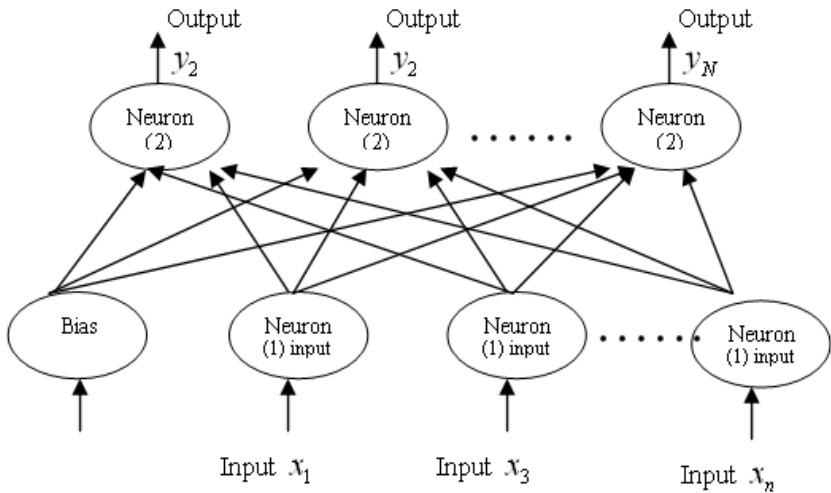


Figure 1.17.

The Current Vs Supply voltage example illustrate the determination of the current from any given supply voltage. This is a simple single input single output problem. If the problem is a more complicated consisting more than a single input and a single output variables, i.e. frequency, noise, bias, and phase, the above simple neural network model has to be modified by including more input and output neurons as shown in Fig. 1.17.

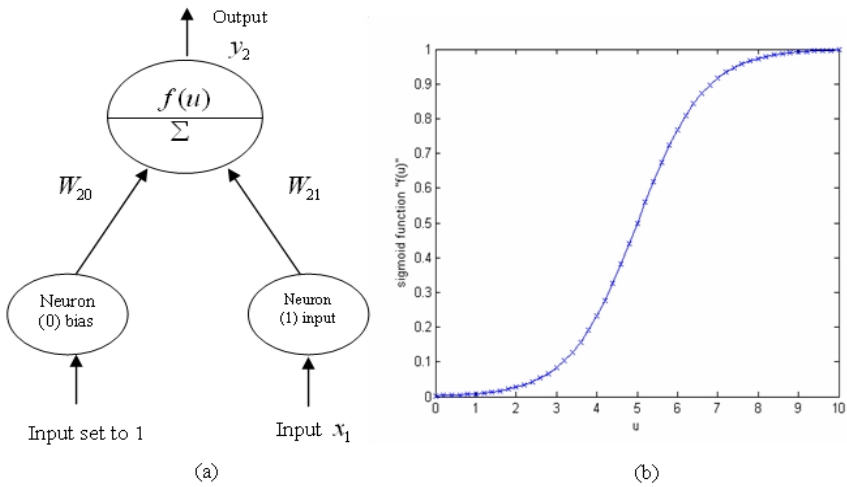


Figure 1.18. (a) A network with a neuron consisting of a nonlinear activation function $f(u)$ and (b) “sigmoid”- a typical activation function

We consider the curve fitting problem again from different perspective. In Fig. 1.14, it is clear that a curve will provide a much better fit than a linear line. Thus, it is reasonable to perform the curve fitting by including a nonlinear activation function inserted at the output

of a neuron. An S-shaped sigmoid function $f(u) = \frac{1 - e^{-au}}{1 + e^{-au}}$ shown in

Fig. 1.18(b) is one of the most widely used activation functions. The output of the network with a nonlinear activation function is

$$y = f(w_{21}x_1 + w_{20}) \tag{1.9}$$

where $f(u) = \frac{1 - e^{-\alpha u}}{1 + e^{-\alpha u}}$. Here, u is the weighted input for the output neuron: $u = (w_{21}x_1 + w_{20})$. This is the widely used feedforward neural network with nonlinear activation function.

1.5.2. Gradient Descent Searching Method

In the last section, we have shown why the performance surface is paraboloid when only 2 parameters are involved. In neural networks learning, a large number of weights are involved that results in a very complex high dimensional performance surface. Searching the performance surface using gradient descent method is an efficient way to determine the networks weights iteratively. As the search is to reach the minimum of the surface, the search direction must be opposite to the local gradient. Thus we can randomly initialize the weights as $W(0)$ where the index in the parentheses denotes the number of iteration, and W denotes the weight vector. With the given initial weights, we can evaluate the gradient of the surface at $W(0)$. The weights are thus adjusted by a magnitude proportional to the gradient and a step size. The new updated weights, $W(1)$ is then obtained. This process continues in a way

$$W(n+1) = W(n) - \eta \nabla J(n) \quad (1.10)$$

where η is a constant step size, and $\nabla J(n)$ denotes the gradient of the surface at n th iteration. The search mechanism stops when the stopping criteria are met. To improve the learning process without leading to oscillation, Rumelhart suggested the learning algorithm to be modified as

$$W(n+1) = W(n) - \eta \nabla J(n) + \alpha \Delta W(n-1) \quad (1.11)$$

where the α is a constant which determines the effect of the past weight changes on the current direction of movement in weight space. The constant α , called momentum, provides the damping effect and reduces the amount of oscillation during the course of training.

It is noted that most learning algorithms use a constant η during the whole course of learning. It is, however, obvious that a flexible learning

rate that varies with different iterations can be a better learning mechanism. A relatively large learning rate at the beginning learning phase is employed but a relatively small learning rate has to be used in order to maintain learning stability when the search is close to the minimum. The ways of adjusting the learning rate according to different scenarios have never been straightforward. These involve complicated computation on the correlation between the weights change and their current learning rate. This will be discussed in later chapter of this book.

When we are handling a complex neural network, the performance surface is very complicated that cannot be depicted in a 3 dimensional space. We cannot be sure if there is only one minimum as shown in Fig. 1.15. Usually, the high dimensional performance surface may be very rugged that results in many local minima. One will never be sure that whether or not our searching is stuck in a local minimum. Also, we will never be able to obtain a zero error despite many trials. The searching mechanism is stopped when the magnitude of the error reaches a predefined level or when certain number of iteration is reached. This is called the stopping criteria. Trapping in local minima has also been a main concern to many neural network users. The likelihood of getting stuck in local minima increases when the complexity of a problem increases because the high dimensional performance surface may become very rugged that results in many local minima. One of the ways to relieve this problem is using advanced learning algorithms or even employing different cost function that may have an effect of changing the performance surface. These issues will be detailed in later chapters of this book.

Exercises

Q1.1. A two input binary neuron shown in Fig. 1.19 has a unit step activation function with bias = 0.5. Find the space of possible values of weights of α , and β for input x_1 and x_2 respectively if the neuron is

- 1) OFF for input (1.0, 1.0);
- 2) OFF for input (0.5, -1.0);
- 3) ON for input (0.5, -0.5).

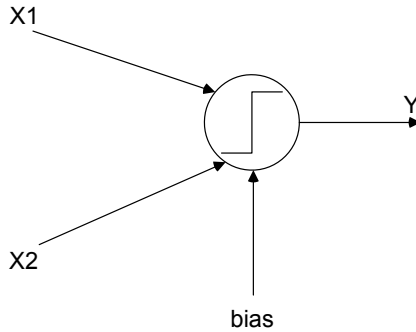


Figure 1.19.

- Q1.2. The gradient vector of a function $g(x_1, x_2) = 12x_1^2 + 3x_2^2$ is defined by a column vector $\nabla g = \left[\frac{\partial g}{\partial x_1} \quad \frac{\partial g}{\partial x_2} \right]^T = c$. It is given that the normalized gradient vector $\bar{c} = \frac{c}{\sqrt{c^T c}}$. Use gradient descent method to search the minimum for $g(x_1, x_2)$ with a given initialization of $x(0) = [13]^T$. Show and find the updated values $x(1)$ and $x(2)$, with a step size of 0.5.
- Q1.3. An LMS algorithm is used to implement a dual-input, single-weight adaptive noise canceller as shown in Fig. 1.20. Use the LMS rule to adjust the weight w of the adaptive filter. Set up an updating equation relating $w(n+1)$ to $w(n)$, $y(n)$, $x(n)$ and the learning rate η .

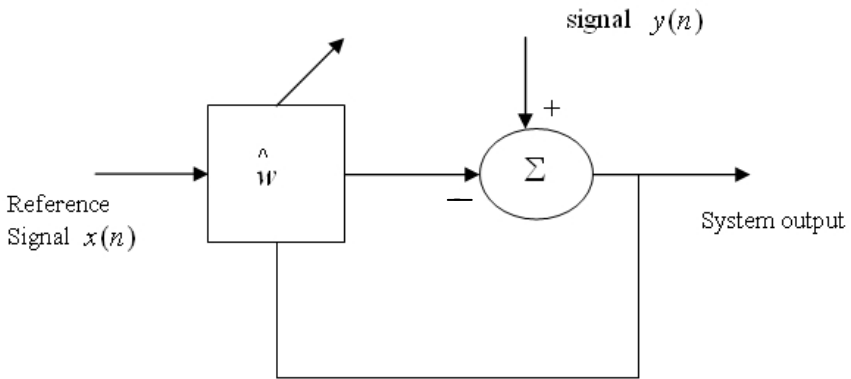


Figure 1.20.

Q1.4. Write a simple program to generate a dataset relating y to x_1 , x_2 , x_3 , and x_4 as the following. Use a feedforward network to model the following equation. Use 100 data for training and another 50 unseen data as testing.

$$y = 5 + 4x_1 + 3x_2 - 2x_1x_4 + 2x_3^2$$

Q1.5. Fig. 1.21 and Fig. 1.22 show the simple McCulloch and Pitts (MCP) model used for modeling a logic function “AND”. It shows the separation line with a gradient of -0.7 , which intersects the y -axis at 1.3 , where x_1 is denoted by the y axis, and x_2 is denoted by the x axis. Assuming the threshold T is 1 , determine the weights w_1 and w_2 for input x_1 and x_2 respectively.

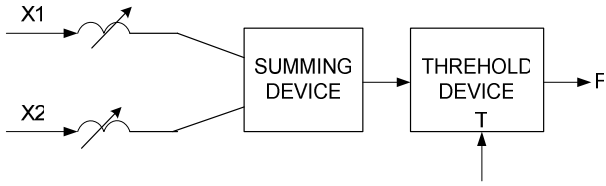


Figure 1.21.

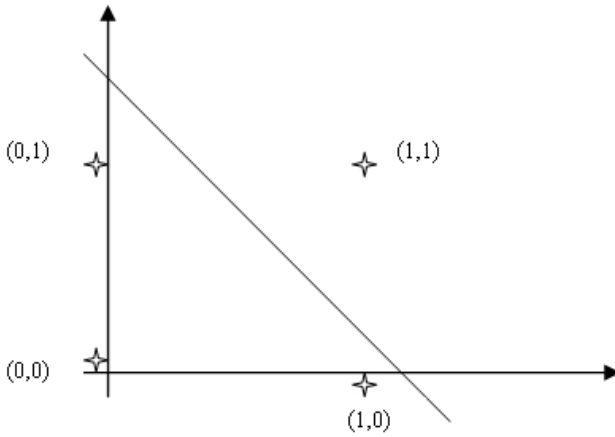


Figure 1.22.