Connectionist Learning Architecture
Based on an Optical Thin-Film Multilayer Model

by

Xiaodong Li

A thesis submitted for the degree of
Doctor of Philosophy
of the University of Otago, Dunedin
New Zealand

April 1997

To my beloved grandma

Abstract

Connectionist models consist of large numbers of simple but highly interconnected “units”. They represent an approach that is quite different from that of classical models based on the structure of Von Neumann machines. Although the term “connectionist models” often refers to artificial neural network models, which are inspired directly by the biological neurons, there are also other connectionist architectures that differ significantly from this biological exemplar. This thesis describes such a “novel” connectionist learning architecture based on the technology of optical thin-film multilayer.

The proposed connectionist model consists of multiple thin-film layers (similar to simple processing units in a neural network model), each with different refractive index and thickness. A light beam incident perpendicular to the surface of the multilayer stack is used to carry out the required computation. The reflectance of the light incident can be used as the general measurement of the outputs. Inputs can be fed into the system by encoding them into some system parameters such as refractive indices, and individual layer thicknesses can be used as adjustable parameters that are equivalent to the connection weights of a neural network model. Since this approach involves optical signal processing, the proposed connectionist learning architecture has unique properties and could offer significant advantages.

Much of the work has focused on developing this new connectionist learning architecture and investigating its capability to accomplish complex computational tasks which have been extensively studied for conventional connectionist models such as the widely used feed-forward neural network using the back-propagation learning by gradient descent. A prototype simulation model has also been built from first principles. A wide range of experiments have been conducted on this simulation model with commonly used data sets from the field of connectionist models.

It has been demonstrated that the proposed connectionist learning architecture is capable of learning many complex computational tasks (supervised learning in particular) that are typical of conventional connectionist models. Its performance in the learning examples described in this research are comparable with those of a feed-forward neural network model using the back-propagation learning algorithm.

This research suggests that the proposed connectionist learning model, which is based on an optical thin-film multilayer model from the field of optics, can stand as a viable computational learning model in its own right, regardless of its feasibility for an optical realization. Nevertheless, such an optical realization would also offer the possibility of more or less instantaneous evaluations of the "trained" thin-film model.

Compared with the scale and extent of the empirical and theoretical research conducted in the field of artificial neural network models, the proposed connectionist learning architecture is still at its initial stage of development and exploration. Further research and experimentation are definitely worthy of more investigation.

Acknowledgements

I would like to especially thank my supervisor, Dr. Martin Purvis, for his guidance over these years in shaping this research, his continuous encouragement and many insightful discussions helped me to push forward. He has guided me into the world of scientific research and taught me not only how to do research, but also how to dedicate myself to it. Without his initial suggestion for this work, his enduring help and vast background knowledge of thin film optics, this thesis would never be possible.

Throughout these years, I have received much help from people of within and outside the Information Science Department. Among them, I would like to thank Dr. Antre Everett of MBA program, Dr. Stephen Cranefield, Dr. Peter Firn, and Aurora Diaz for their valuable discussion and advice on my thesis and its proof-reading; Dr. John Shanks of the Mathematics Department for his help on the optimization theory; Prof. Ian Hodgkinson of the Physics Department for providing validation on my thin-film experimental results; Alec Glove and Eric Koh for proof-reading of my writing; Dr. Feng Zhang and Kim for discussion on using some data sets.

I would also like to thank those experts in the field of connectionist models, who I have only met on the INTERNET, for their invaluable resource of information and enlightening discussions. It has been of great value to this research. The seminars organised by the Information Science postgraduate group has also been a valuable source of inspiration and help. I also appreciate the constant and prompt system support from Peter George and Ivan Mason in these years. I value greatly many friendships in our lab and other labs in the department that I gained during my study here.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

> *The whole is more than the sum of its components, just as a van Gogh painting is so much more than a collection of bold brushstrokes. This is as true for a human society as it is for a raging sea or the electrochemical firing patterns of neurons in a human brain* (Coveney & Highfield 1995: pp.7).

According to physical theory, matter in this world is composed of many smaller particles. For example, a cup of water can be considered to be made up by many smaller "units" of water, which associate or connect in certain ways among each other to make all the units behave as a whole, i.e. a cup of water. Each such unit of water, in turn, consists of even smaller particles – water molecules, that associate among each other to make up an unit. It can be noted that each particle in a piece of matter can be "elementary", but an object comprising many such simple elements can be dynamic and complex. What is the cause of such complexity by these numerous simple "elements"? It is due, not so much to the complexity of the individual particle, but more to the complexity of the "connections" or "interactions" among these particles. Studying how these simple elements associate with each other to form complex behaviours might help us understand various phenomena that exist in this world. Models that are used to simulate such phenomena are called *connectionist models*. Typically a connectionist model consists of a large number of simple processing elements, which are connected together to form a network (Clark & Lutz 1992). This research develops such a connectionist model that derives from the field of optics, based on the technology of optical thin-films. In this chapter we begin with a background description of connectionist models, the motivation for doing this research, and the research goals. The last two sections describe the contribution of this research and outline the thesis structure.

1.1 Background and Motivation

Connectionist models differ significantly from the more conventional symbol-processing models (Fisher & McKusick 1989), in which there is a set of arbitrary symbol "tokens" together with rules for manipulating them (Harnad 1992). Connectionist models can exhibit intelligent behaviour without storing, retrieving, or otherwise operating on structured symbolic expressions (Fodor & Pylyshyn 1988). Connectionist models are "taught" (presented with examples) rather than programmed (specified with rules). They are powerful learning devices, which after exposure to

many instances of the problem set, can learn a set of internal representations that will enable the system to solve other similar problems in that domain. Connectionist models can be applied successfully to problems which are often difficult to treat with conventional models, such as rule-based systems. This has caused a resurgence of interest and enthusiasm for connectionist models in various research areas.

Current work on connectionist models has been focused largely on artificial neural networks (Hinton 1989; Farmer 1990; Dorffner 1993; Khanna 1990), which are inspired by the networks of biological neurons in the human brain. The first abstract model of a neuron was proposed by McCulloch and Pitts in 1943, as shown in Figure 1.1.

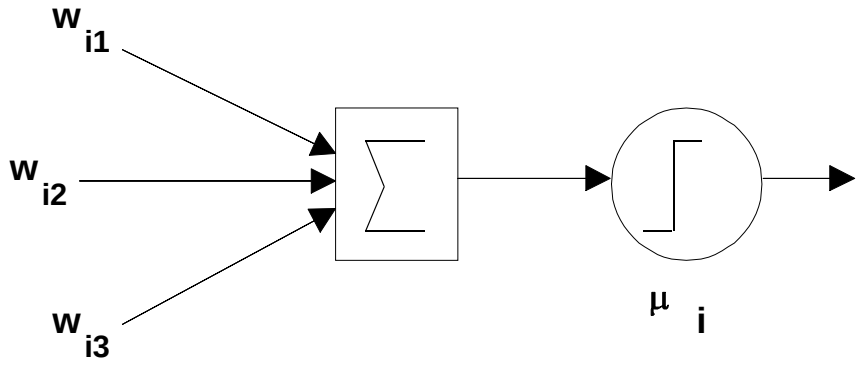


Figure 1.1. McCulloch-Pitts neuron. The unit fires if the weighted sum $\sum_j w_{ij} n_j$ of the inputs reaches or exceeds the threshold μ_i.

The McCulloch-Pitts neuron is simply a binary threshold unit. The model neuron computes a weighted sum of its inputs from other units, and outputs a one or a zero according to whether this sum is above or below a certain threshold as shown in the following expression:

$$n_i(t+1) = \Theta(\sum_j w_{ij} n_j(t) - \mu_i) \tag{1.1}$$

Time t is considered as discrete, with one time unit elapsing per processing step. n_i, which represents the state of neuron i, is either 1 or 0, corresponding to the neuron "firing" or "not firing", respectively. $\Theta(x)$ is the unit step function:

$$\Theta(x) = \begin{cases} 1 & \textit{if } x \geq 0; \\ 0 & \textit{otherwise}. \end{cases} \tag{1.2}$$

McCulloch and Pitts proved that a synchronous assembly of such neurons is capable in principle of universal computation for suitably chosen weights w_{ij}.

Real biological neurons are much more complex than the above description. There can even be significant logical processing, e.g. AND, OR, NOT within the dendritic tree (Hertz, Krogh & Palmer 1991). However the nonlinear relationship between the input and the output of a unit is a universal feature of all connectionist models, as Hertz, Krogh and Palmer (1991: pp.4) remark:

> *Our working hypothesis is that it is the nonlinearity that is essential, not its specific form. In any case, continuous-valued units can be modelled too, and are sometimes more convenient to deal with than threshold units.*

It is a generally held view that a nonlinear relationship between the input and the output of a neuron is essential for it to be useful in any kind of computational learning. However this nonlinear relationship may take almost any form, not necessarily a threshold level of summation of inputs that arrive at the neuron.

The most influential development of artificial neural networks in recent years has been the investigation of the feed-forward neural network model which employs the back-propagation learning algorithm (Dorffner 1993), and is now the most widely used neural network model (Hart 1992). Many people have applied this model successfully to different applications (Pao 1989; Hart 1992; Ripley 1993; Cherkassky, Friedman & Wechsler 1994).

It is important to realize that there are also connectionist models in various forms derived from many other phenomena, such as cellular automata (Serra & Zanarini 1990; Langton 1990; Garzon 1995), and these also show complex self-organizing or learning behaviour. By studying these alternative models, valuable insights may be gained into the nature of connectionist models and their properties that might have been otherwise missed in the study of artificial neural networks. Further computational power may even be realised by considering these alternative connectionist models.

One such potential candidate that is worth exploring is in the area of optical computing. Any operation in which an optical device or phenomenon is used for information processing can be called "optical computing" (Feitelson 1988). For example, research has been undertaken on developing general purpose digital optical computers which are usually based on nonlinear optical effects (Feitelson 1988; Das 1991). These systems try to mimic existing electronic computers, such as logic gates and memory elements using optical devices. The architecture of these computers is normally parallel. In artificial intelligence, optical symbolic and numeric signal processing has been attempted (Neff & Kushner 1989; McAulay 1989). There have also been some interest from the optics community in the optical implementation of neural network models (Wagner & Psaltis 1987; Farhat *et al*. 1985; Duvillier *et al*. 1994).

Optical systems may offer many desirable computational features that a connectionist model requires. For example, in an optical system, waves can cross without interfering with one another, so optical connections can be made by crossing light waves, and operations on different data can be done in parallel. However Hertz, Krogh and Palmer (1991) remark that although it is relatively easy to make the connections, an optical system would have the problem of constructing processing units exhibiting nonlinear behaviour such as thresholding in a connectionist model (as shown in equation (1.1)).

A key issue with respect to the present discussion, is to find an optical system with highly nonlinear units in the system. As demonstrated by the McCulloch-Pitts neuron above, these nonlinear units are essential for constructing a connectionist model. An optical system that consists of multiple thin-film layers is one case that fits this requirement. Thin-film problems are highly nonlinear. A single thin-film layer can be used as a simple processing unit (equivalent to a neuron) that is highly nonlinear in terms of its input and output mapping. We will describe in detail a single thin-film layer and an optical system made up by multiple such layers in Chapter 4. There are many design and optimization methods which have been developed for an optical thin-film multilayer system (Liddell 1981; Dobrowolski & Kemp 1990). In particular, Dobrowolski and Kemp (1990) have reviewed ten different optimization methods which are used for thin-film design. By using the same thin-film design program (i.e. a simulation model of thin-film design, rather than a physical device), they have also made comparisons with respect to the efficiency of these optimization methods. However, these algorithms were only applied to the solution of three different optical thin-film design problems, which are all limited within the thin-film domain. The

author has not seen anyone from thin-film research community taking a connectionist standpoint and using the unique characteristics of a thin-film system to build a connectionist model that is capable of carrying out more general computational tasks that are common in the field of connectionist models. With this in mind, in this thesis, a novel connectionist learning architecture based on an optical thin-film multilayer model is proposed and developed. In conjunction with a simulation model built by the use of the thin-film design and optimization algorithms, the proposed learning architecture is explored in the solution of a wide range of learning problems that have been frequently studied in the area of connectionist models.

1.2 Research Goal

The goal of this research is to propose and develop an alternative connectionist learning architecture to conventional connectionist models such as the feed-forward artificial neural network model, based on an *Optical Thin-Film Multilayer* (**OTFM**) model, which derives directly from the field of optics. Figure 1.2 illustrates the OTFM model architecture in comparison with a feed-forward neural network model. A detailed description on how to feed input information into the OTFM and measure its outputs is given in Section 7.4.

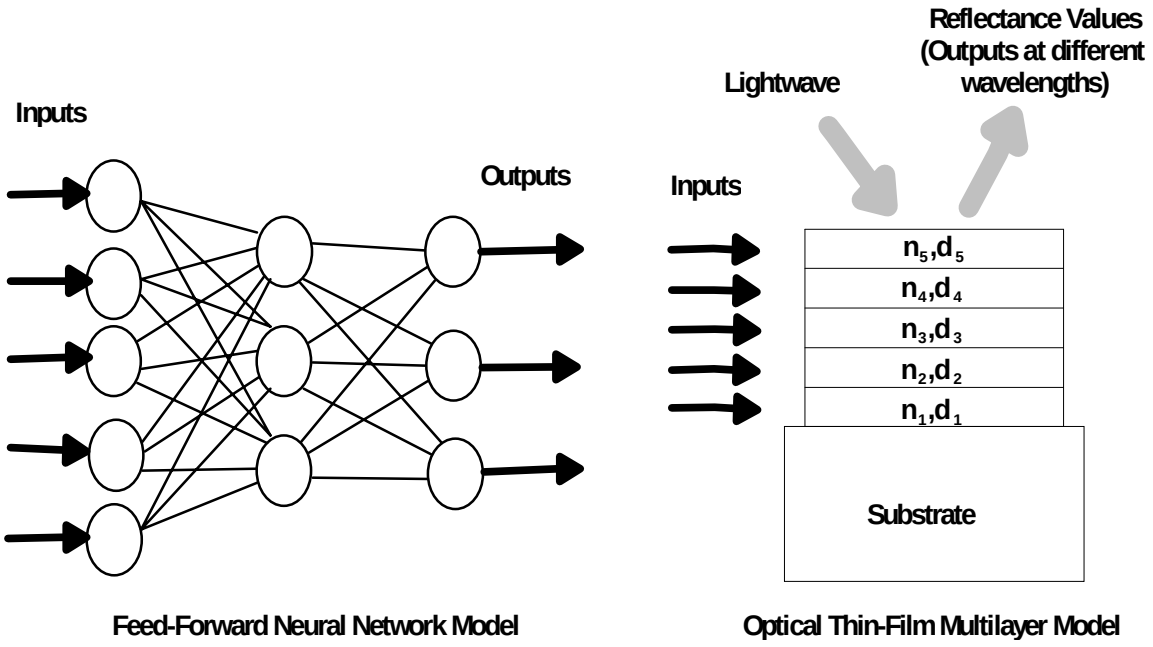


Figure 1.2. The Optical Thin-Film Multilayer (OTFM) model, in comparison with a feed-forward neural network model.

In order to provide a better understanding of OTFM's potential learning capability, the OTFM will be investigated using learning problems that have been widely studied in the field of connectionist models. The research primarily focuses on three areas:

- proposing and developing a novel connectionist architecture based on an Optical Thin-Film Multilayer model,
- investigating the proposed model's capability of accomplishing complex computational learning tasks, and
- comparing the proposed model with conventional connectionist models, in particular, the widely used feed-forward neural network models.

The learning considered in this research is supervised learning, where the calculated output of the model is directly compared with the desired output.

It is important to emphasize that this thesis does not attempt to claim that the OTFM is superior to any other connectionist model (it is often application-dependent). The aim is to provide another alternative that might be useful in solving some computational learning tasks when there is difficulty dealing with them using more conventional connectionist approaches. The models in this thesis are simulated and run on both PC and a SUN work stations, however implementation in hardware has not been attempted.

1.3 Research Approach

Since current research interest in the connectionist model field largely involves neural networks, “Neural Networks” and “Connectionist Models” are often considered to be synonymous by many researchers (e.g. Hinton 1989; Dorffner 1993), while others stress that connectionist models should include those which have different architectures (Fodor & Pylyshyn 1988; Serra & Zanarini 1990; Farmer 1990). This research takes the latter approach and compares the optical thin-film multilayer model with the feed-forward neural network model in order to identify their common features and differences. As a result of such a comparison, the relative strengths and weaknesses of their architectures can be better understood.

A connectionist model consists of a large number of simple individual processing units and connections among them. This feature makes the object-oriented modelling approach particularly suitable to the software implementation of a connectionist model. Many neural network simulation programs have been developed by using the object-oriented approach (Kock & Serbedzija 1993; Masters 1995). The proposed model is developed in a similar manner because of its connectionist nature. However, the optical thin-film multilayer model has its own unique architecture and characteristics; almost nothing can be borrowed from the programs of other connectionist models. Thus the software implementation of the simulation model must be done from first principles.

The optimization methods have always been critical in any learning system. The appropriate choice and use of an optimization method can influence heavily the performance of a learning system. In general a search algorithm or a hybrid search algorithm that is able to search well both globally and locally in a multidimensional space of system parameters is desired. We have adopted two search algorithms with this ability, the so called N-squared Scan Method (Liddell 1981) and Genetic Algorithms (Holland 1975; Goldberg 1989; Michalewicz 1996), though in many cases, it might be sufficient to use either one to carry out a required learning task. In the N-squared Scan Method, the search starts with large steps first, in order to survey a large area of the search space, then it uses smaller steps in order to focus on a local regional search for a better minimum. The scope of the search may be more easily and directly controlled by the designer. When the complexity of the learning problem grows, using the N-squared Scan Method increases the likelihood of missing some better minima globally. A good complement to this is to use a Genetic Algorithm, which is well known for its ability to find good solutions globally for difficult high-dimensional problems. Especially for a complex learning problem, a general approach is to use a Genetic Algorithm first for a good overall search and then use the N-squared Scan Method for a fine-tuning local search of better minima. It has been found that combining global and local searches can improve the performance of a learning system dramatically, especially in solving some difficult learning tasks (Shang & Wah 1996).

By using the OTFM simulation model, various experiments have been conducted in order to investigate the degree to which the OTFM compares with the existing conventional connectionist models. Many issues had to be considered in this regard, such as how the model is trained, how inputs are fed into the model, how to evaluate the performance of the model, how to estimate the

error rate, how the model performs when noise is present, what data sets to use, and how to specify the initial system parameters, etc. The training considered here is supervised learning, in which learning is done on the basis of direct comparison of the output of the model with known correct answers. The inputs are encoded into the refractive indices of thin-film layers, and during the training phase, the performance is measured by directly comparing the outputs of the OTFM, its reflectance values, with the desired outputs at different wavelengths (see Section 7.2 for detail). The approach adopted for estimation of the error rate is in line with those widely used in the connectionist research community, such as resampling techniques. The data sets used in these experiments are widely used in connectionist learning experiment, ranging from small data sets such as the XOR, parity and 5-by-5 grid pattern recognition problems to "real-world" data sets such as iris data classification (Fisher 1936), gas furnace time series (Box & Jenkins 1970) and breast cancer prognosis (Michalski *et al*. 1986). In some problem solving, the OTFM's performance is evaluated by not only its ability to learn and classify the training examples, but also its ability to correctly classify previously unseen examples (generalization). More experiments with different initial values for the system learning parameters, in particular optical constants, are needed for further exploration of this model, since performance may depend greatly on the starting design of these learning parameters and the optimization methods used. The results of experiments, as will be shown in subsequent sections of this thesis, have been analysed and compared with those conducted on a feed-forward neural network model employing the back-propagation learning by gradient descent.

1.4 Contribution of the Research

The following provides a list of contributions this research makes to the existing knowledge of Artificial Intelligence:

- Proposing and developing a novel connectionist learning architecture based on an Optical Thin-Film Multilayer (OTFM) model, that can be used as an alternative to the conventional connectionist models.

- Constructing a prototype simulation model for the proposed OTFM from first principles.

- Experimenting with the OTFM on a wide range of representative learning examples selected from the field of connectionist leaning, and as a result, showing that the OTFM is comparable to the conventional connectionist models in learning many complex computational tasks.

1.5 Thesis Outline

This thesis begins with a review of connectionist models in Chapter 2, where different definitions and approaches are discussed. Various connectionist architectures are then described in more detail and illustrated with two examples, the cellular automata and the feed-forward neural network employing the gradient descent learning algorithm. A brief comparison between the neural network model and the OTFM is also given.

Chapter 3 provides a background for the OTFM by giving a brief overview of the thin-film technology, where the primary concern is on how the thin-film design can be used for constructing a learning system.

In Chapter 4, the architecture and computational process of the proposed Optical Thin-Film Multilayer (OTFM) model are described in detail, with emphasis on how this model can be viewed as a connectionist model, how it achieves the required computational task, and what thin-film calculation algorithms and computationally cost-effective procedures are involved. The architecture of this model is then compared with neural network models.

Chapter 5 is devoted to optimization issues concerning thin-film design in general, as well as the two search algorithms adopted for the proposed thin-film model, the N-squared Scan Method and Genetic Algorithms.

Chapter 6 describes the development a software implementation of the simulation model using the objected-oriented modelling approach. The building blocks of different data types (or classes) are described in a sequence starting with simple ones, and extending to complex ones. A diagram of the class hierarchy is presented to depict the inheritance structure. The GA-based OTFM implementation and related issues are also briefly discussed.

Chapter 7 describes issues concerning conducting effective experiments, including the training procedure, the OTFM system parameters, input encoding methods, data sets, noisy data and missing information, and the resampling techniques used for estimation of the error rate.

In order to see the OTFM's capability to learn and generalize under different circumstances, Chapter 8 presents problem solving examples of various computational learning tasks carried out on the Optical Thin-Film Multilayer simulation model, including problems such as XOR, parity, and “real-world” classification problems, such as the iris data classification and gas furnace time series.

Chapter 9 gives a more detailed analysis of these experimental results, along with the experimental results of these learning problems from published works of other researchers. Analysis is largely by comparison to the feed-forward neural network model.

Chapter 10 summarises the results and conclusions reached from the study of the proposed OTFM. The strengths and limitations of the OTFM which were observed during the investigation are also presented. Finally, possible areas of exploration for future research are discussed.

Chapter 2
Connectionist Models

2.1 Introduction

Information processing is conventionally viewed as a sequence of discrete operations (McClelland 1988). The direct consequence of this approach is the implementation of processing on today's computers based on the Von Neumann architecture, where operations proceed back and forward between a single powerful processor and a large memory (Hillis 1985). In this chapter, an alternative approach called the connectionist model will be described. *Connectionist models* consist of simple processing units which can be "connected" to form a network (Clark & Lutz 1992). *Neural networks* that are inspired from biological neurons are the most well-known connectionist models, but there are also other types of connectionist models that derive from different phenomena and have distinct characteristics. The *Optical Thin-Film Multilayer* model investigated in this research, which is based on the technology of thin-films, can be viewed as such a different connectionist model. The study of connectionist models is highly interdisciplinary. It is necessary first to describe a number of more general definitions of connectionist models, so that the common ground can be more evident.

Two examples of typical connectionist models are presented here. One is a *cellular automaton*, which is extremely simple but can demonstrate impressive complex learning or self-organizing behaviours (Serra & Zanarini 1990; Raghavan 1993; Garzon 1995). The other is the well-known *feed-forward artificial neural network*, which has shown the ability to learn almost any kind of nonlinear mapping (with hidden layers and under certain conditions) (Simpson 1990; Khanna 1990; Hart 1992; Kosko 1992). A third connectionist model, the Optical Thin-Film Multilayer model, is the subject of this thesis and will be described in detail in subsequent chapters. These connectionist models have demonstrated many desirable properties which are lacking in conventional learning models. These properties of the two example connectionist models are described in detail in connection with a pattern recognition example, so that they can be compared with those of an Optical Thin-Film Multilayer model as well.

For simplicity, "OTFM" denotes the Optical Thin-Film Multilayer model.

2.2 Definitions

The term "connectionist models" usually refers to a class of models that compute by way of connections among simple processing units. It is also a term that is frequently applied specifically to neural network models (Hinton 1989; Dorffner 1993; Farmer 1990). Since connectionist models are a subject of research under various disciplines, there are various definitions and viewpoints. A number of connectionist architectures of a more general nature are considered here. Clark and Lutz (1992) define "connectionist models" as:

> *a generic term which embraces a large variety of specific algorithmic forms and architectures. What unites the models is a commitment to the use of simple individual processing units, arranged to process information by a form of cooperative parallelism.*

McClelland (1988) provides a general connectionist architecture to encompass all kinds of connectionist models (Figure 2.1). It shows an autoassociative network, which consists of a set of completely interconnected units (but there is no self connection). All units are both inputs and outputs. Each unit connects to all other units with modifiable weights. There might be many variations of this architecture. For example, some units may receive no input from the environment; some may send no output outside the net; and some of the interconnections among units in the network may be deleted.

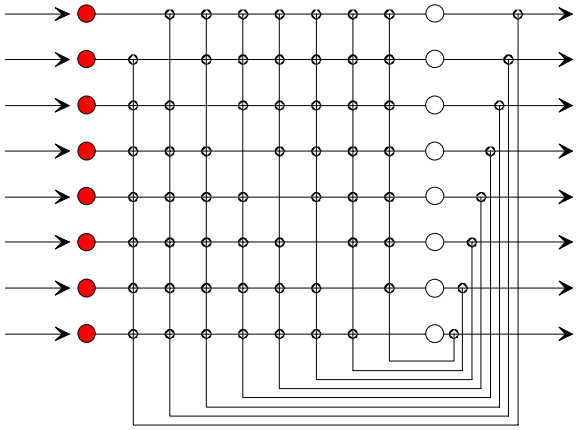


Figure 2.1. A fully connected autoassociative network, with connections from each unit to every other unit. Each unit receives input from outside the network and sends output outside the network. All connections may be either positive or negative in this very general formulation.

Farmer (1990) defines a connectionist model to include its dynamical behaviour and lists two properties:

- *The interactions between the variables at any given time are explicitly constrained to a finite list of connections.*
- *The connections are fluid, in that their strength and/or pattern of connectivity can change with time.*

By using a dynamical approach, a connectionist model can be described by its states, as a dynamical system evolves from an initial state through a succession of states. For example, suppose we have a model in which the initial state is represented by the values of three parameters x_1, x_2 and x_3. For a discrete-time dynamical system, such a transformation of the system from one state to another can be described by the following equations:

$$x_j(t+1) = f_j(x_1(t),\ x_2(t),\ x_3(t)), \qquad j = 1,2,3 \tag{2.1}$$

In general, the function f_j is nonlinear. Each state parameter evolves from $x_j(t)$ to $x_j(t+1)$.

Imagining the parameters as being associated with the nodes of a network, and connecting with arcs the parameters that are effectively coupled through the equation (2.1), a dynamical system can be illustrated in Figure 2.2.

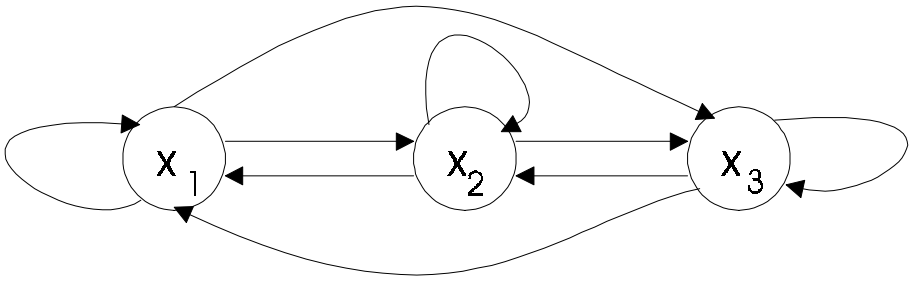


Figure 2.2. A fully connected "dynamical network", showing interactions among the parameters of the system. This network can take various forms. For example, some of the connections can be deleted, like the model shown in Figure 2.1.

The instantaneous connection strength C_{ji} of the connection from node *i* to node *j* (where *i* is an input to *j*) is:

$$C_{ji} = \frac{\partial x_j}{\partial x_i} = \frac{\partial f_j}{\partial x_i} \tag{2.2}$$

With respect to the above three definitions, Clark and Lutz (1992), and McClelland (1988) define a connectionist model by looking at the model's structure, local processing ability and physical connections, while Farmer's (1990) definition places more emphasis on the changes of its states. Connections are not explicitly displayed, instead they are demonstrated through equation (2.2). In all three descriptions, parameters in a connectionist model can be subdivided into two classes, those relating to a single unit and those relating to the connections (Serra & Zanarini 1990). The modification of these parameters in such a dynamical system can be seen as a process of learning.

2.3 Various Models

There are many variations within the scope of connectionist models. One such model is cellular automata, and another is the widely used feed-forward neural network model. They are different connectionist models in terms of architecture, but both demonstrate interesting and complex learning behaviours (Serra & Zanarini 1990; Langton 1990; Garzon 1995). This section describes these two models, so that they can be used to illustrate the general features of the connectionist architecture and then to be compared with the OTFM. The discussion of the feed-forward neural network is more detailed, because the model will be explicitly compared with the OTFM in subsequent chapters.

2.3.1 Cellular Automata

There are two or three dimensional cellular automata that exhibit very complex behaviour. For the purpose of illustration, a particular class of cellular automata, one-dimensional cellular automata (Serra & Zanarini 1990), is examined here as the illustration of an architecture of connectionist models. Consider a set of n binary elements $x_1, x_2,...,x_n$, which can take the values 0 or 1, placed in a regular fashion along a line. Each element is connected to its two "nearest neighbours" (next left and next right). At any discrete point of time, the state of each element is determined by a function of the state of the element itself and of its two nearest neighbours at the preceding time-step. For simplicity, suppose that all the elements use a Boolean function as follows:

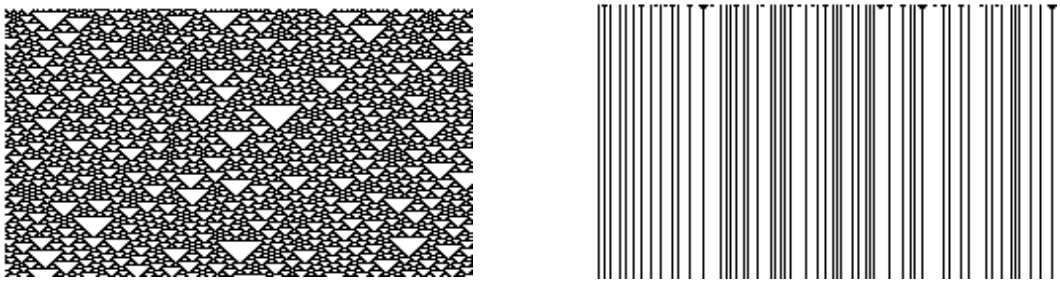
$$x_j^{(i+1)} = f(A_1 x_{j-1}^{(i)},\ A_2 x_j^{(i)},\ A_3 x_{j+1}^{(i)});\quad j = 1,2,...,n \tag{2.3}$$

The Boolean function takes three variables, $x_j^{(i)}$ and its two neighbours $x_{j-1}^{(i)}$ and $x_{j+1}^{(i)}$. Each variable can take either the value 0 or 1. A_1, A_2 and A_3 are three constant values. $x_{j+1}^{(i)}$ is the new state which is determined by the function f. Note that in general f is a nonlinear function, which can represent various rules for an one-dimensional automaton. One such rule of the Boolean function is shown in Table 2.1.

Input	111	110	101	100	011	010	001	000
Output	0	1	0	1	1	0	1	0

Table 2.1. An example of rules for the time evolution of a one-dimensional automaton. The eight possible states of the three nearest neighbour sites are shown as input. They are ordered according to the corresponding three-digit binary number. The output shows the "response" to the specific rule, which can be characterized in this case by an eight-digit binary number or by the corresponding decimal number.

Only the states of each cell's nearest neighbours are used as input in the rule. If the parity of these two states is even, the output is 0, and if the parity is odd, then the output is 1. The input to the function f can be represented by a 3-bit binary number. The rules for the outputs of the eight-variable combination of 3-bit inputs are shown in Table 2.1. It is necessary to assign a meaning to variable x_0 and x_{n+1}: the assumptions $x_0 = x_n$ and $x_{n+1} = x_1$ then correspond to the closure into a ring of the one-dimensional string of elements.



(a) (b)

Figure 2.3. Two examples of one-dimensional cellular automata. Time evolution is from the top to the bottom of the figure. Black dots represent 1's, while white dots represent 0's. Note that (a) and (b) use different rules.

Figure 2.3 (a) shows the time evolution of a one-dimensional cellular automaton from a randomly defined initial condition. The system shows a behaviour which displays complex patterns in the space-time plane; in fact, the appearance and the disappearance of triangular structures of various dimensions can be seen as the system evolves from an undifferentiated initial state to a multiplicity of well-differentiated space-time domains.

In Figure 2.3 (b), the automaton uses a different rule. After a short initial transition, the system behaves time-independently in an extremely regular manner. The number of "lines" and the distance between "lines" depend upon the initial states. In this case, if the initial states are considered as inputs, it can be said that the cellular automata model has "learned" specific spatial patterns within the initial conditions.

Both Figures 2.3 (a) and (b) show limited connections among the units. Each unit interacts with other units only by using a Boolean function determined by the states of itself and its two nearest neighbours at the preceding time-step. In this case, a connection can be considered to be an interaction between two units with a constant value. A_1, A_2 and A_3 in equation (2.3) are used as such constant values. Although the connections are not modifiable, the model still demonstrates complex behaviours which can be considered as learning, or self-organization (Serra & Zanarini 1990). Cellular automata have also been adopted in applications of pattern or image recognition systems (Raghavan 1993).

2.3.2 Feed-Forward Neural Networks

Today's research of connectionist models is largely involved with neural networks (Hinton 1989; Farmer 1990; Dorffner 1993). Neural network models are inspired by natural physiology and mimic the neurons and synaptic connections of the brain (Hertz, Krogh & Palmer 1991). There are many different artificial neural network architectures that have been studied, but all share some common features.

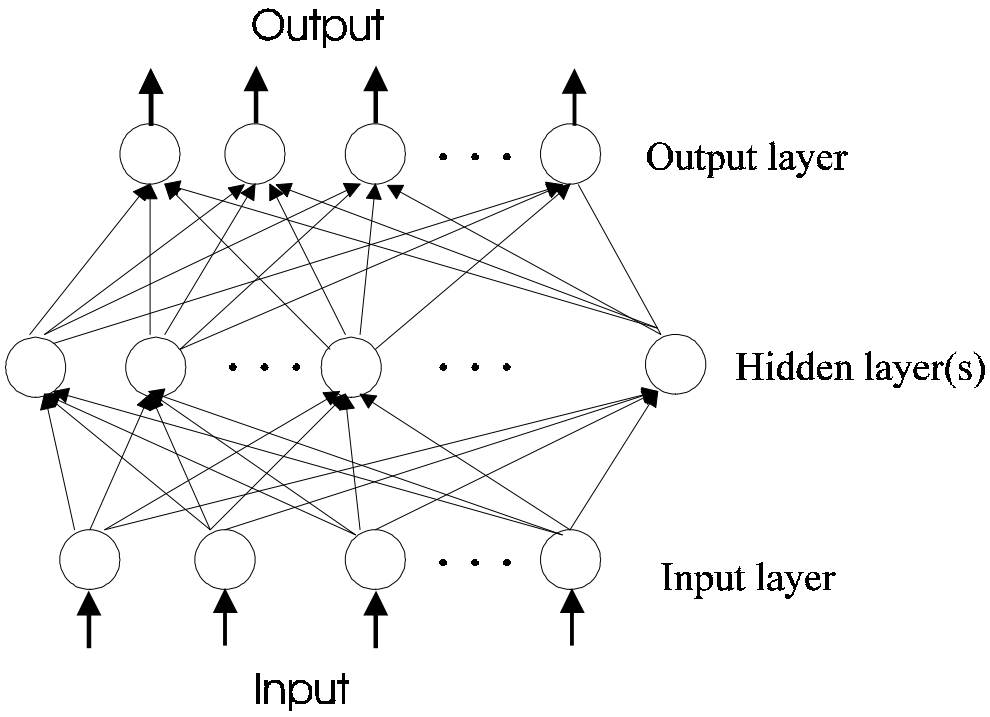


Figure 2.4. A feed-forward neural network model.

Figure 2.4 shows the architecture of a typical feed-forward neural network model. It has three layers: a layer of input nodes, an intermediate layer of nodes and a layer of output nodes. Connections are from each input node to each intermediate layer node, and from each intermediate layer node to each output node in a feed-forward manner.

These nodes of different layers are analogous to biological neurons and are the processing units of the network. A node has an output value, which is determined by a function of the outputs from those other nodes which feed into it. Nodes are connected by direct links with associated weights, which can be positive or negative. This is shown in Figure 2.5, where the three nodes on the left feed into the rightmost node.

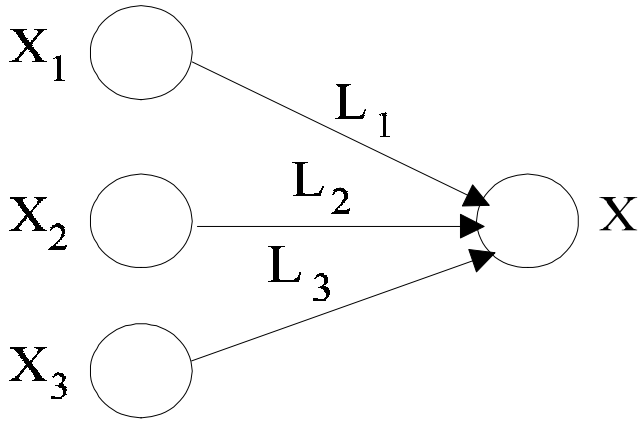


Figure 2.5. Nodes in a network.

If the output of each node is X_1, X_2, X_3, and X (i.e. the rightmost node), and the values of the weights are L_1, L_2, and L_3, then X is given by

$$X = f(\sum X_i L_i + \theta) \tag{2.4}$$

where θ is a threshold associated with the node, and f is the activation function. Usually a nonlinear function f is characterised so that if X_i is high, and L_i is positive, a high value of X_i will tend to increase X, whereas if L_i is negative it will tend to decrease X. In this way positive weights are analogous to excitatory synapses, and negative weights are analogous to inhibitory synapses of biological neurons.

Although simple in structure, the systemic behaviour of a network in which many such nodes are connected together can be very complicated. For example, even a simple perceptron without any intermediate layers (similar to the one shown in Figure 2.5, but possibly having more input and output nodes) can perform well on some linearly separable problems, though it is incapable of learning concepts that are not linearly separable (Minsky & Papert 1969; Mooney *et al.* 1989).

In order for a network to be able to perform a particular task, it needs to have an appropriate architecture, i.e. layout of the nodes and links, and a set of weights and other parameters. A widely used network architecture is the feed-forward multilayer perceptron neural network (as

shown in Figure 2.4), which is often used for classification tasks, such as pattern recognition. The nodes in the network usually take real values in the range 0.0 to 1.0, and the activation function is generally a nonlinear *squashing* function, such as

$$f(x) = \frac{1}{e^{-cx} + 1} \tag{2.5}$$

where x denotes the net input to a node. As shown in Figure 2.6, equation (2.5), representing the sigmoid function, has a maximum value of 1.0 and a minimum value of 0.0. c determines how sharp the curve is. This function is convenient because its output is everywhere differentiable and saturates at both extremes, 0.0 and 1.0.

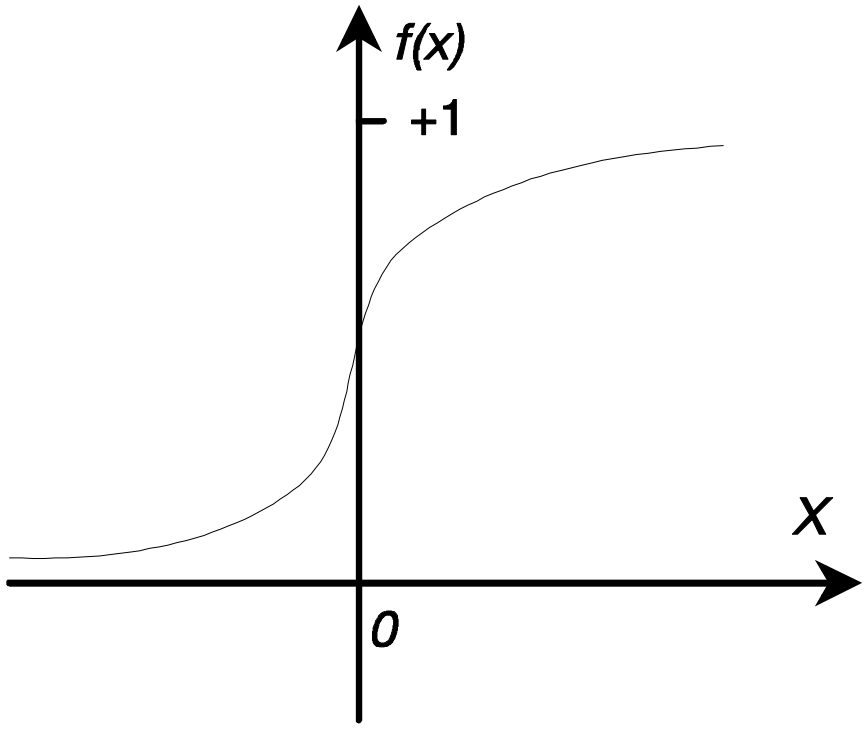


Figure 2.6. The *squashing* function, which is differentiable and saturates at both extremes.

A network with N input, H hidden and M output nodes can be referred to as a *N:H:M* architecture, and such a network can be used to represent a nonlinear function f with N inputs and M outputs:

$$f\colon (I_1, I_2, \ldots, I_n) \rightarrow (O_1, O_2, \ldots, O_m) \tag{2.6}$$

It is represented by examples of the form $(I_1,I_2, ..., I_n; O_1,O_2, ..., O_m)$. Because of the squashing function, the outputs (O_i) must be in the range 0.0 to 1.0, although they can be scaled to meet the needs of practical modelling requirements.

A feed-forward neural network works in the following way. The input values (*Is)* are presented to the input nodes which feed into and activate the hidden layer(s). The values produced by the activation functions at the hidden nodes, in turn, feed forward into the output nodes, and then determine values of the output nodes. With hidden layers, feed-forward neural networks can learn very complex classification problems (Hertz, Krogh & Palmer 1991).

The learning described here in a feed-forward neural network model is called *supervised learning*, where learning is done by the direct comparison of the output of the model with known correct answers (Hertz, Krogh & Palmer 1991). It is achieved through training the model to attempt to minimize the error function, which is a function measuring the performance of the network

$$E = \frac{1}{2}[\sum \sum_i (O_i - \Omega_i)^2] \qquad (2.7)$$

where the outer sum is over all the examples in the training set[1]. O_is are the outputs currently produced by the network, and Ωs are the targets. E is a function of all of the parameters. A numerical optimization method called gradient descent is usually adopted to minimize E. Suppose that the function to be minimized is $E(p_1,p_2,...,p_N)$, where each p represents a network parameter. We have a range of initial values as $(P_1,P_2,...,P_N)$, such as for each p_i there is an initial value P_i. The partial derivatives $\partial E/\partial p_i$ are calculated and then each parameter is updated according to the following rule:

$$\Delta p_i = -\mu \partial E/\partial p_i \ , \quad p_{i(new)} = p_{i(old)} + \Delta p_i \qquad (2.8)$$

Δp_i is the change in p_i, and μ is a constant parameter called the "learning rate", usually less than 1.0. The updated parameters define a new estimate of the outputs at which E is reduced. The gradient is recalculated at this new point, and the parameters are updated again. The process is

[1]Training set are input data examples used for training. There are also examples not used for training, but for testing the trained model.

repeated until it is considered to have converged, i.e. that either E cannot be made any smaller by further iteration, or the outputs are stabilized and close to the targets within an acceptable scope. The parameters are usually updated after the presentation of a single example by attempting to reduce the error associated with that example, i.e.

$$e = \frac{1}{2}\sum_i (O_i - \Omega_i)^2 \tag{2.9}$$

In other words, an example is presented to the input nodes, and the outputs are determined from the current parameters. The parameters are then adjusted, using an iteration of gradient descent, and the next example is presented to the network. After many passes through the whole training set, the error E usually converges to some value and, provided this is small enough, the network can be said to have learned a relationship.

The reason for much of the interest in feed-forward neural networks is their ability to represent complicated and highly nonlinear relationships, as well as their ability to generalize their recognition behaviour to new situations. After being trained on a number of examples of a relationship, the network can often correctly predict new examples from the same data domain. The feed-forward neural network in general performs well in all the experiments described in Chapter 8, such as the parity problem, pattern recognition and classification on the iris data set.

2.4 Properties of Connectionist Models in General

Two different examples of connectionist models have been described so far. Since the feed-forward neural network is the most widely-used connectionist model, it is used here to illustrate the architecture and properties of *connectionist models in general*. Input or output nodes described in this section correspond to the input and output nodes in a feed-forward neural network, as shown in Figure 2.4. Pattern recognition is used here to illustrate some properties of connectionist models.

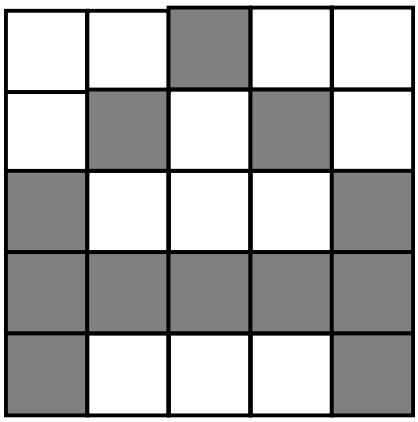


Figure 2.7. The pattern "A" is represented on a 5-by-5 grid. A shaded square is represented by 1, while a blank square is represented by 0.

In Figure 2.7, a representation of pattern "A" is {0010000101010001111111 0001}, where the squares are encoded from left to right and top to bottom. For example, the first row is {00100}, the 2nd row is {01010}, and so on. Each square (shaded or blank) corresponds to one input node. This representation can then be fed into the input nodes of a connectionist model (as shown in Figure 2.4), where for example there are 25 input nodes and 3 output nodes. A pattern of 3 squares (this number could be greater, depending on the task) {1,0,0} is the desired output for "A" at the output nodes, while we might have {0,1,0} for a different letter such as "B". The goal is to train the model to learn the mapping between the input and its corresponding output, i.e. once training is finished, when presenting the trained model with that input, it should be able to produce the corresponding output.

Properties of connectionist models (Feldman & Ballard 1982; Fodor & Pylyshyn 1988; Clark & Lutz 1992) are described as follows:

- Connectionist models can act as content-addressable memory. In Figure 2.7, suppose that the representation for pattern "A" is fed into the input nodes and is associated with another pattern, the desired output {1,0,0} at the output nodes during training. If the "A" pattern is later presented as input then the target output {1,0,0} will be produced at the output nodes.

- Connectionist models are resistant to noise. When the trained model is presented with an noise-added input pattern such as the one shown in Figure 2.8, where two squares have been changed from blank to shaded ones, the trained connectionist model still produces {1,0,0} at the output nodes, which indicates it is able to recognize the learned mapping between the input and the output, although a small part of the input pattern has been changed.

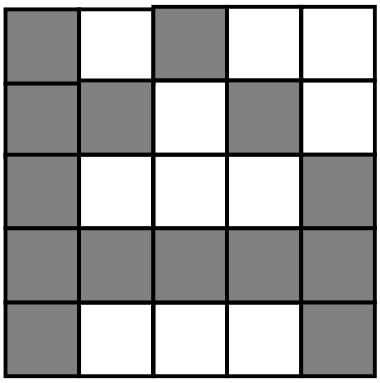


Figure 2.8. A pattern "A" with noise.

- Connectionist models are robust and degrade gracefully when damaged, which stems from the fact that many networks use distributed representations (explained at the end of this section), in which each connection contributes to many stored concepts. These networks exhibit the phenomenon that if a connection or a node is damaged or destroyed the performance of the model will degrade slightly, but no single concept will be lost.

- Connectionist models can learn. Learning usually involves modifying connection weights, and the learned knowledge is stored in the connections. There exist learning algorithms that enable neural networks to adjust their own weights without external supervision until the network performs some particular tasks. In some specific examples, learning also involves modifying other system parameters (not only the connection weights); thus the knowledge learned is also stored in these parameters (e.g. as will be shown later in the OTFM, the wavelength is a parameter involved with learning).

- With respect to knowledge representation and learning, connectionist models can perform some computational tasks that might be more difficult for conventional numerical or rule-

based systems. Rule-based systems, for example, are not error-tolerant: if one of the rules fails, the whole system might fail ("all-or-none").

- Connectionist models can develop their own internal distributed representations from input data. These internal representations can enable the system to generalize behaviour beyond the training set (a good example of this is presented in Chapter 8, on the iris data classification).

The strength of the above properties depends to some extent on what the nodes are supposed to represent. Connectionist models usually use distributed representations, in which concepts are represented by patterns of activity distributed over many nodes, and each node takes part in many such patterns (Hinton 1989). Distributed representations allow the weights to capture important underlying regularities in the task domain, thereby leading to increased generalization.

2.5 Connectionist Learning in a Thin-Film Multilayer Structure

As discussed in the preceding section, a connectionist model has many desirable properties, such as the ability to learn and deal with problems that might be inapplicable for a conventional rule-based system. In comparison with the two different connectionist architectures already described in Section 2.3, the OTFM has its own unique architecture and properties (see Chapter 3 and 4 for detail). In particular, its learning process reflects its unique optical properties.

The OTFM consists of two components, the thin-film multilayer structure and the light wave. A single thin-film layer can be considered as a basic processing unit in the model, and the interactions or "connections" among all the layers occur through a light wave being reflected and transmitted at the boundaries of each layer. The input can be encoded into either a range of wavelengths or refractive indices of thin-film layers. The reflection coefficients at various wavelengths can be used as the outputs of the system, which are quite different from a neural network (i.e. a neural network usually has a specific set of nodes as its output nodes). The thickness of each layer and the wavelength can be used as adjustable learning parameters, similar to the connection weights of a neural network. Since optical thin-film processing in this manner is highly nonlinear, choosing a set of appropriate thicknesses of individual layers can achieve a complex mapping required between the inputs and the outputs. Thus the OTFM can be viewed as a novel connectionist

model whose architecture and properties are different from other connectionist models currently in use. The next chapter provides some background information concerning optical thin films and how such a thin-film model can be adapted to develop a learning model.

2.6 Summary

Connectionist models have been described by many researchers from different perspectives. This chapter has presented a number of more general definitions, followed by an illustration of two well known connectionist architectures. By describing these two examples, we can note that connectionist models may take different forms (e.g. cellular automata or a neural network), but share common features such as having simple processing units and connections among those units. The properties of connectionist models can overcome some of the limitations of conventional computational systems. In order to see how the OTFM can be viewed as a learning model from perspective of connectionist models, a brief description of the OTFM was made in comparison with neural network models. This comparison will continue in the upcoming Chapters 3 and 4 in order to examine the nature of the OTFM as a connectionist learning model.

The next chapter will present some background information on the thin film technology, which is fundamental to the development of the Optical Thin-Film Multilayer model that will be described in the subsequent chapters.

Chapter 3

Thin Film Technology

3.1 Introduction

In this chapter we first answer the question of what thin films are, and then discuss the optical properties of thin films and how their unique characteristics can be used to develop a novel learning model. *Thin film optics* is a well developed research area, and there is a considerable amount of literature on its calculation, design and applications. The research reported in this thesis requires some background coverage of optics and thin films, and this will be covered in this chapter. Only optical information necessary for describing this research will be presented. Additional background concerning thin film optics is provided by Born and Wolf (1980), Chopra (1969), Liddell (1981), Fuman and Tikhonravov (1992).

3.2 What are Thin Films?

According to Chopra and Kaur's definition (1983):

> *A solid material is said to be in thin film form when it is built up, as a thin layer on a solid support, called substrate, ab initio by controlled condensation of the individual atomic, molecular, or ionic species, either directly by a physical process, or via a chemical and/or electronchemical reaction.*

Liddell (1981) defines a thin film in mathematical terms:

> *A thin film is one whose thickness is of the order of the wavelength of light and whose extent is infinite compared to its thickness; we assume it is bounded by semi-infinite planes. The film is characterised by its refractive index, its absorption coefficient and its thickness (assumed to be uniform).*

It should be stressed that it is not simply the “thin” thickness that provides thin films with special and distinctive properties, but more importantly, the microstructure resulting from the unique way of their coming into being by progressive addition of the basic building blocks one by one.

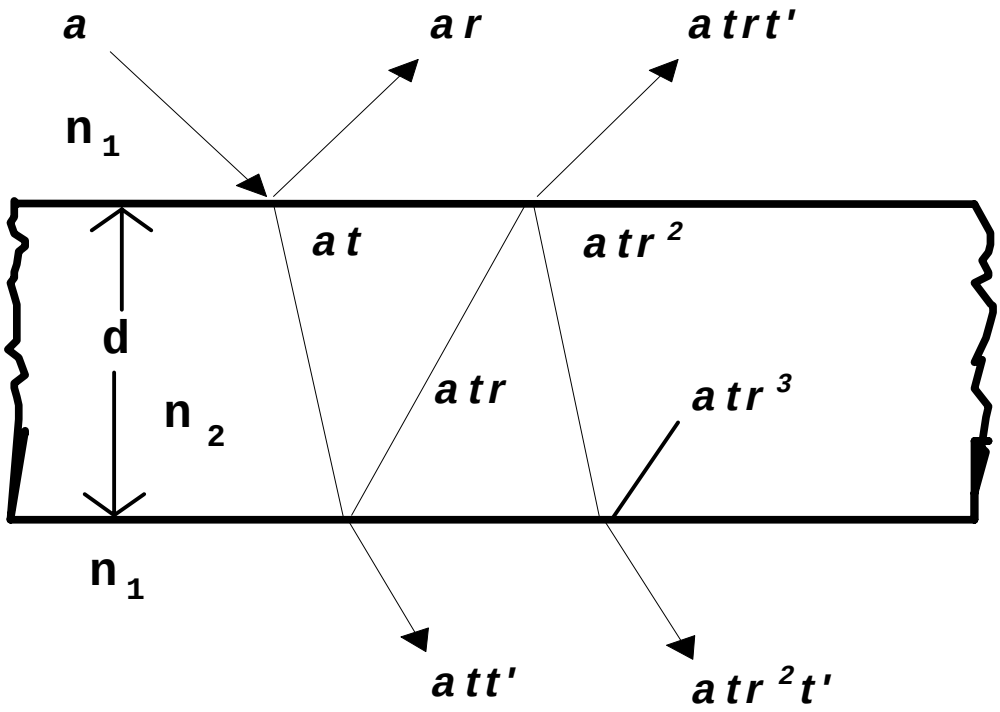


Figure 3.1. A light beam incident on a plane parallel-sided thin film.

Figure 3.1 shows a light beam incident *a* at an angle on a plane parallel-sided isotropic film of refractive index n_2 surrounded by media of index n_1 (e.g. refractive index of vacuum which is 1.0). The thickness of this thin-film layer is *d*. The *r*'s and *t*'s are the Fresnel coefficients of reflection and transmission, respectively. The Fresnel coefficient *r* here is often denoted by *f*, hence coefficients *f* and *t* can be defined as follows,

$$f = \frac{n_2 - n_1}{n_2 + n_1} \quad \text{and} \quad t = \frac{2n_2}{n_2 + n_1} \tag{3.1}$$

In this research, such a single thin-film layer is only one of the components of the proposed connectionist architecture. The overall architecture is based on a thin-film multilayer structure, which is a finite combination of single layers having different optical constants and film thicknesses (Chopra 1969). Multilayer thin-film structures reflect, absorb and transmit light in a much more complicated way, depending on both the construction parameters and the mode of use, such as the wavelength, angle of incidence and polarization. Various mathematical formulations have been

devised to deal with this problem (Liddell 1981; Fuman & Tikhonravov 1992).

Thin-film designs are not always necessarily done by physical experiments. With the aid of modern computing technology, many of the physical experiments with thin films can be simulated on a computer, i.e. rather than observing physical light, it is sufficient to observe numbers generated from computer simulation (similar to neural network models that are biologically inspired and can be simulated on a computer as well). Though some of the simulations of thin-film designs are physically unrealizable, they do provide the designers with physical insight into what the targeted reflectance demands of a multilayer stack and perhaps the extent to which these demands are unrealistic. It is common to program numerically sound algorithms for computing optical effects for thin-film designs. As computers become faster and more powerful, more explorations that demand large amounts of computation have become possible. In this research, we simulate the thin-film multilayer model by the use of thin-film multilayer design algorithms. The software implementation of such a simulation model will be presented in Chapter 6.

3.3 Optical Properties of Thin Films

The usefulness of the optical properties of thin films has been responsible for the immense interest in the study of the science and technology of thin films. The theoretical and experimental studies on the optical behaviour of thin films deal primarily with optical reflection, transmission, and absorption properties, and their relation to the optical constants of films. Consequently, complex multilayer optical-device systems with remarkable reflection, antireflection, interference, and polarization properties have emerged for both laboratory and industrial applications (Chopra 1969).

The processing ability of optics derives from the wavelike properties of light and media which light passes through. The basic properties that characterise a wave are its frequency, wavelength, amplitude, phase, etc. They are defined as follows (Feitelson 1988):

Wavelength: wavelength is the distance the light propagates during one cycle. The wavelength for a given frequency depends upon the speed of propagation. This speed in turn depends upon the medium through which the radiation is propagating.

Index of Refraction: The characteristic of the medium that describes the speed of the light passing through it is called the index of refraction, which is defined as the ratio of the speed of light in vacuum to the speed of the light in material. The refractive index of vacuum is therefore 1.0. For materials which have a higher refractive index, the speed of light in them is proportionately lower.

Amplitude: It is the "height" of the wave.

Phase: It stands for the part of the cycle that the wave is in. Absolute phases are of no interest – only the phase of one wave relative to another is of consequence. A phase difference of $\pi/2$ between two waves means that one of them lags behind the other by one quarter of a cycle.

The value of the velocity of light in a particular medium determines the index of refraction of that medium, and this is an important property characterising the light wave propagation in the medium. When a thin-film multilayer is used for information processing, we can use the wavelength value of a light beam or the indices of refraction of the thin-film materials for carrying information such as input data, and the thicknesses of individual thin-film layers as adjustable variables that help characterise the resulting light wave behaviour of the overall thin-film multilayer structure[1]. We will illustrate that the mapping between the inputs and the resulting light wave behaviour are highly nonlinear in Chapter 4 (see Section 4.3.1). Once the system acquires the necessary input data, these data can then be processed on our simulation model according to algorithms that evaluate the optical effect of the overall structure in an effort to produce desired outputs, such as pre-specified reflectance values.

The Optical Thin-Film Multilayer model can be viewed as a connectionist learning model, because of its unique optical properties and architecture, i.e. it consists of many single thin-film layers that can be viewed as individual processing units like those in conventional connectionist models (see Section 2.2). Interaction or connections among these units occur as a light wave is reflected and transmitted at each boundary of a thin-film layer. In the next chapter, we will present in great detail how this works.

[1]The use of these optical characteristics as system *design parameters* was also discussed in (Liddell 1981: pp.75).

3.4 Application Domain

Various optical applications of single and multilayer films have been developed, such as antireflection coatings, reflection coatings, interference filters and absorptance, thermal emittance of coatings and thin-film polarizers (Chopra 1969). All these applications are mostly specific to the optics domain. However, if we view these applications from the standpoint of connectionist learning models, it can be seen that development of an application here often has similarities with the development of a connectionist computing module that has "learned" some useful information. For example, an Interference Filter can be made to produce high transmission over a limited spectral region of wavelength (see Figure 3.2).

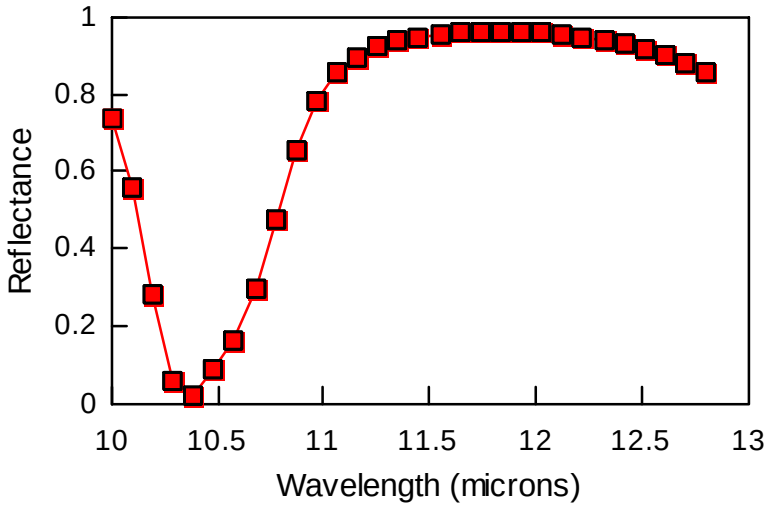


Figure 3.2. An interference filter.

Figure 3.2 shows a non-absorbing thin-film multilayer filter, which only produces higher transmission over wavelengths around 10.4 microns. The required high transmission at the selected wavelength spectrum is normally specified before the thin-film design. Once the desired output, which in this case is the high transmission, is specified over the selected wavelength spectrum, then it is possible to search through the system parameter space to find an optimal set of values in order to match as close as possible to the desired output. This is basically a supervised learning process in a typical connectionist model. Thin-film multilayers can then be made to produce reflectance values according to system parameters which might carry the input

information, at different light waves. If we succeed in solving these two problems: encoding input information (see Section 7.4) and finding the optimal solution according to the specified output (Section 7.2), then it is possible to apply the thin-film multilayer system to a wider range of application domains, such as pattern recognition and classification that are common learning tasks in the field of connectionist models.

3.5 Discussion

This chapter has provided some background concerning the motivation of this research and some information concerning thin films. Researchers in thin film technology typically focus on optical applications and do not usually consider general information processing applications. The research presented in this thesis attempts to fill such a gap by proposing a novel learning architecture based on an algorithm founded on the thin film technology that is in current use in the optical community. In the next chapter, we will give details on how such a learning model works from both the thin film and connectionist perspectives.

Chapter 4
Optical Thin-Film Multilayer Model

4.1 Introduction

As described in the preceding chapter, an optical thin-film multilayer design exhibits desirable properties and structure for building a connectionist model. Such a thin-film multilayer structure can have individual layers that function as simple processing units and "connections" among them which can be characterised by a lightwave being reflected and transmitted through all thin-film layers. By optimizing some system parameters, e.g. layer thicknesses, an optical thin-film multilayer structure can facilitate learning that is typical of connectionist models. From this perspective, we can regard such a thin-film multilayer model as a connectionist model. Hertz, Krogh and Palmer (1991) commented that an optical implementation of a connectionist model would be difficult, as optical system would have the problem of constructing processing units with appropriate nonlinearity, such as threshold behaviour. Although a single thin-film layer does not possess a threshold, it demonstrates high nonlinearity (Baldwin 1969; Dobrowolski & Kemp 1990; Liddell 1981; Apfel 1972), which is an important criterion for choosing such an optical thin-film multilayer architecture as a potential candidate in our research for developing a novel connectionist learning model.

In this chapter we propose a novel connectionist learning model based on an *Optical Thin-Film Multilayer* model (OTFM). We describe the core processing of the OTFM here. Other issues, such as optimization, the learning procedures and the manners of applying input, etc., are described in Chapter 5 and 7.

First, properties of a single thin-film layer are examined and their calculations are described in detail. In the following section we then describe a thin-film multilayer structure and its properties, which forms the basis for the proposed connectionist model. The thin-film multilayer shows more complex behaviour than that of a single thin-film. These properties are fundamental to the OTFM architecture.

To be computationally cost-effective, efficient computational algorithmic procedures are also introduced, which are particularly important when dealing with large complicated calculations. At the end of the chapter, the OTFM is compared with neural network models in various aspects.

4.2 Assumptions

It must be pointed out that the proposed thin-film models are based on the assumptions that a thin film is a homogeneous, isotropic, and plane-parallel plate with a single thickness. Multiple layers of real physical single thin-films may be more complex (because of the real anisotropics and inhomogeneities), but these differences between the real behaviour and that predicted in the ideal model are usually negligible. In the proposed thin-film model, we assume incident light is perpendicular to the surface of the multilayer structure, and it is sufficient to consider only the electric field vector (and ignore the magnetic field vector component in the calculations). The phase of the wave is customarily incorporated into the description by representing the component as having a complex amplitude. The learning considered in the present work is supervised learning.

4.3 Single Thin-Film Layer

Before describing the multilayer model, a single thin-film layer is examined first. We assume that a thin film is a homogeneous, isotropic, and plane-parallel plate with thickness d and is characterized by a refractive index n. If absorption occurs, the refractive index is given by $(n + ik)$, where k is the imaginary component of the complex refractive index that accounts for any absorption of the thin-film material.

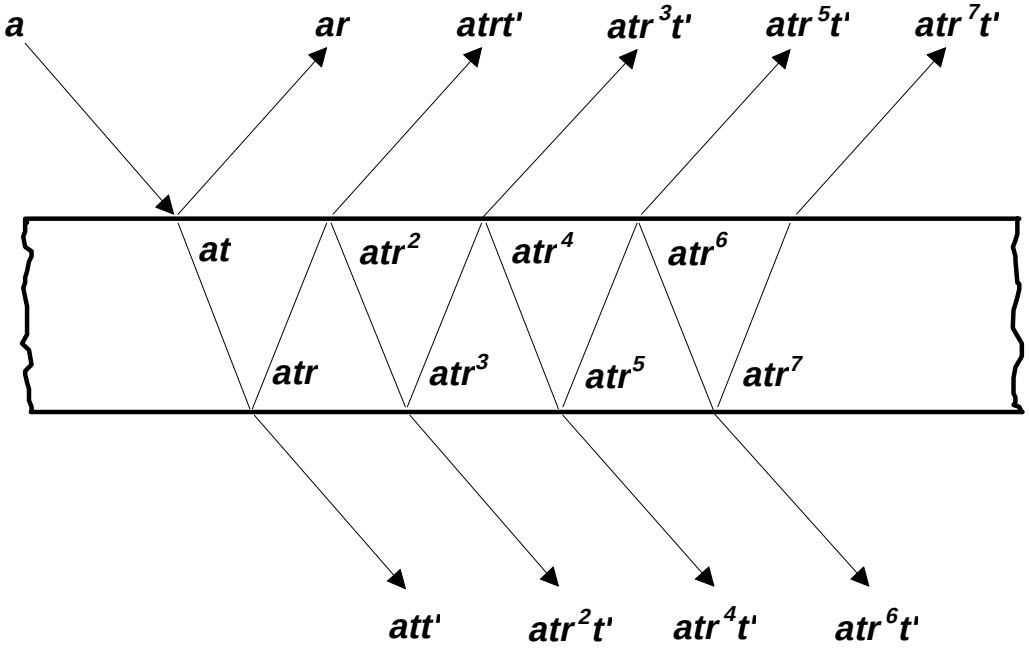


Figure 4.1. Multiple reflections in a thin-film layer.

Consider an incident of a light beam of a single wavelength (monochromatic) λ on such a plane-parallel plate of transparent material as shown in Figure 4.1. The light beam is then divided into two parts, a reflected beam and a transmitted beam which enters the material. When the transmitted beam in the material reaches the second surface, it is again split into a refracted beam and a reflected beam. The reflected beam goes back to the first surface, and this process of splitting of the reflected beam continues as shown in Figure 4.1. The angles and magnitudes of these beams are determined by the polarization and wavelength of light, the angle of incidence of the beam with the plate, and the index of refraction of the plate material (Born & Wolf, 1980).

The reflected beams *ar*, *atrt'*, atr^3t', etc., differ from each other in both amplitude and phase. The overall reflectance of the light beam by the plate is determined by the multiple-beam interference of these various reflected component waves (Born & Wolf 1980). If the incident light beam is perpendicular to the plate surface, the polarization ceases to be a factor, and the overall reflectance is determined solely by the refractive index and thickness of the material and the wavelength of the light.

4.3.1 Reflection and Transmission Coefficients of a Single Thin-Film Layer

With the assumptions made in Section 4.2, considering an incident light wave perpendicular to the film surface (Figure 4.2), we can first derive expressions for calculations of reflection and transmission coefficients of a single thin-film layer.

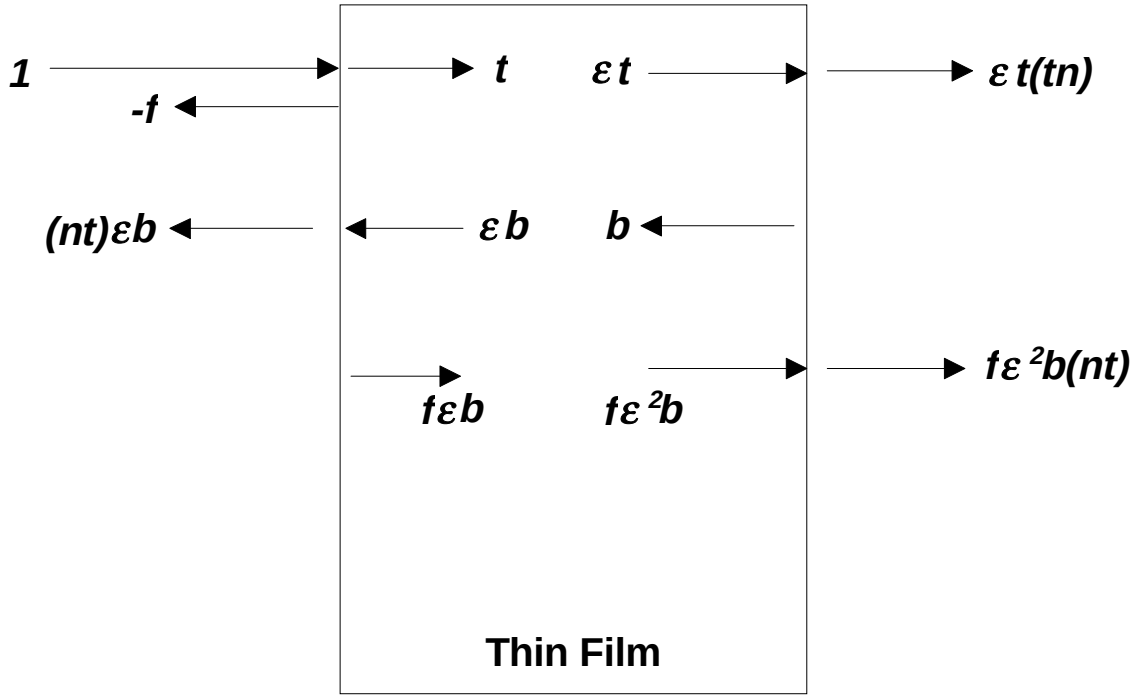


Figure 4.2. A single thin-film with light incident (**1**) on the left.

As shown in Figure 4.2, a monochromatic light beam with wavelength λ is incident on the left side of the thin-film layer. The total amplitude of incident light is represented by 1, and it is sufficient to consider only the electric field vector (see Section 4.2). The incident beam may be thought to be divided at the left surface of the thin-film into a transmitted component and reflected component, with amplitudes $-\boldsymbol{f}$ and $\boldsymbol{t}$ (i.e. the Fresnel coefficients of reflection and transmission, see equation (3.1), and in this case, $n_2 = \tilde{\boldsymbol{n}}$, $n_1 = 1$), respectively, where

$$f = \frac{\tilde{n} - 1}{\tilde{n} + 1} \qquad \text{and} \qquad t = \frac{2}{\tilde{n} + 1} \tag{4.1}$$

$\tilde{n}$ is the complex refractive index of the thin-film, and $\tilde{n} = n + ik$, where n is the ordinary index of refraction and k is the attenuation coefficient. The transmitted beam component with amplitude t now undergoes a phase shift as it travels a distance d (the thickness of the film) up to the right hand surface of the film, so the phase shifted amplitude of the transmitted beam becomes εt, where

$$\varepsilon = e^{i\frac{2\pi\tilde{n}d}{\lambda}} \tag{4.2}$$

At the right-hand boundary of the film, the beam is again divided up into transmitted and reflected components. The transmitted component that goes off the film at the right-hand surface has an amplitude of $\varepsilon t(tn)$. As the reflected component is one of an infinite series of reflected components at this interface, they can be combined together to represent the interference combination of all of the reflected components by b in Figure 4.2; εb is the phase shifted result of this leftward-travelling wave at the inside left-hand surface of the film. At the left hand surface, the wave with amplitude εb is split into a refracted (transmitted) component with amplitude $nt\varepsilon b$ and a reflected component travelling back to the right with amplitude $f\varepsilon b$. At the right-hand surface, the phase-shifted rightward travelling wave, here with amplitude $f\varepsilon^2 b$, is also split into two components: a reflected component already accounted for by b and a transmitted component with amplitude $f\varepsilon^2 bnt$.

Expressions for the reflection and transmission coefficients of the thin-film can now be taken directly from Figure 4.2:

$$r_f = -f + nt\varepsilon b \qquad \text{and} \qquad t_f = (\varepsilon t + f\varepsilon^2 b)nt \tag{4.3}$$

Eliminating the unknown b from equation (4.3), the complex reflection and transmission coefficients can be given in terms of the refractive index and thickness of the film and the wavelength of the incident light (using equations (4.1) and (4.2)):

$$r_f = \frac{-f + \varepsilon^2 f}{1 - f^2\varepsilon^2} \tag{4.4}$$

and

$$t_f = \frac{\varepsilon(1 - f^2)}{1 - f^2\varepsilon^2} \tag{4.5}$$

From the perspective of connectionist models, a single thin-film layer can be used as a simple processing unit in a conventional connectionist model, and the calculation of its reflection or transmission coefficients, as shown by equations (4.4) and (4.5), can be seen as its activation function with the reflection or transmission coefficients as its outputs.

Note that if the input is encoded into $\tilde{n}$ or λ, ε is nonlinear (equation (4.2)). As a result, the output of a thin-film layer, reflection coefficient r_f and the transmission coefficient t_f, are highly nonlinear (equations (4.4) and (4.5)). Nonlinearity is an important property for a processing unit (a layer) in a connectionist model and is necessary for the model to carry out more complex computational tasks.

If a second layer is added next to the first one as shown in Figure 4.2, r_f will change to some other value, which cannot be calculated from equation (4.4), since the light wave at the incident boundary of the first layer will no longer be 1.0, and it will change to the total amount of light transmitted through the added layer. A similar change applies to t_f as well. Thus whenever more layers are added, or the thickness of an existing layer is varied, r_f and t_f of each layer will change accordingly. Contributions from the infinite multiple reflection and transmissions for multiple layers can be calculated in a recursive manner which will be described in Section 4.4.1. The final outputs, overall *r* and *t* of the multiple layer structure, are highly nonlinear.

4.3.2 Visual Representation of the Reflectance of a Single Thin-Film Layer

Suppose we have a thin-film layer suspended in a vacuum, with refractive index 2.4 and thickness close to 0.0. If we increase the layer thickness steadily, we can see how the complex reflection coefficient of a single thin-film layer changes with increasing thickness.

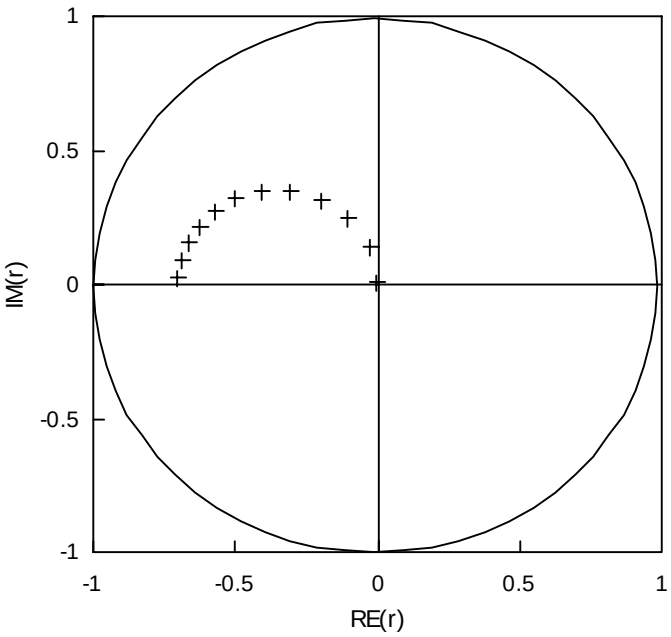


Figure 4.3. Complex reflection coefficient of a thin-film layer changes when its layer thickness is increased.

As shown in Figure 4.3, the reflection coefficient traces out a half-circle within the unit circle (complex reflection coefficients are always within this unit circle with radius 1.0). The reflectance value (i.e. the square of the absolute value of the reflection coefficient) reaches a maximum value when the circle crosses the real axis, which corresponds to a film optical thickness of one-quarter the wavelength of interest[1]. When the optical thickness is increased to one-half wavelength, the complex reflection coefficient returns to the starting point (see Figure 4.4), which indicates that the reflection coefficient for a given wavelength of light is invariant with respect to a change in $n_i d_i$ by the amount $\lambda/2$. In this case, the reflectance becomes 0.0, and the reflection coefficient is located at the origin of the complex reflection coefficient plane.

[1]An optical thickness is equal to $n_i d_i$ (*i-th* layer in a multilayer stack), so a film optical thickness of one-quarter the wavelength means that $n_i d_i$ is equal to $\lambda/4$.

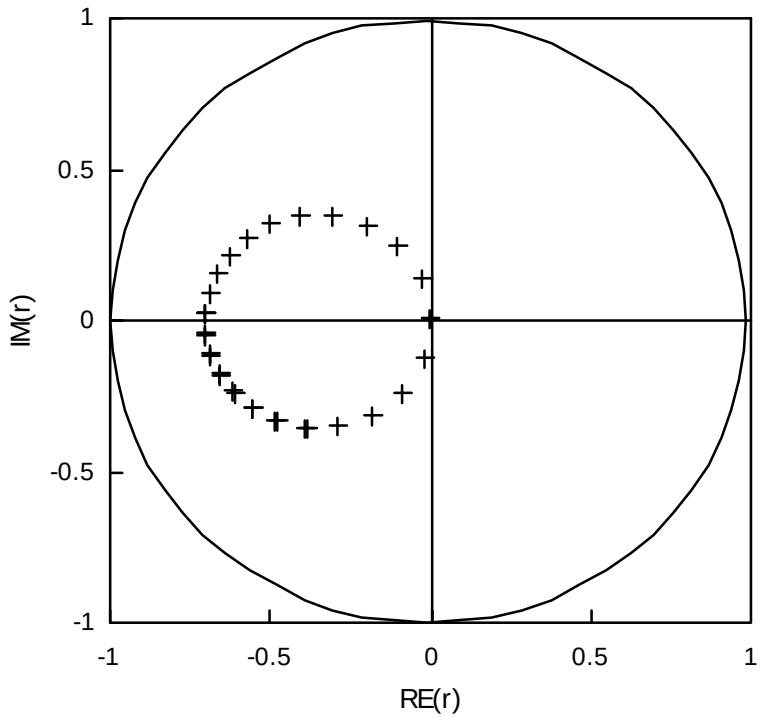


Figure 4.4. The reflection coefficient returns to the starting point when the layer thickness is increased to one-half wavelength.

The reflection coefficient of a single thin-film layer shows an interesting regularity with respect to its nonlinear association with thickness value, wavelength and refractive index. If somehow we could manipulate such behaviour, for example to force the reflection coefficient point to reach a specified target point (i.e. a reflection coefficient point) somewhere in the unit circle shown above, we might be able to achieve certain kind of learning. However the reflection coefficient point of a single thin-film might not be so easily made to reach a specified target. It only goes around a single circle. In the next section we describe the multiple thin-film layer structure which demonstrates much more complex behaviour that can be used for learning.

4.4 Multiple Thin-Film Layer Structure

The preceding section has shown that the reflectance of a single layer responds to increasing thicknesses in an interesting manner. What if we use a thin-film multilayer instead of a single one? We answer such a question in this section and demonstrate that a multilayer structure exhibits much more complex optical properties compared with a single layer.

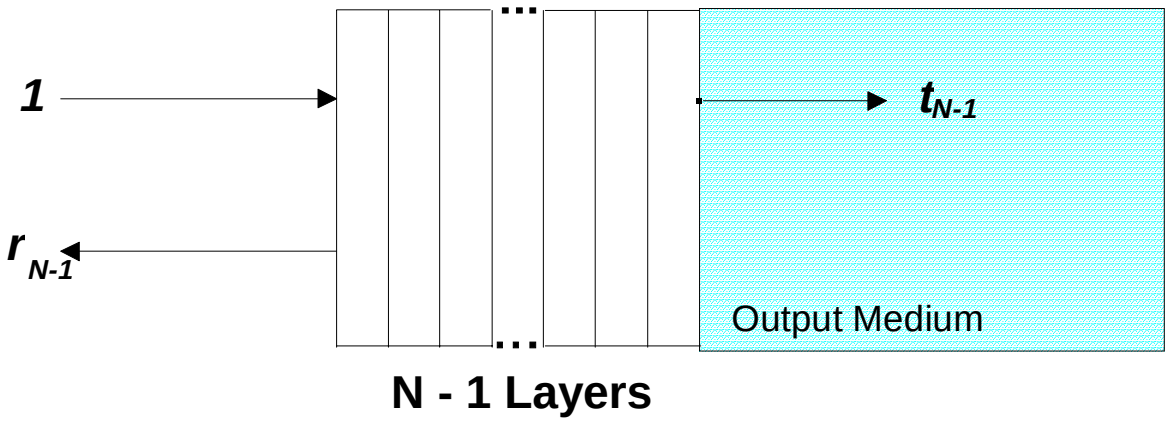


Figure 4.5. Multilayer thin-film stack of *N - 1* layers.

Suppose several thin-films are deposited one on top of each other to construct a thin-film multilayer, and consider a monochromatic light beam, with wavelength λ, incident onto such a multilayer. Figure 4.5 shows the structure with *N - 1* layers. 1.0 represents the total amount of light. Let r_{N-1} denote the (complex) reflection coefficient for the electric field vector in vacuum at the vacuum-multilayer interface, and t_{N-1} denote the transmission coefficient in the output medium at the last boundary surface. The values of these coefficients depend on the refractive indices and thicknesses of all the films in the structure, the refractive index of the output medium, and the wavelength of light.

4.4.1 Reflection and Transmission Coefficients of a Thin-Film Multilayer Structure

Suppose another thin-film layer, with a complex refractive index of $\tilde{n}_N = n_N + ik_N$ and thickness d_N is added to the existing *N - 1* layer structure. The added thin-film layer will be the *Nth* layer. The distance between the *Nth* layer and *N - 1* layer substructure will be zero, however the layers are shown separated in Figure 4.6 for the purpose of explanation.

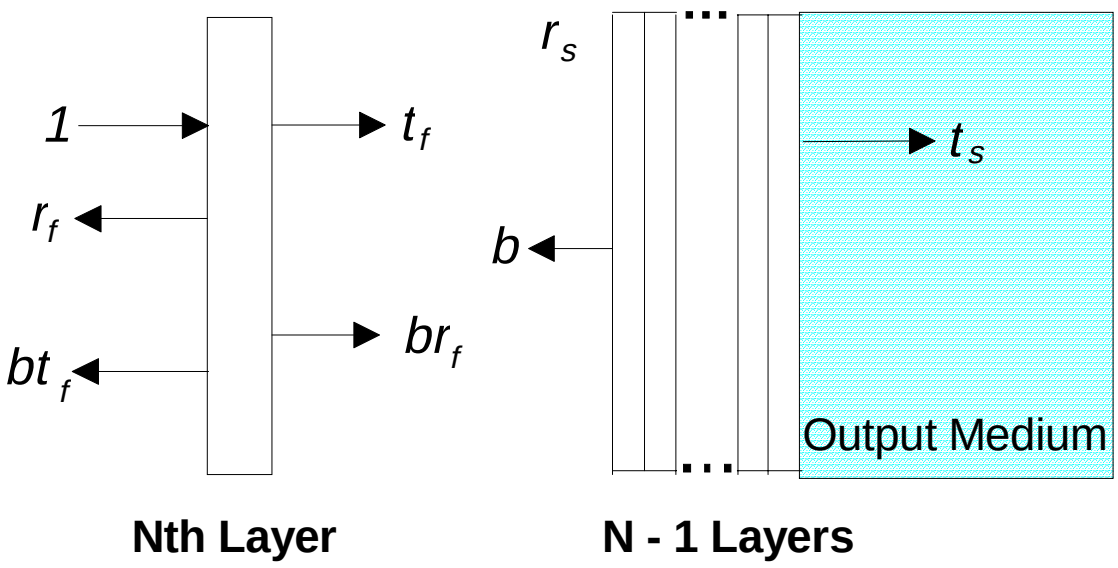


Figure 4.6. A layer is added to an existing multilayer structure (substructure).

In Figure 4.6, A thin-film layer is added to an existing multilayer part, the substructure. This layer is considered to be isolated (i.e. surrounded by vacuum). r_f and t_f denote the reflection and transmission coefficients of this single layer. r_s and t_s denote the reflection and transmission coefficients off the *N - 1* layer substructure. Since the light incident on the *N - 1* layer substructure does not have an amplitude of 1.0, the amplitude of light reflected off its front surface is not its reflection coefficient r_s, but some other value, which in this case is denoted by ***b***. Therefore $\boldsymbol{bt_f}$ is the component of the reflected amplitude of light off the single layer resulting from ***b***, and $\boldsymbol{br_f}$ is the component of the transmitted amplitude of light through the single layer, resulting from ***b***.

The overall reflection *r* and transmission coefficients *t* can be expressed by (***b*** is still unknown):

$$r = r_f + t_f\, b \tag{4.6}$$

and

$$t = (t_f + r_f\, b)\, t_s \tag{4.7}$$

For "self-consistency", the following relation must hold:

$$(t_f + r_f\, b)\, r_s = b \tag{4.8}$$

Eliminating ***b*** from the above expressions, *r* and *t* of the *N*-layer structure can be obtained:

$$r = \frac{r_f + (t_f^2 - r_f^2)\, r_s}{1 - r_f\, r_s} \tag{4.9}$$

and

$$t = \frac{t_f\, t_s}{1 - r_f\, r_s} \tag{4.10}$$

Then substituting in the equations (4.9) and (4.10) for r_f and t_f from equations (4.4) and (4.5) yields the following expressions for calculating the thin-film multilayer reflection and transmission coefficients:

$$r_N = \frac{-f_N(1-\varepsilon_N^2) + (\varepsilon_N^2 - f_N^2) r_{N-1}}{(1-f_N^2\varepsilon_N^2) + f_N(1-\varepsilon_N^2) r_{N-1}} \tag{4.11}$$

$$t_N = \frac{\tilde{n}_N \tau_N^2 \varepsilon_N}{(1 - f_N^2 \varepsilon_N^2) + f_N (1 - \varepsilon_N^2) r_{N-1}} \tag{4.12}$$

where

$$f_N = \frac{\tilde{n}_N - 1}{\tilde{n}_N + 1} \quad , \quad \tau_N = \frac{2}{\tilde{n}_N + 1} \quad and \quad \varepsilon_N = \exp^{\frac{i2\pi\tilde{n}_N d_N}{\lambda}} \tag{4.13}$$

Note that the factor ε_N accounts for both phase shift and amplitude attenuation due to light propagation. The Fresnel coefficients f_N and τ_N are appropriate for a boundary between material of refractive index $\tilde{n}_N$ and vacuum. The recursive expressions shown in equations (4.11) and (4.12) provide the means for computing the reflection and transmission coefficients for a thin-film stack of any number of layers. If $\tilde{n}$ of each layer is used as an encoded input, by choosing an appropriate value of d for each layer, and different wavelengths for the multilayer model, almost any kind of nonlinear mapping between the inputs (encoded into many such $\tilde{n}$) and the outputs (many r_N or t_N) can be accomplished. It is these highly nonlinear expressions that form the basis for the proposed connectionist computation.

Compared with conventional connectionist models, such as neural networks, the thin-film multilayer system has its unique properties and architecture. It consists of many individual thin-film layers, which can be considered as basic processing units of a connectionist model. The interactions which occur through a light wave being reflected and transmitted at the boundaries of each layer can be viewed as the "connections" among all the units.

4.4.2 Visual Representation of the Reflectance of a Thin-Film Multilayer Structure

In this section we provide an example to illustrate how the reflection coefficient of a thin-film multilayer structure changes when each layer thickness is increased. Assume we have a 3-layer stack surrounded by air. The refractive index of the first and third layer is 2.5, and the refractive index of the second layer is 4.1. All the initial layer thicknesses are set to 0.0. The wavelength of the light incident is 10.0μm. We start varying layer thickness from the first layer, ranging from

0.001 to 1.0, then the second and the third layer. Figure 4.7 shows how the changes occur in the overall reflection coefficient of the 3-layer stack.

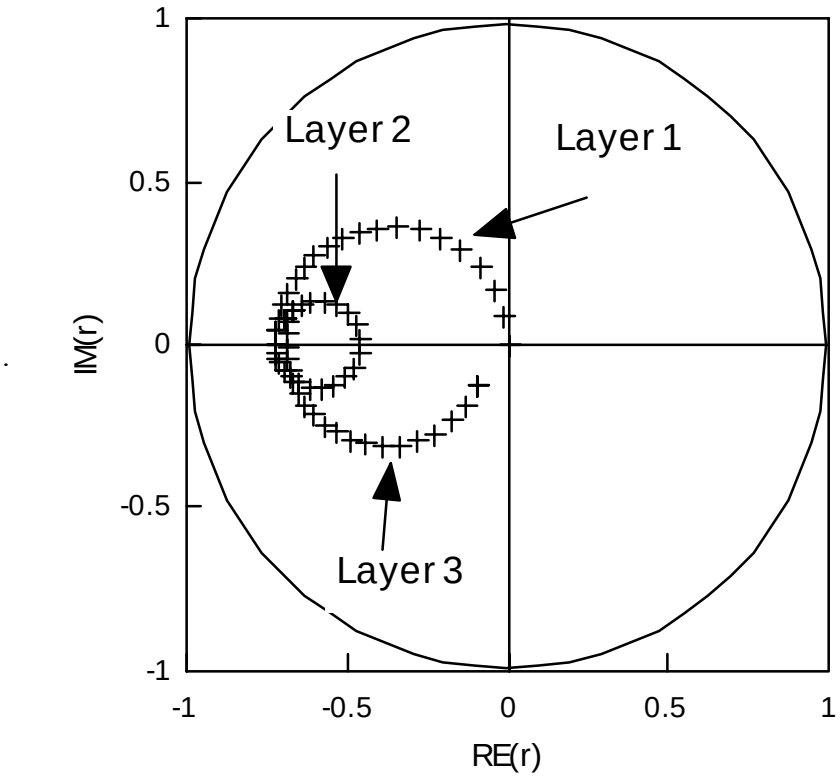


Figure 4.7. Reflection coefficient of a 3-layer thin-film stack.

The reflectance changes in the counter-clockwise direction. When the first layer thickness is increased from 0.001 to 1.0, the corresponding reflection coefficient changes gradually from the origin of the complex plane to the other end of a half-circle labelled as "Layer 1". When the second layer thickness is changed, from 0.001 to 1.0, we can see that the reflection coefficient point changes from where the it ends with "Layer 1" and carries on until it traces out a full circle, coming back where it starts (note that since the second layer has a higher refractive index, the diameter of the circle is smaller than that of the other two). Then again when the third layer thickness is changed from 0.001 to 1.0, the reflection coefficient point changes from where it ends with "Layer 2" and traces out another half-circle. This process will continue as any thickness of a layer changes. If building an anti-reflectance thin film structure is our goal, then we can carry out this learning task by training the model to move its reflection coefficient points towards the origin, since the reflectance is 0.0 when the reflection coefficient point is at the origin. Such a

training is always associated with searching through many sets of thicknesses and eventually finding one that produces the desired reflection coefficients as the outputs. In fact this kind of operation can be applied to any points specified by the model designer within the unit circle (complex reflection plane), not just the origin of the complex plane.

4.5 Efficient Computational Algorithmic Procedures

Note that expressions (4.11) and (4.12) are recursive for calculation of reflection and transmission coefficients. Using r_{N-1} and t_{N-1} from the previous $N - 1$ layer substructure, we can obtain r_N and t_N for the N layer structure that includes the $N - 1$ layer substructure and the newly added layer. To achieve some kind of nonlinear mapping between the encoded inputs (i.e. input data encoded into many $\tilde{n}$) and the outputs (i.e. many r_N or t_N), we need to search through a multi-dimensional space of thickness values to get an appropriate set of thicknesses by using an optimization algorithm.

Searching for such a set of appropriate thicknesses (i.e. d_N of each layer) is the actual learning process of the mapping between the inputs and the outputs. It involves adjusting the individual layer thicknesses and then calculating the multilayer reflectance for each new configuration. However, equation (4.11) only provides a means to calculate the multilayer reflectance r_N by first calculating $r_1, r_2, r_3, ..., r_{N-1}$. If a layer is adjusted, to calculate the new overall multilayer reflectance r_N, all the $r_1, r_2, r_3, ..., r_{N-1}$ must be recalculated. If more layers are adjusted, the computation cost is even more expensive. Thus it is desirable to produce optimized algorithms that reduce the amount of computation for these thin-film multilayer calculation. The next two sections will introduce two efficient algorithmic routines to substantially reduce the computation cost required by reflection and transmission calculations described in Section 4.4.1.

4.5.1 Procedure #1

The first algorithmic routine was developed by Purvis (1982), which takes advantage of the values of some intermediate parameters of previously evaluated multilayer calculations, that can be retained for subsequent use. This method works in the following way, consider a multilayer (instead of a single layer) added to the substructure (Figure 4.8).

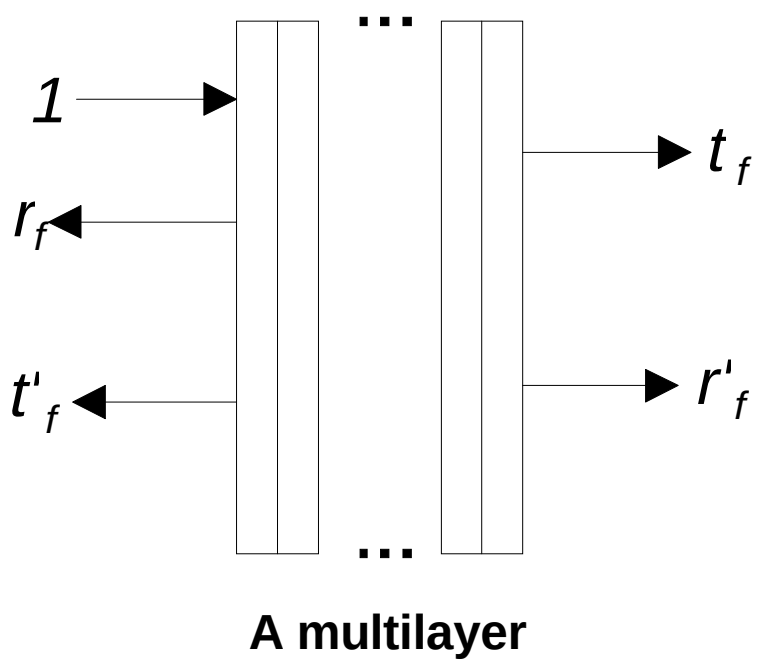


Figure 4.8. Reflection and transmission coefficients of a multilayer, r'_f and t'_f from the opposite direction.

As shown in Figure 4.8, for a multilayer structure, r_f' is the reflection coefficient from the other side of the multilayer (opposite to r_f) . Note that r_f' is equal to r_f in the case of a single layer, but for a multilayer they are not equal. t_f and t_f' are always equal in either case. We can replace equation (4.7) with (4.14) and (4.8) with (4.15) respectively in the case of adding a multilayer (instead of a single layer) to an existing multilayer structure (see Figure 4.6):

$$t = (t_f + r_f' b)t_s \tag{4.14}$$

and

$$(t_f + r_f' b)r_s = b \tag{4.15}$$

then eliminating ***b*** from the above expressions and equations (4.6) and (4.7), r and t of the new combined multilayer structure can be obtained:

$$r = \frac{r_f + (t_f^2 - r_f r_f')r_s}{1 - r_f' r_s} \tag{4.16}$$

and

$$t = \frac{t_f t_s}{1 - r_f' r_s} \tag{4.17}$$

When one multilayer part is added to another one, to calculate the overall combined multilayer reflection coefficient by using equation (4.16), we only need to know reflection and transmission coefficients from both sides of each multilayer. From there, a third or several multilayers can be added to this combined multilayer, and the overall *r* of the new combined multilayer can be easily obtained from equation (4.16). Thus equation (4.16) provides a means of faster computation of the reflection coefficient on many combined multilayers without resort to equation (4.11).

To illustrate this approach, consider the multilayer as divided into three parts: an underlying part (fixed), a varying part and an overlaying part (fixed) as illustrated in Figure 4.9.

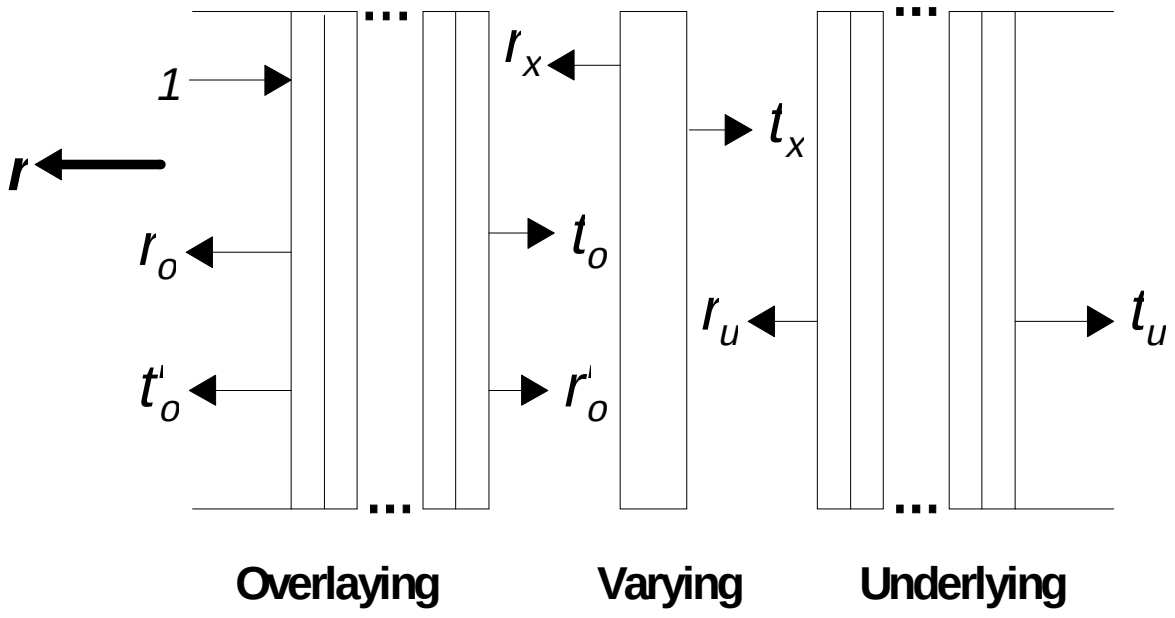


Figure 4.9. The multilayer is divided into three parts, an overlaying part, a varying part and an underlying part.

In Figure 4.9, the three divided parts are considered isolated (i.e. surrounded by vacuum). r_0 and t_0 represent the reflection and transmission coefficients of the overlaying part (fixed). r_u and t_u denote the reflection and transmission coefficients of the underlying part (fixed). r_x and t_x denote the reflection and transmission coefficients of the varying part. r denotes the overall multilayer reflection coefficient.

The underlying and overlaying parts are fixed, and usually consist of many layers. The varying part is a layer which is required to be adjusted as learning proceeds. Now vary the layer thickness of the varying part, and it is easy to obtain r_x and t_x using equations (4.4) and (4.5). Assume that r_0, t_0, r_u and t_u are known, then the new overall multilayer reflection coefficient r can be calculated using equation (4.16). Different thicknesses can be tried on the varying layer, and corresponding to a chosen thickness, a new overall reflection coefficient r can be always obtained according to equation (4.16). In Figure 4.9, equation (4.16) is used in two stages: first the varying layer is added to the underlying layer (r_x combined with r_u), then this combination is added to the overlaying part. Thus equations (4.16) and (4.17) provide a means of faster computation of the overall reflection and transmission coefficients of many combined multilayers.

After we have tried enough various thicknesses on the varying part, a different layer is chosen to be varied for continuous calculation, so the current varying part is added to the underlying part, while at the same time, a layer is taken off the overlaying part and becomes the new varying part as shown in Figure 4.10.

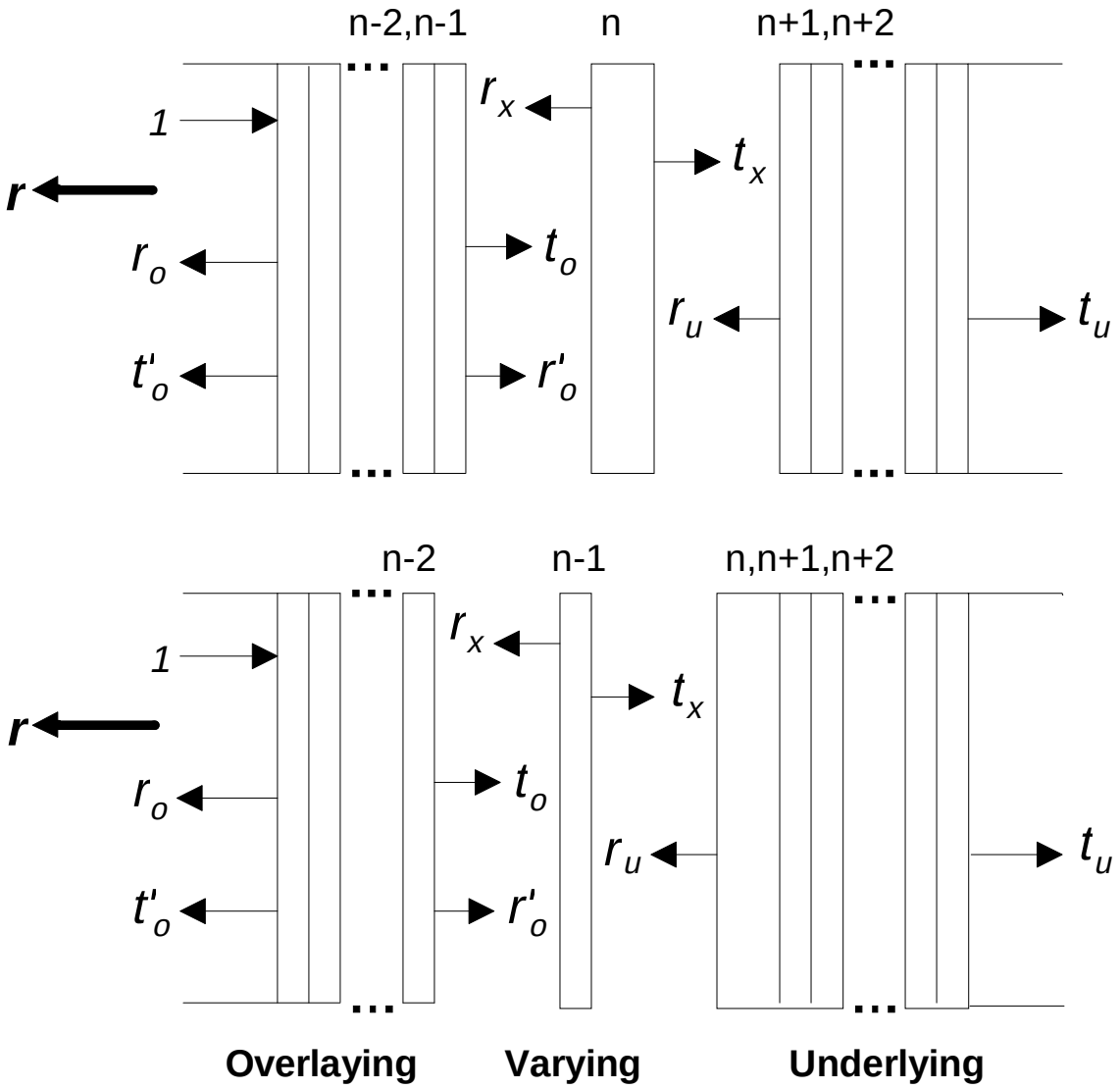


Figure 4.10. The current varying part is added to the underlying part, and at the same time, a layer is taken off the overlaying part and becomes the new varying part.

When a layer is taken off the overlaying part, it can be considered to be a layer with a negative thickness that is added to the multilayer, so by using equations (4.16) and (4.17) we can calculate the new r_u and t_u of the new underlying part (see Figure 4.10) after the current varying layer is added the underlying part, and in the same manner we can get the new r_o and t_o of the new overlaying part after one layer is taken off the overlaying part. Then the thickness of the new varying layer (i.e. the one taken off the overlaying part) is varied, and the new r_x and t_x can be easily obtained by equations (4.4) and (4.5). The new overall reflection coefficient r can be obtained by using equation (4.16) again. One continues to vary the layer thickness of the varying

part, and obtain the new overall reflection coefficient *r* accordingly. How many times the thickness of the varying layer is being varied and how it is varied are determined by the model designer or maybe a specific optimization method. The above procedure continues in as many iterations as desired. Eventually all the layers in the multilayer stack might be varied, and corresponding calculations of *r* and *t* are achieved by using equations (4.16) and (4.17), instead of (4.11) and (4.12) each time, which might result in very expensive computation.

In this research the above routine has been used in conjunction with a search algorithm called the N-squared Scan Method (see detail in Section 5.3.1).

4.5.2 Procedure #2

The second routine (Case 1983) aims to save computation by reducing the number of multiplications and divisions required in equations (4.11) and (4.12), by deriving some intermediate parameters, as it can be critical for large complex computation in terms of efficiency.

Figure 4.11 illustrates how this method works. The steps shown in Figure 4.11 indicate successive conceptual stages when a new thin-film layer is added to an existing multilayer thin-film structure.

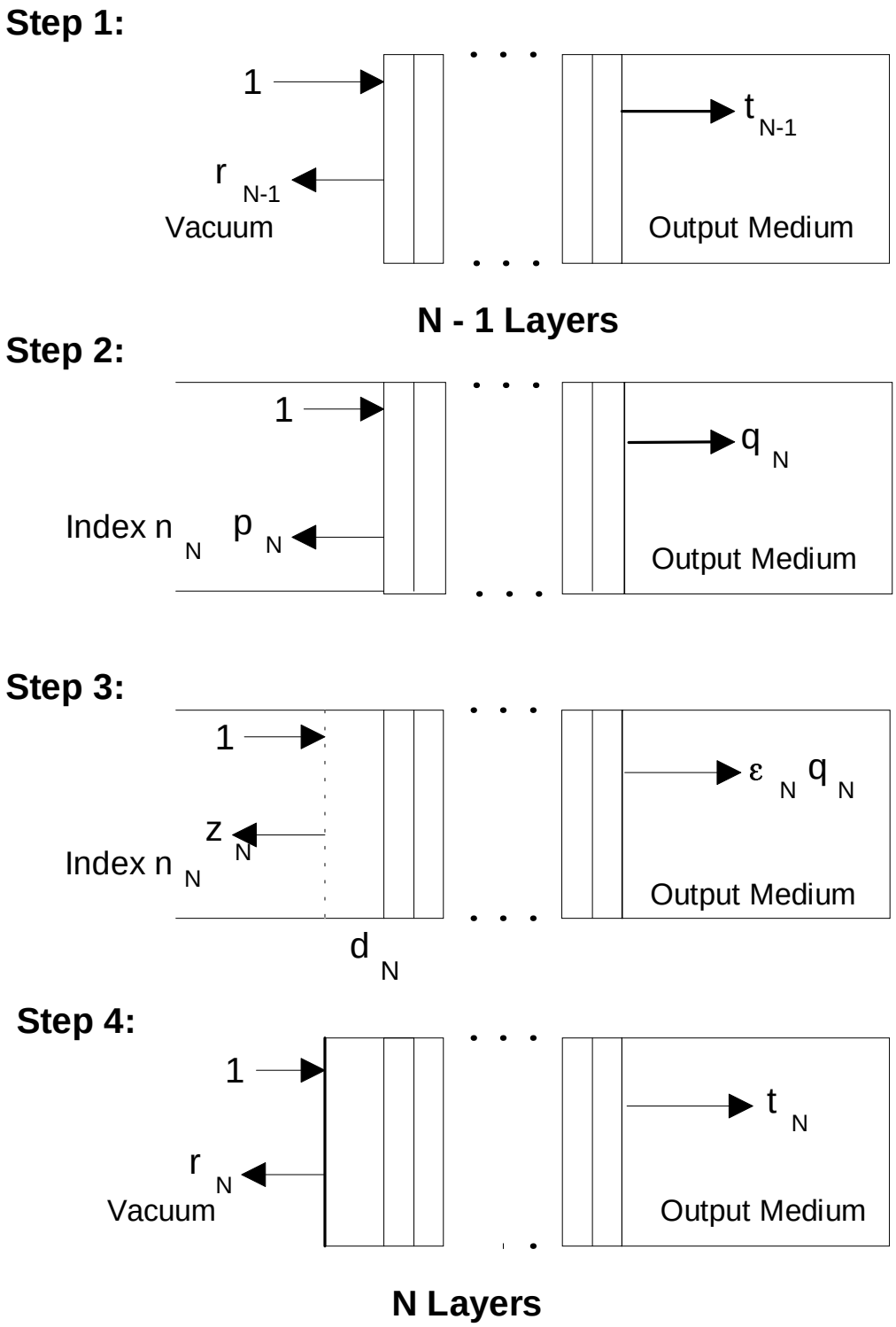


Figure 4.11. Four steps illustrating the addition of one layer to a *N - 1* layer structure.

Step 1 shows the *N - 1* layer structure. In Step 2, the refractive index of the input medium has been changed from vacuum to *n*, with corresponding changes in the reflection coefficient from r_{N-1} to p_N and transmission coefficient from t_{N-1} to q_N. p_N can be derived as shown in Figure 4.12.

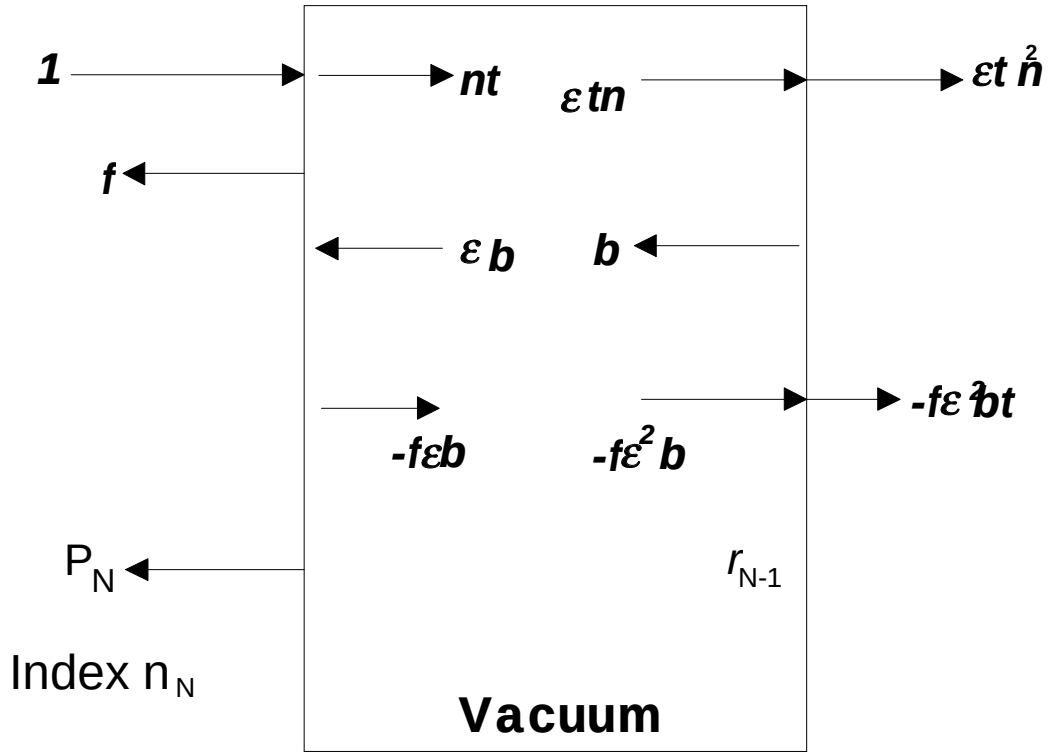


Figure 4.12. Derivation of p_N.

In Figure 4.12, we assume that in Step 2, there is a "vacuum gap" between the newly-added layer with refractive index n_N and the *N - 1* layers. The thickness of this vacuum gap is really zero, but it is shown here for the purpose of derivation. Figure 4.12 is similar to Figure 4.2, where ***b*** denotes the interference combination of all of the reflected components. The initial reflected component on the left side of the Figure 4.12 (indicated by ***f)*** is positive because the light incident goes from within the newly-added layer, which has higher refractive index, to vacuum (see also equation 4.1). Thus we can calculate the overall reflection coefficient p_N at the inside surface of the newly-added layer by the following expressions:

$$b = (nt\varepsilon - f\varepsilon^2 b)r_{N-1} \tag{4.18}$$

and

$$p_N = f + t\varepsilon b \tag{4.19}$$

Eliminating b, in favour of p_N, and because ε in the vacuum gap is 1.0 (because the thickness of the gap is really zero), we then have

$$p_N = \frac{f + (f^2 + nt^2)\, r_{N-1}}{1 + f\, r_{N-1}} \tag{4.20}$$

Since we can easily prove that $f^2 + nt^2$ here is equal to 1.0 from equation (4.1) (note that f is in fact f_N of the newly-added layer), we can obtain p_N as follows:

$$p_N = \frac{f_N + r_{N-1}}{1 + f_N\, r_{N-1}} \tag{4.21}$$

Figure 4.12 also shows how to derive q_N as follows:

$$t_x = nt^2\varepsilon - f\varepsilon^2 tb \tag{4.22}$$

t_x is the reflection coefficient at the right-hand surface, across the boundary within the next medium. From equation (4.18) we can obtain b. And again considering that ε is 1.0 in the vacuum gap, we replace the b in equation (4.22) and obtain an expression which associates t and t_x by a scaling factor as shown in the following

$$t_x = \frac{nt}{1 + f\, r_{N-1}}\, t \tag{4.23}$$

Since for each layer of the multilayer, t_x is reduced by the above scaling factor of t, therefore we can eventually obtain q_N (by equation (4.23)) at the right-hand surface of the multilayer within the output media (see Step 2 in Figure 4.11), by replacing f with f_N, t with τ_N and t_{N-1} (τ_N and t_{N-1} are the reflection coefficients on the left-hand and the right-hand surfaces of the multilayer respectively), and t_x with q_N:

$$q_N = \frac{n_N \tau_N\, t_{N-1}}{1 + f_N\, r_{N-1}} \tag{4.24}$$

We can rewrite equations (4.11) and (4.12) into the following:

$$r_N = \frac{-f_N + z_N}{1 - f_N\, z_N} \tag{4.25}$$

and

$$t_N = \frac{\tau_N\, \varepsilon_N\, q_N}{1 - f_N\, z_N} \tag{4.26}$$

where

$$z_N = \varepsilon_N 2\, p_N \tag{4.27}$$

Step 3 shows an imaginary boundary separating the new material as the input medium and the same material as a layer with thickness d_N. z_N is the reflection (ratio of wave amplitudes) at this boundary, differing from p_N by a factor associated with light propagation over a distance of $2d_N$, i.e., ε_N^2. Since the light incident 1 is now at the imaginary boundary, when it first travels to the

right-hand boundary of the newly-added layer, we can obtain a reflection coefficient which is $\varepsilon_N p_N$ here, and when it travels back to the imaginary boundary, the reflection coefficient becomes $\varepsilon_N^2 p_N$. The transmission coefficient relative to the new input boundary is now $\varepsilon_N q_N$.

The final step is to change the refractive index of the input medium from $\tilde{n}_N$ back to unity and transformation of z_N into r_N and q_N into t_N. Note that Step 3 to Step 4 is the inverse operation of Step 1 to Step 2 (see Figure 4.11). This observation allows a concise derivation of equations (4.25) and (4.26) by replacement of variables, e.g. substituting z_N for p_N and r_N for r_{N-1} in equation (4.21), which is

$$z_N = \frac{f_N + r_N}{1 + f_N r_N} \tag{4.28}$$

Equation (4.28) is exactly the same as (4.25).

Recursive patterns are found now for z and t coefficients without resort to intermediate steps. Since $z_N = \varepsilon_N^2 p_N$ (equation (4.27)), the z pattern can be obtained by rewriting equation (4.25) for r_{N-1} and substituting it in equation (4.21) in favour of p_N,

$$p_N = \frac{\alpha_N + z_{N-1}}{1 + \alpha_N z_{N-1}} \tag{4.29}$$

where the Fresnel coefficients of adjacent layers combine as

$$\alpha_N = \frac{f_N - f_{N-1}}{1 - f_N f_{N-1}} \tag{4.30}$$

The t pattern is obtained from combining equations (4.24) and (4.26) and eliminating r_{N-1}, in favour of p_N:

$$t_N = \frac{\varepsilon_N(1 - f_N\, p_N)t_{N-1}}{1 - f_N\, z_N} \tag{4.31}$$

We have adopted the above method in our simulation model (see Chapter 6). By doing so, we can eliminate the direct need of r_{N-1} for the calculation of r_N. Intermediate steps in the calculation only require successive evaluations of z_1, z_2, z_3, ..., etc. Then equation (4.25) is used to obtain z_N. With these intermediate variables, e.g. z_N, p_N and α_N, these transformation expressions provide fast calculation of thin-film multilayer properties, as they require fewer multiplications and divisions (from *12* in equation (4.11) to *8* in equations (4.25), (4.27), (4.29) and (4.30)), that is of particular importance for large complex calculations.

The thickness of each layer, d_i, is contained in the value of ε_N for that layer (see equation (4.13)). If the thicknesses of individual layers in the multilayer structure are regarded as adjustable parameters, then equations (4.29) and (4.30) provide a highly nonlinear mapping of the reflection coefficient r_N as a function of the thickness parameter d_i, $i = 1, ..., N$.

4.6 Comparison with Neural Network Models

In general, the task of a connectionist model is to learn certain nonlinear mapping between its inputs and outputs. In neural network models, this can be achieved by adjusting the weights of connections among processing units. Compared with neural networks and cellular automata described in Chapter 2, the OTFM has unique characteristics and architecture. The thickness of each layer can be considered as a weighted connection (from the perspective of a dynamical system approach, it is merely an adjustable parameter). By searching for a set of appropriate thickness values of individual layers, the goal of learning the association between the inputs and the outputs can be eventually achieved.

Rumelhart, Hinton and McClelland (1986) describe a general framework for parallel distributed processing models or connectionist models from eight aspects. This framework is largely based on neuron-inspired models. Table 4.1 illustrates a comparison between this framework and the OTFM.

Neural Network Models	OTFM
A set of processing units, including input, hidden and output units.	Individual thin-film layers, which are only used as input units. No hidden and output units. Outputs are measured by reflection or transmission coefficients over different wavelengths.
A state of activation.	Intermediate reflection coefficient at a boundary of each layer.
An output function for each unit.	Intermediate reflection coefficient at a boundary of each layer.
A pattern of connectivity among units. It is normally represented by weighted connections in the network.	Interaction among all the layers through a lightwave being reflected and transmitted at boundaries of each layer. It can be represented by the thicknesses of individual layers.
A propagation rule for propagating patterns of activities through the network. Generally it is the weighted sum of inputs to the unit.	Lightwave amplitude propagation through all the thin-film layers. It is not the weighted sum of inputs to a layer.
An activation rule for combining the inputs on a unit with its current state to produce a new level of activation.	No activation rules are used, but the mapping of inputs and outputs for a multilayer is highly nonlinear (see equation (4.11)).
A learning rule whereby patterns of connectivity are modified by experience.	The same.
An environment within which the system must operate.	The same.

Table 4.1. Comparison between neural network models and the OTFM.

As shown in Table 4.1, the corresponding components of the OTFM and the neural network models are different in various ways. The OTFM consists of many thin-film layers as its basic processing units. They can be used as input units (i.e. encoding the input into the refractive index of each layer), but no hidden units and output units are visible in the OTFM. OTFM's outputs

can be measured by the overall reflection or transmission coefficients at different wavelengths. Each layer has a reflection coefficient at its incident boundary, and it can be seen as its activation state, as well as the output of the layer (i.e. in the OTFM, the output level in individual processing unit is equal to the activation level of the layer). Equation (4.4) can be seen as an activation function of a layer. Although it is not a threshold function (which is often adopted in neural network models), the reflection coefficient as the output of a thin-film layer is highly nonlinear (see equations (4.2) and (4.4)). In the OTFM, any change in a thin-film layer can affect the reflection coefficient of every other layer. In this sense, all the thin-film layers as simple processing units are interconnected together, but not grouped into different layers. In contrast most neural networks, the feed-forward neural networks in particular, are often formed by simple processing units grouped into different layers.

In the OTFM context, a lightwave propagates from one layer to another. The reflection coefficient, which is the output value for a multilayer is obtained by recursively using equation (4.11) through all the layers in the model. This is a highly nonlinear calculation. By adjusting the individual layer thicknesses, the OTFM can produce output values (reflection coefficients) as desired. Learning of the association between the inputs and outputs is achieved by searching for such a set of appropriate thickness values of multiple layers. This can be carried out by search algorithms (see Chapter 5), such as the N-squared Scan Method and the Genetic Algorithms adopted in this research. A search algorithm normally tries one set of thickness values at a time. If the evaluation is better than the previous one, the new set of thickness values is retained, and the previous set of thicknesses is discarded. This process continues until the best thickness combination is found. The evaluation is done by measuring the overall difference over all the examples from the training data set (see Section 7.2). More coverage on search algorithms and training procedure will be presented in Chapters 5 and 7.

4.7 Summary

This chapter has described the characteristics and architecture of the OTFM and some calculation algorithms for it. In particular, we have derived a number of schemes to calculate reflection and transmission coefficients for the thin-film multilayer structure, as well as algorithms for more efficient calculation of multilayer properties. The OTFM has a novel architecture compared with the conventional connectionist models. Its optical properties particularly its high nonlinearity and other connectionist features together demonstrate that it is capable of accomplishing some learning tasks that are common to the conventional connectionist models. With the help of optimization methods and fast calculation algorithms, the OTFM may perform comparably to other connectionist models in learning many computational tasks.

In the next chapter, issues concerning optimization in the OTFM will be discussed.

Chapter 5 Optimization

Optical design has traditionally given rise to numerous and difficult optimization problems, and at the same time, has motivated significant research towards their solution. As long as methods of evaluation have been available, it has been possible for designers to work towards optimal designs by the construction of change tables, interpolation, and trial and error (Rigler & Pegis 1980: pp.212).

5.1 Introduction

Optimization typically involves the maximization or minimization of a function that represents the performance of some system. It is carried out by the finding of values for those variables which cause the function to yield an optimal value (Foulds 1981; Beightler, Phillips & Wilde 1979; Press *et al*. 1992; van der Smagt 1994). In an optical thin-film multilayer model (OTFM), the optimization issue is to find the system parameters which satisfy the desired optical specifications. These parameters include the thicknesses and refractive indices of thin-film layers, etc. Basic approaches to the design of thin-film problems include graphical, analytical and numerical methods (Dobrowolski & Kemp 1990). Numerical methods are much more flexible than the other two and can handle complicated problems, and the computations are often based on merit functions which can be defined according to quantities of interest. In this chapter, we first present an overview of the performance of different optimization methods for the thin-film design. This overview provides an overall picture on the issue of thin-film optimization methods. Considering the computational cost involved, and furthermore the constraints to calculate the gradient information for this specific optical thin-film model (OTFM), algorithms that can substantially reduce computation without using derivatives are preferable. Two such search algorithms, the *N-squared Scan Method* (Liddell 1981) and the *Genetic Algorithm* (Holland 1975), which are well suited to this requirement, are described in detail in this chapter. Genetic Algorithms, in particular, are efficient search algorithms for finding a better minimum globally in a highly nonlinear multi-dimensional search space, whereas the gradient-based algorithms tend to be easily trapped at local minima.

5.2 An Overview of Thin-Film Optimization Methods

A number of optimization procedures have been developed and applied to different optical thin-film design problems (Liddell 1981; Dobrowolski & Kemp 1990). Dobrowolski has made a comparison of the efficiency of ten different optimization methods as shown in Table 5.1 (Dobrowolski & Kemp 1990; Li & Dobrowolski 1992). They were all implemented in conjunction with the same thin-film design program, since such a comparison would not be possible if different starting design, constants or routines were used. These optimization methods can be classified according to whether they vary one or all of the system parameters at a time. The parameters refer to the thicknesses only, the refractive indices only, or the thicknesses and refractive indices together of the layers of the system. Another way of categorizing the optimization methods is by whether they require the calculation of derivatives or not. This calculation might not be practical when the quantities of interest are particularly complicated.

1	2	3	4	5	6	7	8
A Adaptive Random Search	ARS	Yes	Yes	No	D	0.107	0.221
B Damped Least Squares	DLSQ	No	Yes	Yes	A	0.058	0.091
C Modified Gradient	GRAD	Yes	Yes	Yes	A	0.134	0.188
D Golden Section	GOLD	Yes	No	No	N.A.	0.086	0.166
E Hooke & Jeeves Pattern Search	H&J	Yes	Yes	No	A,B	0.054	0.094
F Basic Powell's Conjugate Search	BPCS	Yes	Yes	No	C	0.062	0.163
G Rosenbrock's Rotating Coordinates	ROSE	Yes	Yes	No	B	0.074	0.178
H Generalized Simulated Annealing	GSA	Yes	Yes	No	D	0.105	0.376
I Monte Carlo Simulated Annealing	MCSA	Yes	Yes	No	D	0.099	0.198
J Revised Nelder-Mead Simplex	SIMP	Yes	Yes	No	B	0.114	0.309

Table 5.1. Names and properties of optimization methods investigated.
Column information: 1 - Name of method; 2 - Code; 3 - Use of merit function;
4 - Simultaneous optimization of parameters; 5 - Use of derivatives; 6 - Type of parameter constraint (see text); 7,8 - Performance factors *PF1, PF2* (see text) (after Dobrowolski & Kemp 1990).

Most of the routines described in Table 5.1 were originally designed for unconstrained optimization. Some routines were modified to include constraints to prevent negative thicknesses and to keep the refractive indices within the range of available values. Four different methods were used to control the system parameters, so if a parameter goes out of bounds:

A) it can be set equal to the boundary value;
B) the merit function can be set equal to an arbitrarily large value;
C) a penalty can be added to the merit function that is proportional to how far the parameter went out of bounds; and
D) the parameter change vector can be rejected outright.

Since it is possible that the performance of these methods may, to a certain extent, depend on the particular problem, Dobrowolski applied each of these optimization procedures to the solution of three different thin-film design problems: an infrared antireflection coating, a 50% broadband visible reflector and a wide angle infrared absorber coating. A good best-performance of a method for all three examples, coupled with an insensitivity to the starting parameters, could be considered as a strong endorsement of that method. Dobrowolski used two performance factors, *PF1* and *PF2* to evaluate the methods. The factor *PF1*,

$$PF1 = 1/3\sum_{i=1}^{3} (M_{B,i} / M_{S,i}) \tag{5.1}$$

where I denotes the problem number and $M_{S,i}$ the merit function of the starting design, and $M_{B,i}$ the best merit function, rates the ability of the method to give the best result. If factor *PF2*, given by

$$PF2 = 1/3\sum_{i=1}^{3} [(M_{A,i} + \sigma_{A,i}) / M_{S,i}] \tag{5.2}$$

(where $M_{A,i}$ is the average value of the merit function and $\sigma_{A,i}$ is its standard deviation for ten different sets of optimization), is close to *PF1*, then the method is considered to be robust. Though the three problems examined are not sufficient to draw any valid statistical conclusions, these two factors can be used as a rough indication of the performance of the methods. According to this criterion, the *Hooke and Jeeves* and the *Damped Least-squares Methods* are among the best. In other methods such as *Golden Section Method* and *Basic Powell Conjugate Search Procedure* only a single parameter or a subspace of the parameter space are searched each time, so they are not as efficient as those global search algorithms for multidimensional search problems.

Dobrowolski also pointed out that his intention in this review was to compare and contrast different optimization methods and find how efficiently they converge to a minimum. However because of the differences in using constants, routines and rounding, etc., this review can only give a rough estimation on the performance of different optimization procedures. The optical thin-film multilayer model (OTFM) proposed in this research is applicable to a wider range of learning and optimization problems than those considered by Dobrowolski. These problems may involve a vast number of local minima, in which some search routines may become trapped. For this reason we have adopted two other search algorithms which are useful for searching for a global minimum. Two other search algorithms, the N-squared Scan Method (Liddell 1981) and the Genetic Algorithm described in the following sections, are found to be sufficient in solving many learning tasks though they are not included in Dobrowolski's review. We have adopted them in the OTFM learning, in order to overcome the optimization problems with this specific thin-film model design.

5.3 Optimization for OTFM

A function that measures the performance of the OTFM must be first defined and then we can apply various optimization procedures to minimize the value of such a merit function. For the thin-film design problem, the Least Squares Method is used in conjunction with merit function evaluation (Liddell 1981) and applied to the OTFM that has fixed values of refractive index and a pre-specified total number of layers (see Section 7.2 for more discussion on the merit functions). The merit function adopted here is as follows:

$$F(x) = \sum_{k=1}^{m} (R_0(\lambda_k) - R(x,\lambda_k))^2 \qquad (5.3)$$

where x is the vector of design variables, R_0 is the specified reflectance at λ_k and R is the computed value of reflectance at λ_k for a particular value of x. m is the number of wavelengths. The program based on this method has the following three basic requirements:

- a routine for computing the reflectance of a multilayer with specified layer thicknesses.
- a means of comparing the resultant reflectance at any wavelength with the specified value, that is the evaluation of *F(x)* in equation (5.3).
- a systematic method for adjusting the system variables in order to produce a minimum of *F(x)*.

The flow diagram of the method is shown in Figure 5.1.

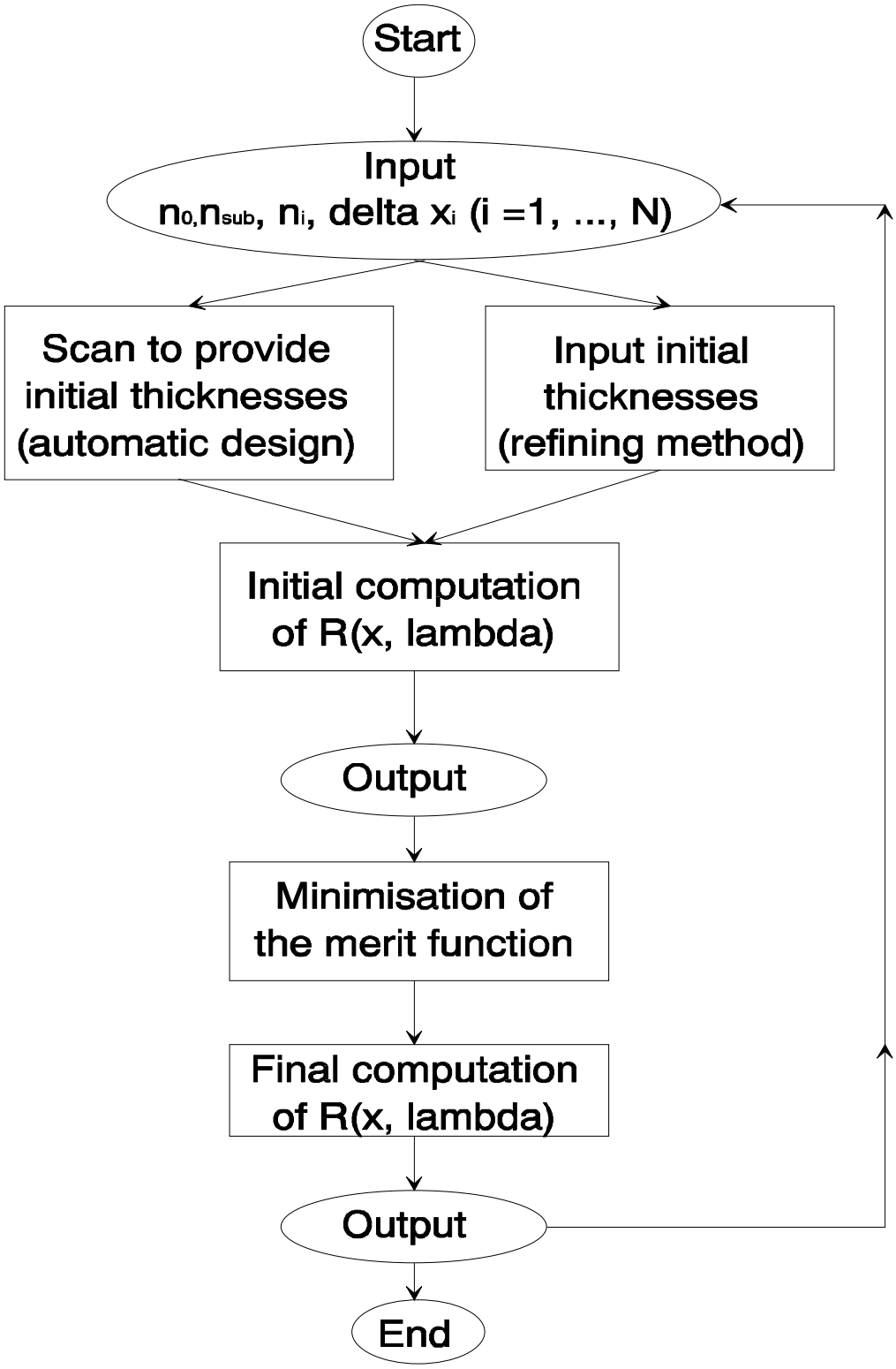


Figure 5.1. Flow diagram for the Least Squares Method (after Heavens and Liddell 1968).

We can either generate a completely automatic design, or refine an initial design (Figure 5.1). The difference between these two approaches is the initial scan to locate a suitable initial value which is incorporated in the automatic design program. The Genetic Algorithm (see Section 5.3.2) can be regarded as an instance of the former approach, since the GA starts with a randomly chosen initial population and then automatically searches for initial values, whereas the N-squared Scan Method described in the following section usually uses the latter approach.

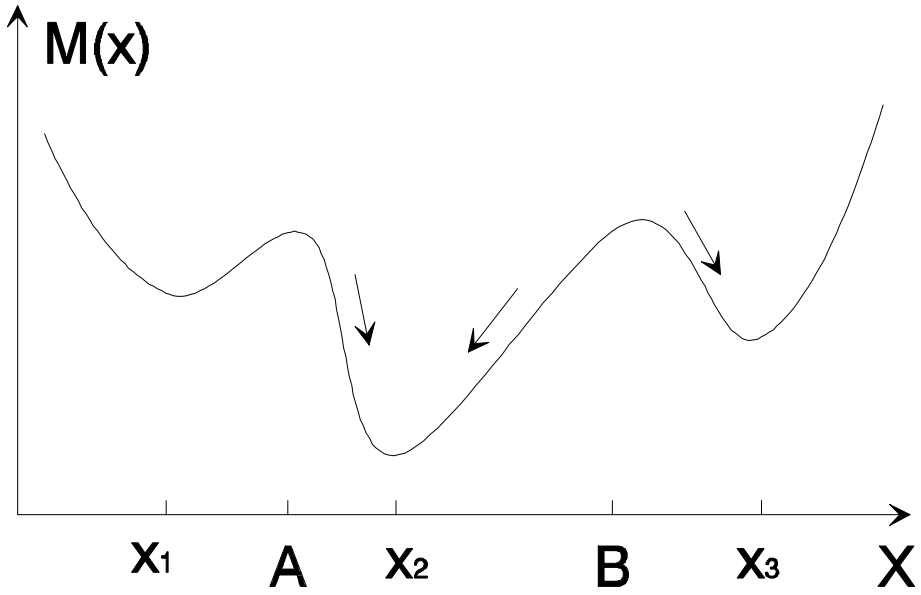


Figure 5.2. The local minima problem.

Optimization methods often have the so called local minima problem, as shown in Figure 5.2, where x represents a parameter vector of thickness values and $M(x)$ is an evaluation determined by x. Three minima are found at x_1, x_2 and x_3. Note that there can be many peaks and valleys, and it is not always the case that the lowest valley (i.e. global minimum) can be reached. Assuming that the initial state is between A and B, if we use the gradient descent search algorithm, the evaluation of M evolves towards the final state x_2, but if the initial state is outside of this range, M likely reaches only local minima x_1 or x_3. This "local minima" problem could be encountered one way or the other by various search algorithms. For the OTFM's work in this thesis, this problem has been handled by the N-squared Scan Method or the Genetic Algorithm by beginning a global search at many different, scattered points, and identifying promising regions and then focusing on a local regional search. Although it is still possible to miss some good minima,

starting with a global search increases the likelihood of finding a minimum that is satisfactory for the given problem specification.

Training the optical thin-film multilayer involves searching for an appropriate set of thickness values and then calculating the merit function value for each set of thicknesses. As discussed above, it can be difficult. In particular, two problems need to be addressed:

- Exhaustive search of all possible sets of thicknesses is not appropriate for larger numbers of layers. For instance, for a 10-layer model, assume that each thickness varies 20 times, it would require 20^{10} evaluations, which is intractable.

- Equations (4.11) and (4.12) are used to compute the reflection and transmission coefficients of a multilayer. Whenever a layer thickness is adjusted, in order to obtain the reflection and transmission coefficients of the new multilayer structure, the reflection and transmission coefficients for each layer must be calculated again, and this would take a considerable amount of computation.

The first problem can be dealt with by many optimization procedures of optical thin-film multilayer refinement (Dobrowolski & Kemp 1990; Liddell 1981). The second problem can be handled by an algorithmic routine (Purvis 1982) developed to provide an efficient calculation procedure for the multilayer reflectance, without relying on equations (4.11) and (4.12) to compute the reflection and transmission coefficients for each layer repeatedly (see Section 4.5). It takes advantage of some intermediate elements of previous multilayer calculations that can be retained for subsequent use. Two optimization procedures are adopted to provide more effective searching in the system parameter space. One is the N-squared Scan Method (Liddell 1981), to reduce the number of searches of thickness combinations. It might miss some optimal solutions, but it has been shown to be useful in practice. The second optimization procedure adopted is the Genetic Algorithm (Holland 1975; Goldberg 1989), which has been widely used as an efficient global search algorithm. These two search algorithms will be described in the following two sections.

5.3.1 N-squared Scan Method

Consider a thin-film stack of *N* layers. Initially vary the thickness of the first layer (the initial variation can start at any *i-th* layer, not necessarily the first one), while the thicknesses of the remaining layers are held constant, and the set of thicknesses that yields the lowest value of the merit function is saved. The same procedure is repeated for layer 2, 3, ..., *N - 1*. Each time the best set of thicknesses is saved. This yields a first approximation to the target. Then the whole sequence is repeated, this time starting with layer 2 and ending with layer 1, and so on. In the final iteration, the layer *N* is varied first, then layer 1, 2, ..., and finally layer *N - 1*. Thus if *q* different thickness values are evaluated for each layer in a *N* layer system, the N-squared Scan Method requires qN^2 evaluation, while the exhaustive search would require q^N evaluations (Liddell 1981).

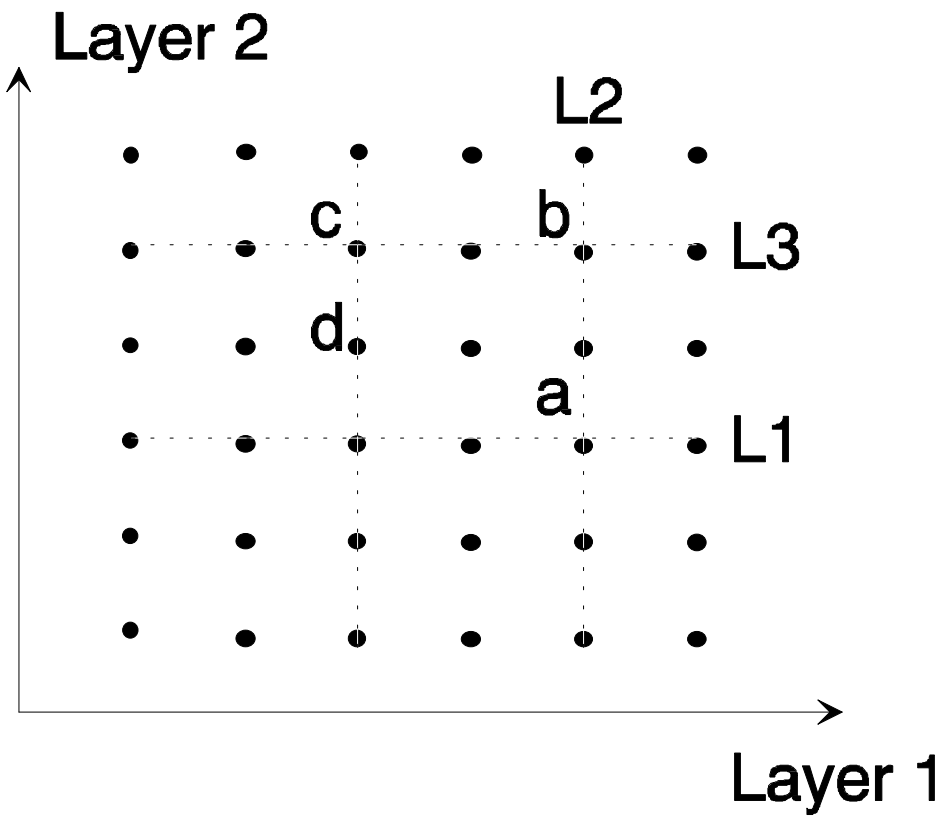


Figure 5.3. Searching for a solution in a 2-layer stack using the N-squared Scan Method.

Figure 5.3 shows schematically how this method searches possible sets of thickness values selectively. Each black dot represents a possible set of thickness values for a solution. For a 2-layer stack, the search proceeds as follows:

i. Assume that the thickness of *Layer* 1 is varied across a range of 6 possible values. It can be depicted by a search moving along line *L1*. Evaluation is done on each of the 6 dots and a minimum is found at dot "a".

ii. The thickness of *Layer* 2 is then varied 6 times. The search continues from "a", moving along line *L2* and finally find a new minimum at dot "b".

iii. Repeat the above two steps until a better minimum is found at dot "d".

This 2-layer thin-film stack using the N-squared Scan Method only requires 24 evaluations (searching along the dashed lines), instead of using the exhaustive search that would require 36 evaluations (all the dots). If *q* is 20, the N-squared Scan Method requires 80 evaluations, while the exhaustive search would require 400 evaluations. This would make a big difference especially when a large number of layers are used. The N-squared Scan Method can be used in conjunction with the two calculation procedures introduced in Section 4.5 in order to further reduce the amount of computation cost needed. Experience shows that this approach has been effective in the design of the OTFM training procedure. This reduced search-space can still yield acceptable results.

5.3.2 Genetic Algorithms

Genetic Algorithms are adaptive searching methods, originated by Holland (Holland, 1975), and loosely based on the principles of genetic variation and natural selection. The appeal of GAs comes from their simplicity and elegance as algorithms, as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. In general, a GA performs a dimensional search, and it encourages information formation and exchange among such dimensions. It does so by maintaining a population of proposed solutions (chromosomes) for a given problem. Each solution is represented in fixed alphabets (often binary) with an established meaning. The population undergoes a simulated evolution: relatively "good" solutions produce offspring, which subsequently replace the "poor" ones. The estimate of the quality of a solution is based on an evaluation function, e.g. equation (5.3). The existence of such a population provides for the superiority of Genetic Algorithms over pure hill-climbing methods, as the GA provides for both exploitation of the most promising solutions and exploration of the search space at any time.

In the simplest form of the GA, bit strings play the role of chromosomes, with individual bits playing the role of genes. An initial population of individual chromosomes (bit strings) is generated randomly, and each individual then receives the result of a numerical evaluation associated with the parameters represented by the chromosome that is later used to make multiple copies (reproduction) of higher-fitness individuals and to eliminate lower-fitness individuals. Genetic operators, such as crossover (exchanging substrings of two parents to obtain two offsprings) and mutation (flipping individual bits) are then applied probabilistically to the population to produce a new population, or generation, of individuals.

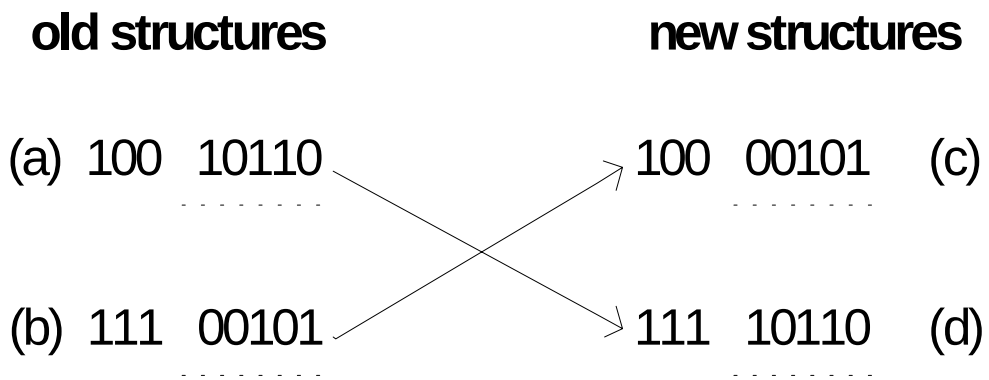


Figure 5.4. New strings produced after crossover.

It could be said that the main distinguishing feature of a GA is the use of crossover (Mitchell 1996). In its simplest form, single-point crossover, a single crossover position is chosen at random and the parts of two parents after the crossover position are exchanged to form two offspring. For examples, the two strings "10010110" and "11100101", split after a randomly chosen position (after the third bit in this case) as shown in Figure 5.4 a) and b), would create strings c) and d) after the swap. The GA is considered to be successful if a population of highly fit individuals evolves as a result of iterating this procedure. There are many other variants of crossover including the two-point crossover, in which two positions are chosen at random and the segments between them are exchanged. There is no simple answer to which crossover operator should be used. The success or failure of a particular crossover operator depends in complicated ways on the particular fitness function, encoding, and other details of the GA (Mitchell 1996).

Compared with crossover in GA, the common view about mutation is that it plays more of a background role. In traditional evolutionary computation methods, such as evolutionary

programming and early versions of evolution strategies, mutation is the only source of variation (Mitchell 1996). It has been recognised that it is not a choice between crossover or mutation but rather the balance among crossover, mutation, and selection that is more important. However precisely how to strike such a balance still remains to be elucidated.

How to set the values for the GA parameters, such as population size, crossover probability, and mutation probability is a tricky issue, because these parameters are associated with one another nonlinearly. They cannot be optimized one at a time. A great deal of literature in evolutionary computation has been devoted to the discussion of GA parameter settings and adaptation, however, there are no conclusive results on what are the best values for parameter settings (Mitchell 1996). A study by De Jong suggested that a good GA performance generally requires the choice of a high crossover probability, a low mutation probability (inversely proportional to the population size), and a moderate population size (Goldberg 1989). De Jong tested a series of parameter values across a five-function suite of function optimization problems in his study. Following these suggestions we adopted a crossover probability 0.6, a mutation probability 0.033 and a population size varying from 300 to 500, in most of the OTFM experiments as described in Chapter 8. Our experience also suggested that generally a satisfactory performance of the OTFM learning can be gained by using these GA parameter values.

Specifying a genetic algorithm for a particular problem involves describing a number of components. Among them, the most important ones are:

a) a genetic representation for potential solutions to the problem, which also defines the search space of the algorithm;
b) a method of generating the initial population of potential solutions;
c) an evaluation function that plays the role of the environment, rating solutions in terms of their “fitness”;
d) genetic operators that alter the composition of chromosomes during recombination;
e) values for various parameters that the GA uses.

The key feature of GA search is that it is domain-independent. If an application problem solution can be represented as a bit string, and syntactic manipulation of that string by the operators produces meaningful new strings, then the GA is a powerful tool for searching noisy,

discontinuous, multidimensional search spaces. The primary source of power in the genetic algorithm approach lies not in the evaluation of individual strings (search space points), but in the way the algorithm exploits, in an implicitly parallel fashion, the wealth of information found in schemata of high performance strings. High performance schemata become the building blocks for future generations of strings and are propagated in parallel throughout the knowledge base, generation after generation.

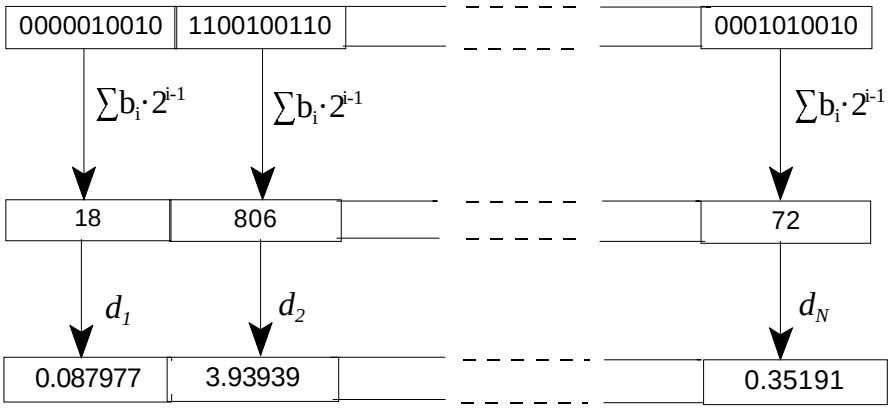


Figure 5.5. Mapping between chromosome segments and a set of thin-film layer thickness values.

Applying Genetic Algorithms to the thin-film model is straightforward, because of its domain-independency. The thin-film calculation procedure does not require any change when applying GA search. In general only two issues are significant: the problem encoding and the evaluation function. Problem encoding with respect to a thin-film model requires mapping its system parameters into a genetic representation, such as a binary string. Figure 5.5 illustrates this encoding process for chromosome segments of 10 bits, which allows the representation of integers {0,1,...,1023}. These integers can be then mapped linearly to a desired interval for thickness values, e.g. d_1, d_2, ..., d_N within the range [0.0, 5.0] (each integer is divided by 1023 and multiplied by 5 respectively). The evaluation function can be used in the same way as in other optimization methods (see equation (5.3)).

The GA search for an optimal set of layer thicknesses can be defined by the following procedure:

Start: generate initial population of binary bits which represent sets of thickness values. For each set of thicknesses, the system evaluates its merit.

Loop through following steps:

a) Select parents whose genetic make-up contributes to fitter offspring.
b) Produce offspring, using suitable genetic operators (reproduction, crossover and mutation).
c) Evaluate the performance of these offspring.
d) Replace certain parents with new offspring.

This process is repeated until the population has converged to a solution or a number of iterations has been reached.

5.4 Summary

Optimization methods developed specifically for the thin-film design problems have been briefly reviewed. Two other search algorithms, the N-squared Scan Method and the Genetic Algorithm have been chosen for the optical thin-film model, because of various constraints and advantages of these two algorithms. In particular, Genetic Algorithms are often described as a robust global search method that can be applied to problems with nondifferentiable functions. However Genetic Algorithms might not be well suited for fine-tuning structures which are very close to optimal solutions (Grefenststte 1987). On the other hand, with the N-squared Scan Method, the search step is adjustable, so it can be used for fine-tuning after a GA search to further improve the performance. It has been found that combining global and local searches can improve the performance of a learning system particularly when dealing with difficult learning problems (Shang and Wah 1996). However experience shows that in many cases either search method can find satisfactory solutions. Experiments involving the N-squared Scan Method and GA search in order to find optimal solutions for the OTFM are described in Chapter 8.

Chapter 6
Software Implementation

6.1 Introduction

As described in Chapter 2, a connectionist model consists of a large number of simple processing units and connections among them. These characteristics make an object-oriented approach particularly suitable for developing connectionist simulation models. By using the object-oriented approach we can have the flexibility of defining new types or classes for those processing units or connections in a connectionist model, and declaring many instances of these classes. These classes can also be used to derive new classes or embraced in other classes for more specific purposes. The OTFM has thin-film layers as individual processing units, and can be viewed as a connectionist model with its own unique architecture and characteristics. The OTFM's connectionist characteristics make the object-oriented approach particularly suitable for constructing its simulation model. This chapter describes the implementation of the OTFM simulation model using the object-oriented approach, as well as how the OTFM is combined into an existing Genetic Algorithm (GA). The structure of class hierarchy and the training procedure are also presented.

The OTFM simulation model was developed using Borland C++ for Windows on a 486 PC machine with 8MB of memory. The Genetic Algorithm code adopted is SGA-C, the C-language translation and extension of the original Pascal SGA code presented by Goldberg (Goldberg 1989; Smith *et al*. 1994). Its operation is basically the same as that of the original Pascal version. In the present work, the GA-Based OTFM simulation code was written in C and incorporated into the GA code and then compiled with the GNU gcc compiler on a Sun SPARC station 5.

The author also developed a data plotting routine for displaying computational results of the OTFM, which was used as part of the iterative process of developing thin-film multilayer structure with suitable properties.

6.2 Modelling OTFM Using Object-Oriented Concepts

We describe OTFM's C++ classes and objects in an order beginning with the simple ones and extending to more complex items. First the class `TSingleLayer`, is used to characterise individual thin-film layers as the basic processing element of the OTFM model. It is defined as follows:

```
Struct Complex
{
   double Re;
   double Im;
};

class TSingleLayer :  public Object
{
    friend class TBlock;
    friend class Item;
   protected:
            double thickness, *Lambdas;
            Complex ref_index, *rx, *tx, f;
            int numlam;
            TPublic_funs PubFuns;
   public:
            TSingleLayer(Complex Ref_index, double Thick,
                   int NumLam, double *Lambdas);
            ~TSingleLayer();
            void Calc_layer_x(double lambdas, Complex &Rx,
              Complex &Tx);  // Calculation for a single layer.
            // the rest of the functions are inherited from Object
            ......
};
```

Note that a complex number type `Complex` is used in this model, which includes real and imaginary components.

Borland's ObjectWindows (1991), an object-oriented class library, which encapsulates the behaviours that Windows applications commonly perform, was used to develop the model. `Object`, as the base class for all derived classes from Borland's ObjectWindows, provides the basic mechanism and structure for type checking and encapsulation.

The `TSingleLayer` class includes variables and functions related to the common properties of a single thin-film layer, as well as variables such as wavelengths that might be used for calculation. These variables include the thickness and refractive index of a layer and an array of wavelengths `Lambdas` to store all specified wavelengths. `rx` and `tx` are arrays storing calculation results, reflection and transmission coefficients. `f` is a Fresnel coefficient which is only dependent on an

individual layer (see equation (4.1)). `TPublic_func` is a predefined class that is in fact a collection of many simple functions necessary for complex number calculation (note that complex numbers are used throughout the OTFM model).

Friend classes `TBlock` and `Item` are used because of necessary access to the protected variables of the `TSingleLayer` class by these two classes.

Having defined the basic data type `TSingleLayer` for a single layer, then a multilayer type, `TBlock`, accommodating more than one layer can be defined. Object-oriented programming offers the advantage that one can reuse or inherit classes from existing standard libraries. In this case, `Array` from the Container Class Library (a standard library of predefined classes) is used for building a new array type `TMultiLayers` to have single layer objects as array elements. `TMultiLayers` inherits all the functions from the `Array` class. This is shown as follows:

```
class TMultiLayers : public Array
{
 public:
   TMultiLayers(int upper, int lower = 0, sizeType aDelta = 0)
            : Array(upper, lower, aDelta){};
   virtual classType isA() const { return __firstUserClass + 1; }
   virtual Pchar nameOf() const { return “TMultiLayers”; }
};
```

The class `TBlock` which specifies a multilayer and its relevant parameters and functions is defined as follows:

```
class TBlock
{
   friend class TSingleLayer;
   friend class Item;
 protected:
   TMultiLayers* MLayers;
   int NumItem, numlam;
   double *d_sav, *lambdas;
   Complex *n, Out_F, In_F, Base, *alfa;
   Complex *r_o, *t_o, *r_o_back, *t_o_back;
   TPublic_funs PubFuns;
 public:
   TBlock(int NumLam, int NumLay, double *Lambdas, Complex *N,
               double *D_sav, Complex In, Complex Out);
   ~TBlock();
   // rest of the functions are for calculation purpose.
   ...
};
```

Within the class `TBlock`, `MLayers` is declared as a pointer to `TMultiLayers`, so that whenever a new `TSingleLayer` instance is created, it can be put into this `MLayers` array as one of its elements. `MLayers` provides a mechanism that links all the single layers together as a whole (indexed), and eases the calculation of reflection coefficients through many ordered layers. Other variables and functions are similar to those in a single layer.

```
class Item
{
   friend class List;
   friend class TOptThinFilm;
public:
...
// Some functions
protected:
   TBlock *SubStrate, *Supers;
...
// Variables and functions.
private:
   Item(int numlam, int maxlay, double *lambdas, Complex *n,
               double *d_sav, Complex *Rtarget, Item *item = 0);
   ~Item();
   Item *next;
};
```

A new class `Item` is created to include two elements of the `TBlock` type, `SubStrate` and `Supers`. This is necessary for implementing efficient computational algorithmic procedures (see Section 4.5). During the OTFM training, when an input example is presented to the OTFM, we use an instance of the class `Item` to store this example, conduct necessary calculation and store the results. `Item` has a pointer (`*next`) of its own type, so these instances can be stored on a linked-list class, `List`, which is defined as follows:

```
class List
{
public:
   List(){head = 0; at_end = 0;}
   ~List(){ remove();}
   void append(Item *item);
   void remove(); // remove all items
   BOOL is_empty();
   void display();
   void readin(ifstream s);
   double TotalMerit(double dx, int layer_to_vary);
protected:
   Item *head, *at_end;
};
```

Thus during training, many `Item`s representing input examples are initially stored on such a linked-list. One iteration of presenting all the training examples in the training data set involves

going through each `Item` on the linked-list. The `head` and `at_end` are two pointers that always point at the beginning and the end of a linked-list. `readin(ifstream s)` is a function that reads in the input data. `TotalMerit(double, int)` calculates the merit function value over all the elements on the linked-list, so training is carried out by many iterations of the evaluation of the merit function value in order to find the lowest merit (see Section 7.2). Other functions are used for maintaining the linked-list itself.

Class `TOptThinFilm` inherits all functions from class `List`. It represents the entire optical thin-film multilayer model (OTFM), which contains all the class objects defined in preceding paragraphs.

```
class TOptThinFilm : public List
{
 protected:
   ......// variables.
 public:
   TOptThinFilm();
   ~TOptThinFilm();
   // Other functions.
   ......
   BOOL encode();
   ......
};
```

`BOOL encode()` implements the training procedure described in Section 6.4.

6.3 Class Hierarchy

The class hierarchy shown in Figure 6.1 depicts the relationships among the defined OTFM classes. In Figure 6.1, each class is shown as a node in a directed graph, where a directed arc indicates that a class is derived from another class. The arrowhead on the arc points to the parent class for a given class, e.g. the class `SubStrate` "is-a" `TBlock;` the class `TOptThinFilm` "is-a" `List`.

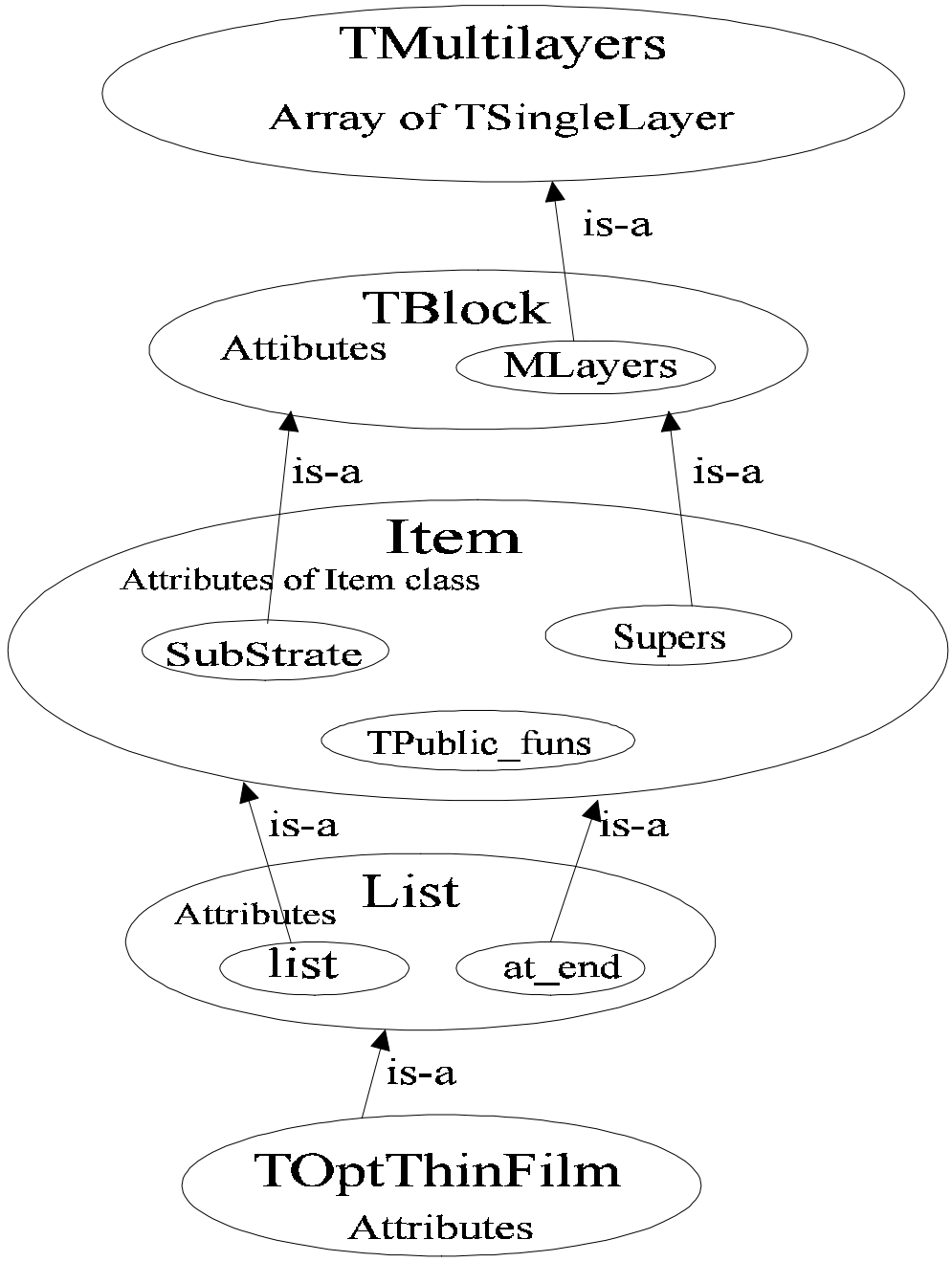


Figure 6.1.Class hierarchy for the OTFM.

There are “is-a” and “has-a” class relationships in Figure 6.1. Class hierarchies which depict the inheritance structure are represented by the “is-a” relationships. The “has-a” relationships are depicted by showing one class containing another (however the “`friend class`” relationships among the OTFM classes are not illustrated in Figure 6.1). These two ways of depicting relationships form the building blocks of the class hierarchy diagram for the OTFM software implementation.

6.4 Training Procedure

Besides using object-oriented concepts building up the basic elements of the simulation model, an important part of the code is to implement a training procedure employing a search algorithm, such as the N-squared Scan Method described in Section 5.3.1. For simplicity, many lines of the code are omitted, and only basic steps at each level of the loops are examined (loops are highlighted). Dotted lines denote that a number of lines are omitted. A double slash “//” denotes a comment or explanation. The function `encode()`, which belongs to the class `TOptThinFilm`, is used for illustrating the training procedure:

```
BOOL TOptThinFilm::encode()
{
  OverallBestMerit = 999.9;    // The best lowest merit of all the searches; starts with an arbitrarily high value.
  ..... // Initialize parameters
  for(iter=1; iter <= num_iterations; iter++)
  {
     ..... // Calculate the searching step (d_inc).
    for(lay_col=1; lay_col <= maxlay; lay_col++)
    {
      BestMerit = 999.9;       // The best lowest merit obtained within the searches of a layer; start with a high
                               // value.
       ..... // Initialize parameters that are used for the next calculation within this loop.
   for(lay_row=1; lay_row <= maxlay; lay_row++)
   {
      ..... // Obtain the actual thickness used for the following calculation.
     for(thick_count=1; thick_count <= numthix_plus1; thick_count++)
     {
       Merit = 0.0;
       for(Item *pt=head; pt; pt=pt->next)  // Go through all the training examples.
       {
         LocalMerit = 0.0; // for each Item on the linked-list.
         for(wav=1; wav <numlam; wav++) //If we use multiple wavelengths.
              LocalMerit = LocalMerit + (Rtarget[wav] - R_u[wav])
                          *(Rtarget[wav] - R_u[wav])/numlam; // Merit function.
         Merit = Merit + LocalMerit/NumItem; //Contribution to the overall merit.
        } // One iteration of presenting examples from the training data set.
                   if(Merit < BestMerit)
        {
              BestMerit = Merit;
              ..... // Save the best merit and the corresponding thickness combination.
        }
         ..... // Vary the thickness of the current layer by a small step.
                 }// for thick_count = ...
                 if(BestMerit < OverallBestMerit)
         {
         OverallBestMerit = BestMerit;
         ..... // Save the best merit (overall) and the corresponding thickness combination.
      }
      ..... // Adjust the parameter settings, and vary the thickness of the next layer.
    }// lay_row loop.
    ..... // Vary the thickness of the next layer.
      }//lay_col loop.
      ..... // Scale down the searching step for even finer, more local searching.
  }// iteration loop.
}
```

The code includes 5 nested loops (if different wavelengths are used, it would be 6). At the beginning of each level of a loop, adjustment and saving of parameter values are carried out. The `iter` loop specifies how many complete searches are needed. The `lay_col` and `lay_row` loops are used for a complete N-squared Scan search of sets of thickness values (see Section 5.3.1). In `thick_count` loop, each iteration involves a new thickness value for the layer that is being altered. Then the merit function value is evaluated (perhaps over several wavelengths). At the end of each iteration loop the best merit and its corresponding thickness combination are saved. The search in this multidimensional search space continues until an appropriate set of layer

thickness values is found that produce the overall best merit, which can be considered as a solution for a required computational task.

6.5 GA-Based OTFM

SGA-C is a C version of Goldberg's Pascal SGA (Goldberg 1989; Smith *et al*. 1994). It is intended to be used for simple GA experimentation, without much consideration for efficiency or the grace of code implementation. There are many more sophisticated GA programs available (see GA archive at: "http://www.aic.nrl.navy.mil/galist/"). SGA-C was coded in such a way that it can be easily used for the incorporation of application specific programs into it. SGA-C was chosen for this research because changes need only be made to one part of the code when incorporating application specific code. Furthermore it can be executed on either PC or Unix platforms.

For the present OTFM work, routines that had to be incorporated into the SGA-C procedure include thin-film specific calculation algorithms, routines for getting input data, calculation of reflection coefficients and evaluation of the fitness of a bit string which represents a set of thin-film layer thickness values. Routines to be modified are placed in *app.c*, where the OTFM specific requirements are coded and sets of thin-film thickness values are converted into bit strings using the *ithruj2int* function before starting a GA search (Smith *et al*. 1994), as well as converting them back to real values after the GA search. Additional variables used for OTFM were declared in both *sga.h* and *external.h*. Further detail can be referred to (Smith *et al*. 1994).

A schematic outline of GA code is illustrated as follows:

```
procedure GA;
begin
   initialize population P(0);
   evaluate P(0);
   t = 1;
   repeat
          select P(t) from P(t-1);
          recombine P(t);
          evaluate P(t);
          t = t + 1;
   until (termination condition);
end;
```

`P(0)` represents an initial population of chromosomes that are randomly generated. These chromosomes are binary strings that can be mapped to many sets of thin-film layer thickness values which are often constrained within a range, i.e. $0.0 < d_i < 5.0$. t denotes a generation step, so `P(t-1)` represents the population of the previous generation. From one generation to another, GA first selects parents (binary strings) whose fitness values are higher from the previous generation. Secondly GA produces new offspring (new binary strings) using various recombination operators (reproduction, crossover and mutation). Finally GA evaluates the fitnesses of these new offspring and replaces some parents with fitter offspring. This process continues until the population has converged to a solution or a number of iterations has been reached. For a detailed description of GA, the reader can be referred to Section 5.3.2.

In order to maintain the flexibility and the ease of execution across both PC and Unix platforms, both the OTFM and GA-based OTFM were coded without specific Windows features. All the necessary input data are read from a number of input data files which use the same name with different extensions, and experimental results are written to a series of output files. There are also intermediate results written to the computer screen in order to assess the on-going performance of the OTFM. For example, when running a GA experiment, it is only necessary to name the input and output files on the command line, e.g. *sga my.input my.output*.

Within the *my.input* file are a number of GA parameters to be specified:

- The number of GA runs to be performed (*int*).
- The population size (*int*).
- The chromosome length (*int*).
- Print the chromosome strings each generation (*y/n*)?
- The maximum number of generations for the run (*int*).
- The probability of crossover (*float*).
- The probability of mutation (*float*).
- Application-specific input, if any.
- The seed for the random number generator (*float*).

6.6 Summary

This chapter has described a number of important building blocks for the software implementation of the OTFM simulation model using an objected-oriented approach. It has demonstrated that object-oriented modelling is an useful approach in developing a connectionist model. The GA-based OTFM has also been briefly described, as well as the training procedure, input and output files, etc. For a source code listing of the OTFM using the N-squared Scan Method, as well as the GA-Based OTFM, the reader can refer to Appendix A and B. The simulation model described in this chapter is the testbed of the proposed Optical Thin-Film Multilayer model. We will describe various experiments conducted in conjunction with this simulation model in Chapter 8. The next chapter will discuss issues concerning how the experiments are conducted.

Chapter 7
Experimental Methodology

7.1 Introduction

Comparative statistical evaluation is significant for neural network experiments, as it is with other empirical techniques. However in a study by Prechelt (1995), out of 119 articles about neural network learning published in 1993 and 1994 in well known journals, he “observes a general lack of comparison with other algorithms and the use of too few and often artificial data sets” (Flexer 1996). 29% of the articles neglected to use a realistic learning problem. One third of them do not present a quantitative comparison with previously known algorithms.

In this chapter a number of issues concerning the way experiments are conducted with the OTFM are discussed. We first describe the training procedure for the OTFM, since the performance of the OTFM is largely determined by how it is trained. Then we discuss the OTFM's system parameters which are critical to the OTFM design. Different initial values of the system parameters may have a dramatic impact on solutions. We then go on to illustrate the important matter of how input data are fed into the OTFM in comparison with how it is done with the feed-forward neural network model. We then describe a number of widely used data sets selected from the literature in the field of connectionist models. Later on, in Chapter 8, we describe experiments conducted on the OTFM with these data sets. We also provide an overview of techniques on how to evaluate the performance of a connectionist model. In the present research, we try to adopt the most reliable and widely used approaches and techniques wherever possible. The problem with noise and missing information is also considered. As other connectionist models generally work well in the presence of noise, for comparison, it is important to test how the OTFM responds and degrades when feeding the trained model with noisy data.

7.2 Training Procedure

The goal of training in the context of the OTFM is to determine a set of appropriate thicknesses of the thin-film layers in a multilayer model that satisfies the target optical specifications. This procedure can be viewed as the learning process of the OTFM. The training described here is supervised learning, where learning is done on the basis of direct comparison of the output of the model with known correct answers (Hertz, Krogh & Palmer 1991). It can be described in detail as follows:

a) Collect many samples to serve as exemplars. Each sample in the training set completely specifies all inputs, as well as the outputs which are desired when those inputs are presented.
b) Choose a subset of the training set and present the samples in that subset to the model, one at a time. For each sample, compare the outputs obtained by the model with the desired outputs.
c) After the entire subset of training samples has been processed, we update the thicknesses of layers in the thin-film learning model. This updating is done in such a way that we hope to reduce a measure of the error (e.g. a merit function, equation (7.1)) in the model's results.

Steps b) and c) in the above procedure are reiterated until an acceptable output is obtained when the model is presented with the corresponding input. The extent to which the output is acceptable can be measured by the value of a merit function which generally takes the form:

$$M(x) = \sum_{k=1}^{m} (R_0(\lambda_k) - R(x,\lambda_k))^2 \qquad (7.1)$$

where x is the vector of design variables including parameters such as individual layer thicknesses; R_0 is the target reflectance at λ_k; R is the computed value of reflectance at λ_k for a particular value of x; and m is the number of wavelengths at which output is measured. Minimization of the merit function value shown in equation (7.1) is an example of the Least Squared Method (It is in fact the same merit function as equation (5.3), as described in Section 5.3). It is equivalent to the cost function applied in a neural network. This merit function can also have another variation:

$$M(x) = \sum_{k=1}^{m} |(r_0(\lambda_k) - r(x,\lambda_k)|^2 \qquad (7.2)$$

where r_0 is the target reflection coefficient at λ_k, and r is the computed reflection coefficient at λ_k for a particular x. Note that r_0 and r are complex numbers, for example, $r = j + i_k$ (j and k are the real and imaginary components of r), so $R = |r|^2$. R is the square of the absolute value of r. The difference between equation (7.2) and (7.1) is that using equation (7.2), the real and imaginary parts of a reflection coefficient can be treated as independent values in the range of -1.0 to 1.0, rather than the reflectance from 0.0 to 1.0. Equation (7.2) provides one extra variable that can be specified for a target (real and imaginary components of r_0 rather than just a real value R_0). Hence it is possible to specify more tasks with fewer wavelengths, and as a result, a considerable amount of computation is saved. However equation (7.1) is the conventional approach of measuring optical performance and it is easier to implement in a physical optical realization of the OTFM.

Another way of defining the merit or evaluation function for training is that when using the *Forced Classification*, one can define the merit function value based on the classification performance, i.e. the percentage of the correct classified examples out of the total number of training (testing) examples, as follows:

$$M(x) = \frac{number\ of\ correct\ examples}{total\ numberof\ training\ (testing)\ examples} \qquad (7.3)$$

Note that examples can include more than one category, so the merit $M(x)$ is in fact the percentage of correct classification out of the total number of training (testing) examples for all categories.

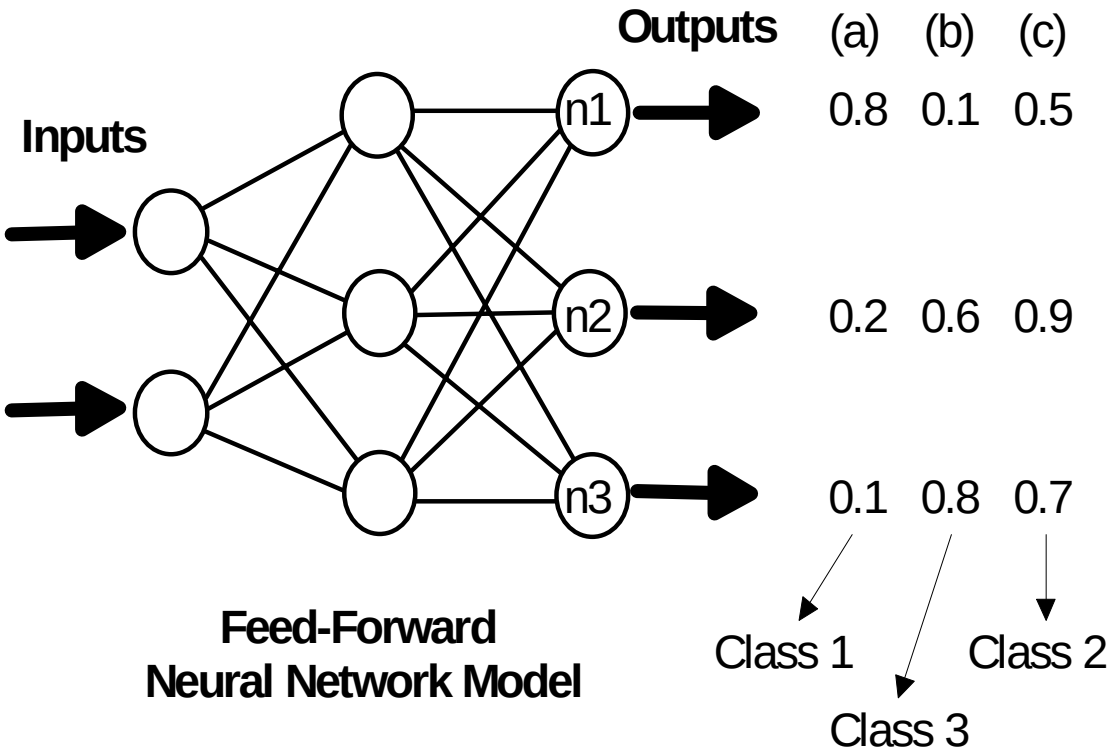


Figure 7.1. Forced classification example.

Forced classification can be illustrated by an example in Figure 7.1, where a feed-forward neural network has been used for training to classify three categories. Three output nodes are used, and each one is used for classifying each category respectively. For example, the output node (n1) is used for class 1, (n2) for class 2 and (n3) for class 3. During the training phase, when an input example is presented to the network, the network produces an output value for each output node. Among these three values, if output node (n1) is the highest one, then the input example will be assigned to the class 1; if it is matched to the genuine category as we know for each training input and output pair, then it will be counted as a correct classification. After one iteration of presenting all the input examples from the training set, the merit value can be evaluated using equation (7.3), i.e. the percentage of the correct classified examples out of the total number of training examples of class 1. In the same way we can get the merit values for class 2 and 3, so equation (7.3) can be seen as a sum of these three values for class 1, 2 and 3.

This process is repeated for each iteration of presenting training examples, and the goal of training is to search for a set of optimal values of system learning parameters that produce the best overall merit using the training examples. When testing, an example is assigned to the category in the exactly same manner. According to the above method, three examples are given in Figure 7.1, where a) is classified as class 1, b) is classified as class 3, and c) is classified as class 2 respectively, based on the highest value of the output nodes.

For the OTFM, for example, if the task is to classify two categories of the breast cancer data set as either developing or not developing a recurrence of breast cancer (see Section 7.5), two wavelengths can be used for training in order to produce two output bits for classification on two classes (equivalent to two output nodes in a neural network model). The learning criterion specifies that for an output to be classified as belonging to the first class, the reflectance at the first wavelength must be higher than the reflectance at the second one; and for it to be classified in the second class, we want to see just the opposite, i.e. the reflectance at the first wavelength should be lower than the reflectance at the second wavelength.

The above approach or similar approaches combining multiple learning criteria can be found in the literature (De Jong *et al*. 1993; Janikow 1993). We have applied this approach, in particular, to the problems of breast cancer prognosis (see Section 8.4.3). A similar "winner-take-all" approach was used in classification of the iris data (see Section 8.4.1), however the merit function value was calculated using equation (7.2), instead of (7.3).

Besides the above forms (i.e. equations (7.1), (7.2) and (7.3)), merit functions can be defined in many different ways. There is a review by Dobrowolski, Ho, Belkind and Koss (1989), on the different types of merit functions that have been used in optical thin-film calculations. Further research in this regard is desired.

The training procedure is an iterative approach that can be described in the following steps:

(1) Initialize the thickness and refractive index of each layer, and assign the initial overall merit $M_B = \infty$. Specify the R_0 (target) at each λ (wavelength). M_B is the current lowest (best) value of the merit function that has been found at each stage of this algorithm.

(2) Repeat until the M_B is sufficiently small (or only repeat for a specified number of iterations).

(2a) Vary thicknesses of some layers (by using some suitable search algorithms).

(2b) Calculate R at each λ.

(2c) Calculate M, the sum of the square of the error, $(R - R_0)$, at each λ by using the merit function $M(x)$ introduced by equations (7.1) or (7.2).

(2d) If $M < M_B$, then set $M_B = M$, and update the thickness of each layer and other parameters necessary for the later calculations.

(2e) Adjust the system parameters for the next calculation.

This process continues until a satisfactory solution is found.

7.3 OTFM Parameters

The OTFM requires careful use of system parameters in order to get satisfying training results. It requires the model designer to have some knowledge of optical properties. A number of significant system parameters are discussed here.

The choice of the number of layers is up to the model designer to determine. It can be compared with the number of hidden nodes in a feed-forward neural network. However, since input attributes need to be mapped into refractive indices of many layers, the minimum number of layers to choose has to be no less than the number of input attributes. The number of layers may also

potentially affect the learning ability of the OTFM - too many layers could slow down the learning or increase complexity unnecessarily (like too many hidden nodes in a neural network); too few layers might restrict the self-organizing ability of the OTFM to achieve the desired computation.

The reflection coefficient is a complex number, with real and imaginary components in the range of -1.0 to 1.0. It is used as the output of the multilayer learning model, and evaluated in connection with the merit function (see equation (7.2)). Note that one can get different reflection coefficients at different wavelengths, so the output can be an array of such reflection coefficients (not just one), compared with other connectionist models, most of which have a set of individual, explicit output nodes.

Thicknesses of individual layers are analogous to the connection weights in a neural network model. In order to obtain desired outputs, the model's layer thicknesses are adjusted in an effort to obtain a set of appropriate thicknesses. This is fundamental to the actual learning process of the OTFM. Once such a set of layer thicknesses is found, it can then be used for retrieving previously learned patterns.

The wavelength is another significant parameter. It is always part of the training process (all the light waves passing through the multilayer material have a wavelength). It gives training an extra dimension, since training can be achieved over a range of different wavelengths, while retaining the same thickness combination (Because of the linear superposition property of lightwave at ordinary interface, these input at multiple wavelengths could be applied and measured simultaneously in a real physical implementation).

The refractive index (n), the thickness (d) of each layer, and the wavelength λ must be initialized prior to the training of the OTFM. Generally we initialize refractive indices of all the layers alternatingly with high and low values, i.e. if i-th layer has a relative higher n, then $(i+1)$th layer would have a lower value of n. The thickness of each layer d must be set within certain range with respect to the distribution of the reflectance values. For the outputs of the OTFM, the reflectance values or the reflection coefficients at different wavelengths are normally used. It is noted in equation (4.11) that the reflection coefficient is invariant with respect to a change in $n_i d_i$ (i-th layer) by a amount $\lambda/2$ (see Section 4.3.2), so only the range $0 \le 4\pi n_i d_i / \lambda \le 2\pi$ is normally considered (Case 1983). For the overall the OTFM initialization, normally we consider the

thicknesses and refractive indices to be in the range of $0.0 \leq d_i \leq 5.0$ μm, and $1.0 \leq n_i \leq 4.5$ respectively.

7.4 Encoding Inputs

The available parameters that can be manipulated by the designer of the thin-film systems are thicknesses of the layers, the wavelengths used, the refractive indices and absorption coefficients of the substrate, the surrounding medium, and the materials used for construction of the multilayer. The reflection (or transmission) coefficients of the multilayer, with real and imaginary values always in the range of -1.0 to +1.0, may be considered to be generalized outputs as follows:

$$r = f(n_i, d_i, \lambda_j); \quad i = 1,..., N \quad , \quad j = 1,..., m \tag{7.4}$$

where N is the number of layers and m is the number of wavelengths used. It is also possible, by equation (7.1), to use more conventional optical measurements, reflectance and transmittance as generalized outputs, which would take real values between 0.0 and 1.0. With these generalized outputs and for a specified range of optical wavelengths, almost any desired spectral characteristic of these coefficients can be obtained by choosing a set of thin-film layers with appropriate values of layer thickness and refractive index.

Two encoding schemes are adopted for feeding input information into the thin-film systems, one involving the wavelengths and the other involving the thin-film refractive indices (Purvis & Li 1993; 1995).

In encoding method #1, where the input would consist of an array of binary numbers, each input array is first converted into a single binary number and then mapped into a unique decimal number that is to represent the wavelength for an optical signal. Each input, then corresponds to a lightwave at a different wavelength value.

Input		Decimal		Wav		Wavelength
0000	→	0	→	1	→	0.100
0001	→	1	→	2	→	0.105634
0010	→	2	→	3	→	0.11194
0011	→	3	→	4	→	0.119048
0100	→	4	→	5	→	0.127119
0101	→	5	→	6	→	0.136364
0110	→	6	→	7	→	0.147059
0111	→	7	→	8	→	0.159574
1000	→	8	→	9	→	0.174419
1001	→	9	→	10	→	0.192308
1010	→	10	→	11	→	0.214286
1011	→	11	→	12	→	0.241935
1100	→	12	→	13	→	0.277778
1101	→	13	→	14	→	0.326087
1110	→	14	→	15	→	0.394737
1111	→	15	→	16	→	0.500

Table 7.1. Converting the sixteen 4-bit input rows into a range of wavelengths.

Cons

idering the parity problem as shown in Table 7.1, an input row of 4-bit binary numbers is converted into a decimal number, which is then mapped to a wavelength to be used in the training. Transferring these binary numbers into a decimal number is done by

$$decimal = w\cdot 2^3 + x\cdot 2^2 + y\cdot 2^1 + z\cdot 2^0 \tag{7.5}$$

where *w, x, y* and *z* represent each digit of a 4-digit binary number respectively. For example, as shown in Table 7.1, an input row {1,0,1,0} can be converted to 10 by the above formula (7.5). This decimal number has a corresponding value to *wav*, and this *wav* can then be encoded into a wavelength by

$$Wavelength[wav] = \frac{1}{1/minlam - LamFac \cdot (wav-1)} \tag{7.6}$$

where

$$LamFac = \frac{1/minlam - 1/maxlam}{numlam - 1}, \quad wav = 1,2,..., numlam$$

Wavelength[wav] is a specific wavelength of an array at the *wav*-th value of wavelength. *minlam* and *maxlam* denote the minimum and maximum wavelengths. *numlam* is the number of wavelengths. Thus eventually a range of unique wavelengths are obtained corresponding to such an input row (Table 7.1). Note that it is an unevenly spaced range of wavelengths from *minlam* to *maxlam*. This range of wavelengths is the actual range of input values used for training. The experiment using encoding method #1 to solve the 4-bit parity problem will be presented in Chapter 8.

In encoding method #2, the refractive indices of the various thin-film layers are used as input variables. Feeding the OTFM with *M* input values would require at least *M* thin-film layers to be available. The thicknesses of the various layers are then used as adjustable "weights" to obtain the desired outputs.

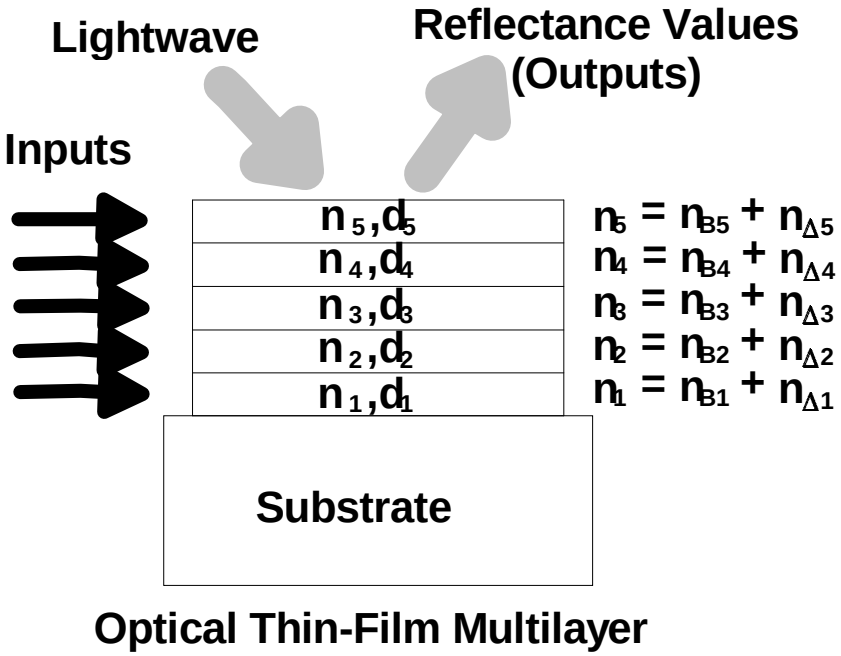


Figure 7.2. A 5-layer stack with inputs encoded as incremental values of the refractive indices.

Figure 7.2 illustrates how this encoding method works. A 5-layer stack is used here as an example, where each layer is characterized by its refractive index n_i, and thickness d_i. For each

layer *i*, one of the inputs is encoded into an appropriately scaled value of the refractive index $n_{\Delta i}$, and added to a base value of that layer's refractive index n_{Bi} (with $n_{\Delta i} << n_{Bi}$) to arrive at the layer's index value n_i. The value of n_{Bi} are kept large enough relative to the encoded incremental values $n_{\Delta i}$, to ensure that there is always a substantial variation of the refractive index n_i from layer to layer, irrespective of the value of the input for that layer. Note that it is not necessary for every layer of the stack to be used for input; a thin-film stack could be designed with some extra layers that are not used for encoding input. Extra layers might be necessary when the stack deals with more difficult learning problems.

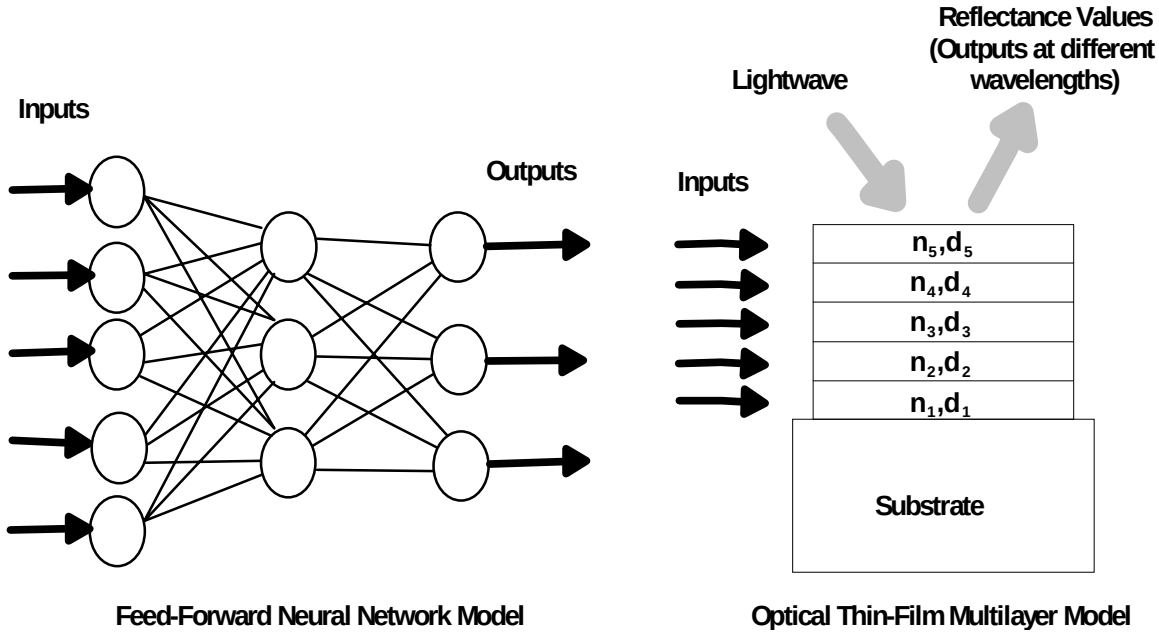


Figure 7.3. Comparison between a neural network model and the OTFM.

Feeding input information into the OTFM using encoding method #2 is a fundamental part of the OTFM that is proposed as a novel connectionist architecture. Figure 7.3 provides a comparison between a typical neural network model and the OTFM. Note that in a neural network model there are explicit input nodes and output nodes, while in the OTFM, inputs are normally fed into the model by using encoding method #2, and the outputs are measured by the overall reflectance of the thin-film multilayer structure at different wavelengths. During the learning phase, the connection weights in a neural network model (i.e. among all the nodes) are adjusted in order to produce desired outputs, compared with the thicknesses of multiple layers to be adjusted in the OTFM.

Encoding method #1 is more limited than the encoding method #2. In dealing with the parity

problem, it requires that a mapping be found between the input rows, which have to be an array of binary numbers, and a range of different wavelengths, therefore the wavelengths are no longer free parameters during training. Encoding method #2 does not have such limitations. It converts the input digits directly into the refractive indices of layers in a distributed way ("distributed representations", as described in Section 2.4), without putting any restriction on the wavelengths. Many researchers regard using distributed representations as fundamental to a connectionist model (Andy & Lutz 1992). Thus encoding method #2 is considered to be the more general approach with input values, while method #1 is simpler to implement on an optical system.

7.5 Data Sets

Experiments were conducted in this thesis using data sets that have been widely used in the field of connectionist models (Rumelhart & McClelland 1988; Hertz, Krogh & Palmer 1991; Weiss, Kapouleas 1989; Hart 1992; Fisher, McKusik 1989; Ripley 1993). These data sets are divided into three groups: PARITY, PATTERN RECOGNITION and REAL-WORLD PROBLEMS, including the XOR, four-bit parity, four and eight 5-by-5 grid pattern recognitions, breast cancer prognosis, iris data classification and gas furnace time series prediction.

PARITY

XOR: It is the Boolean exclusive-OR or XOR function, which is the simplest case of a *N*-input parity problem. It has only four 2-bit binary numbers as inputs. The desired output is 1.0 if one input is 0.0 and the other is 1.0, and the output is 0.0 if both inputs are 0.0 or both are 1.0. In other words if the number of inputs that have a value of 1 is an even number (the parity is even), then the output is 0.0. If the parity is odd, the output is 1.0.

Four-bit Parity: This data set consists of 16 four-bit binary numbers as inputs, which is used for solving the four-bit parity problem. The desired output is 0.0 if the parity of the four-bit number is even, and 1.0 if the parity is odd.

PATTERN RECOGNITION

Four 5-by-5 Grid Pattern Recognition: This data set consists of 4 input examples, each with 25 binary numbers representing the pixels of an alphabetic character on a 5-by-5 grid.

Eight 5-by-5 Pattern Recognition: This data set consists of 8 input examples, each with 25 binary numbers representing a letter on 5-by-5 grids.

REAL-WORLD PROBLEMS

Breast Cancer Prognosis: This is a data set for evaluating the prognosis of breast cancer recurrence (Michalski *et al.* 1986). The data set consists of a total of 286 samples, with 9 attributes, and 2 output classes. There are 9 instances of missing attribute values. Following Michalski's sampling method, 70% of examples were randomly selected for training and the remaining 30% were used for testing. This process was repeated 4 times.

Iris Data Classification: The iris data was first used by Fisher (1936) and is still one of the standard discriminant analysis examples (Weiss & Kapouleas 1989). There are three classes of iris flowers to be discriminated using four continuous valued features that represent physical characteristics of the flowers. The data set consists of 150 cases, 50 for each class. The iris data set was divided into training and testing data sets. After training the system using the training data set, its performance was measured on the testing data set. The goal of training is to enable the model to classify well on these three different classes of iris flowers, not only on the training data set, but also on novel examples from the test data set (i.e. generalization).

Gas Furnace Time Series Data: This data set is used for the modelling of a dynamical process (Box and Jenkins 1970), which is a gas furnace with the gas flow rate as the single input *u(t)* and the CO2 concentration as the single output *y(t)*. *t* denotes the present time. The time series used for prediction purposes consists of 292 data pairs: the first 250 pairs were used for training and the remaining 42 were used for prediction. Training can be carried out by using different variables from the historical data. Three training runs were conducted for the prediction purpose. The first one used two variables *y(t-1)* and *u(t-1)* as input attributes to predict the current *y(t)* (however in this training, all the 292 data pairs were used for training); the second one used *y(t-1)*, *y(t-2)*, *u(t-1)* and *u(t-2)* as input attributes; and the third one considered ten variables *y(t-1)*, ..., *y(t-4)*, *u(t-1)*, ..., *u(t-6)* as input attributes.

In the data sets of the XOR and the four-bit parity problems, the training sets contain all possible input patterns, so there is no question of generalization. In the other data sets, the training set is only a part of the problem domain, and the OTFM is required to generalize to previously unseen

examples from the testing data set. The details of using these data sets for conducting experiments will be presented in Chapter 8.

7.6 Noisy Data and Missing Information

Connectionist models often show their robustness when trained or tested with noisy data. Noise was also used to test the robustness of the thin-film learning system and its ability to degrade gracefully with increasing noise. In the experiments with pattern recognition, a number of patterns in which noise was steadily increased were used to test how the OTFM responds to noise.

Missing values are generally handled in the following ways (Weiss & Kapouleas 1989):

- Fill in the missing values with the means of values for the cases in the same class.
- Throw out the cases or features with the missing values.
- Establish surrogates, i.e., substitutes, for features when their values are missing.

In the breast cancer data set, there are 9 missing attribute values. We adopted the first approach in this case and took the means of values for all the samples of a particular attribute.

There are also inconsistent data descriptions found in the breast cancer data set, i.e. some patients have exactly the same description, but are classified differently. The thin-film learning model was also trained with this contradictory information, without making specific accommodation for these inconsistencies.

7.7 Estimating Error Rate

The techniques for estimating the true error rate are of fundamental importance in evaluating the results for learning systems (Weiss & Kapouleas 1991; Ripley 1993; Cohen 1995; Flexer 1996; Dietterich 1996). The simplest technique for estimating error rates is to separate the sample cases into two groups of cases, a training set which is used to train the model, and a test set which is used to test the trained model (Figure 7.4). The classifier is independently derived from the training cases, and the error estimate is the relative performance of classifier on the test cases. However, such a single random partition of training and test cases can be somewhat misleading.

Instead of a single train-and-test experiment, multiple random train-and-test experiments can be performed. For each random train-and-test partition, a new classifier is derived. The estimated error rate is then taken as the average of the error rates for discriminators derived for the independently and randomly generated partitions. This random resampling technique can produce better error estimates than a single train-and-test partition.

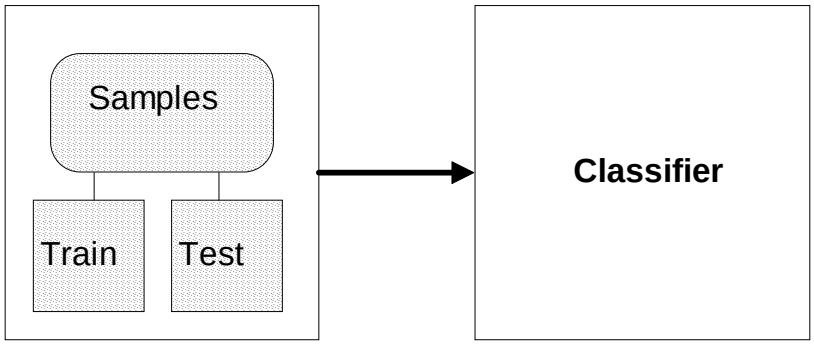


Figure 7.4. Partitioning samples for classifier design.

There is a standard resampling method for dealing with small training sets in traditional statistical techniques. It is called the *cross-validation* error estimation (Weiss & Kapouleas 1989; Masters 1995). In *k*-fold *cross-validation*, the cases are randomly divided into *k* mutually exclusive test partitions of approximately equal size. The model is trained *k* times, each time leaving out one of the subsets from training, and the resulting classifier is tested on the omitted subset. The average error rates over all *k* partitions is the cross-validated error rate. If *k* equals the sample size, this is called *leave-one-out cross-validation*.

Bootstrapping is another resampling based method for estimating generalization error. In its simplest form, instead of repeatedly analysing subsets of the data, one repeatedly analyses subsamples of the data. Each subsample is a random sample with replacement from the full sample.

Some researchers have recommended introducing a third division of data for use (Flexer 1996). As in the *k*-fold *cross-validation*, if the test set used for repeated tuning, in fact, becomes a training set, then the obtained error rates will be biased. With this third division, the network should hold back normally 20% of the data and divide the remaining data into a training set and a testing set, and then tune the parameters using these two sets and an appropriate resampling

technique. The final tuned network should be tested with this never-before-used 20% of the data.

The objective of dividing a sample of cases into a training and testing set is to help design a classifier with minimum error rate. With a single train-and-test partition, too few cases in the training set can lead to the design of a poor classifier, while too few test cases can lead to erroneous error estimates. Using the same data, resampling methods provide an easy way for researchers to make comparisons of classifiers and algorithms. Analysis condition can be readily duplicated, and error estimates with new results can be more easily compared.

In the experiments with the thin-film learning model, a resampling technique is applied to some of the data sets, such as the breast cancer prognosis. In this case, we randomly generate 4 different partitions of training and test data sets, with 70% of the total samples in each training set and 30% in each test set (because this data set partitioning, with this resampling method and these divisions, has been widely used for training by other researchers reported in the literature). Using the same method allows us to readily compare our experimental results with others.

7.8 Summary

There seem to be only few papers in the literature that describe the role of experimentation in the area of connectionist models. In this chapter we have attempted to bring together all the issues concerning experimentation with connectionist models in general, as well as the OTFM in particular. We have illustrated how to feed input information into the OTFM and how training is carried out with the OTFM. Other issues related to conventional connectionist models have also been discussed, including data sets, noise and missing information, and finally, how to evaluate the performance of a connectionist model in a fair way by using resampling techniques. All these issues are of importance when evaluating a novel connectionist model in particular. In the next chapter we will describe the experiments that are largely based on the approaches and techniques introduced in this chapter.

Chapter 8 Experiments

8.1 Introduction

This chapter describes experimentation on the OTFM with various learning examples that are widely used in the field of connectionist models. Most of them have been studied by many researchers (Rumelhart *et al.* 1988; Hertz *et al*. 1991; Weiss & Kapouleas 1989; Hart 1992; Fisher & McKusik 1989; Ripley 1993; Box & Jenkins 1970). In order to examine the performance of the OTFM and compare it with other more conventional connectionist models, experiments were also conducted using the same learning examples on a feed-forward neural network model employing the back-propagation learning algorithm (Hertz, Krogh & Palmer 1991), since it is the most widely used connectionist learning model. The learning rate for the feed-forward neural network model was set to 0.2 and the momentum term was set to 0.1. The same data sets were used for training and testing. The best training result was selected out of 5 training runs for each data set.

For all the experiments illustrated in this chapter, the thin-film multilayer is assumed to be deposited on a substrate with a refractive index equal to 4.0 (this would correspond to a germanium substrate, which is commonly used in microwave thin-film optics), or otherwise stated, and it is used as the output medium. The input medium is air and has a refractive index of 1.0.

In order to train the OTFM, its parameters must be initialized. Apart from the discussion presented in Section 7.3, the initial values for these OTFM parameters were simply chosen by trials to find a good starting design, based on our experience and knowledge of optical systems (similar to a neural network model).

The experiments were conducted on both the OTFM simulation model (see Chapter 6) and a back-propagation feed-forward neural network model which was developed in the Information Science Department at Otago University. Both simulation models were run on a 486 PC platform with 4 Mbytes of memory. When Genetic Algorithms were used, the GA-Based OTFM simulation model was run on a Sun SPARC station 5.

To observe and evaluate how the OTFM solves a wide range of problems, these learning examples for experimentation were divided into three groups: PARITY, PATTERN RECOGNITION and REAL-WORLD PROBLEMS. For a brief description of these data sets, the reader should refer to Section 7.5. For each of the following experiments, an optical description of the corresponding OTFM parameter will be given in a table, where λ denotes the wavelength used; N_λ denotes the number of wavelengths used; Δ denotes the scaling factor which is used for an input bit to be scaled and added to the base value of that layer's refractive index n_{Bi} (see Section 7.4).

A comprehensive summary analysis of the experimental results will be presented in Chapter 9.

8.2 PARITY

Two parity problems, the XOR and 4-bit Parity were used as examples to demonstrate how the OTFM deals with highly nonlinear data sets (the output changes radically when a single input bit is altered). Though the data sets are small, they have been widely used as test examples for problem solving.

8.2.1 XOR problem

The XOR problem is the simplest case of an *N*-input parity problem. It calls for an evaluation of the Boolean exclusive-OR or XOR function. We use 0 to represent even parity and 1 to represent odd parity, as shown in Table 8.1. The desired computation (target) is 1.0 if one or the other of the inputs is 0, and 0 if they are both 0 or both 1. This is perhaps the simplest learning problem that is not linearly separable. The 4-bit parity and the iris classification problems described in the following sections are also examples which are not linearly separable.

Inputs		Targets	OTFM Reflectance (8 layers)	Neural Network Output node (with 2 hidden nodes)
0	0	0	0.100546	0.0276
0	1	1	0.813596	0.9701
1	0	1	0.728428	0.97008
1	1	0	0.195737	0.03752

Table 8.1. Training result on the XOR problem.

Traditionally, the XOR problem played a special role as an impossible problem for a perceptron-like network with only a single layer of trainable weights (Minsky & Papert 1969). The XOR problem has been extensively tested by many researchers (Rumelhart, Hinton & Williams 1986). There are two popular forms used in solving this problem with a back-propagation feed-forward neural network. The first has a single hidden layer of two units, each connected to both the inputs and the output. The second form has only a single hidden unit, but also has "shortcut" connections from the inputs to the output unit. There are also other forms, often including more hidden units.

Compared with the feed-forward neural network, determining what architecture the model should use is more straightforward for the OTFM in this situation. Generally, the number of the OTFM layers should increase as the complexity of the problem increases. We found that 4 thin-film layers were sufficient to solve the XOR problem. However a 8 thin-film layer stack was used here as an demonstration of this problem solving.

Layer	Refractive Index	Thicknesses (μm)	
		Initial	Found
(Output)	4.0	-	-
1	1.2	0.8	1.278507
2	2.4	0.3	1.010522
3	1.2	0.8	2.427093
4	2.4	0.3	0.093837
5	1.2	0.8	0.798172
6	2.4	0.3	0.299938
7	1.2	0.8	0.797708
8	2.4	0.3	0.291884
(Input)	1.0	-	-

Table 8.2. Optical description of a 8-layer OTFM for solving the XOR problem. λ = 4.0μm; N_λ = 1; Δ = 0.7. The N-squared Scan Method was used.

As shown in Table 8.2, an 8-layer OTFM was used for training to solve the XOR problem. In each input row, the two input bits were encoded into the refractive indices of the first layer and the second layer respectively (see Section 7.4). Because this is a nonlinear separable problem, the other 6 layers were used to increase the self-organizing ability of the OTFM to achieve the desired computation (in comparison with a feed forward neural network, hidden layers are necessary for solving the XOR problem). The model was trained with a light wave at a single wavelength 4.0μm (N_λ = 1 as shown in Table 8.2). 0.7 was chosen as the scaling factor Δ. Using the N-squared Scan optimization method with a set of initial thicknesses as given in Table 8.2, we found an optimal set of thickness values (as in the "Found" column in Table 8.2) as a solution which corresponds to the training result as shown in Table 8.1. If the tolerance is set to 0.3, the results in the "Reflectance" column as shown in Table 8.1 are acceptable according to the "Target" column.

The training time for OTFM was 4 minutes and for a feed-forward neural network model was about 10 minutes (see the result in Table 8.1). We also used the Genetic Algorithm to train the OTFM in solving the XOR problem, and a similar result to the training done with the N-squared Scan Method was obtained.

8.2.2 Four-bit Parity Problem

The parity problem examined here has 16 four-bit binary numbers. The desired output is 0.0 if the parity of the four-bit number is even, and 1.0 if the parity is odd (Table 8.3).

Four-bit Binary Number (as Input)				Desired Output
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 8.3. Four-bit parity problem.

It is a rather difficult problem to solve, since the output must change whenever any single input digit changes (Hertz, Krogh & Palmer 1991). This parity problem is often used for testing or evaluating a connectionist model design.

The 4-bit parity problem can be solved by a feed-forward neural network as shown in Figure 8.1, employing the back-propagation learning algorithm by gradient descent. Solutions are provided by Rumelhart and McClelland (1988); and Hertz, Krogh and Palmer (1991).

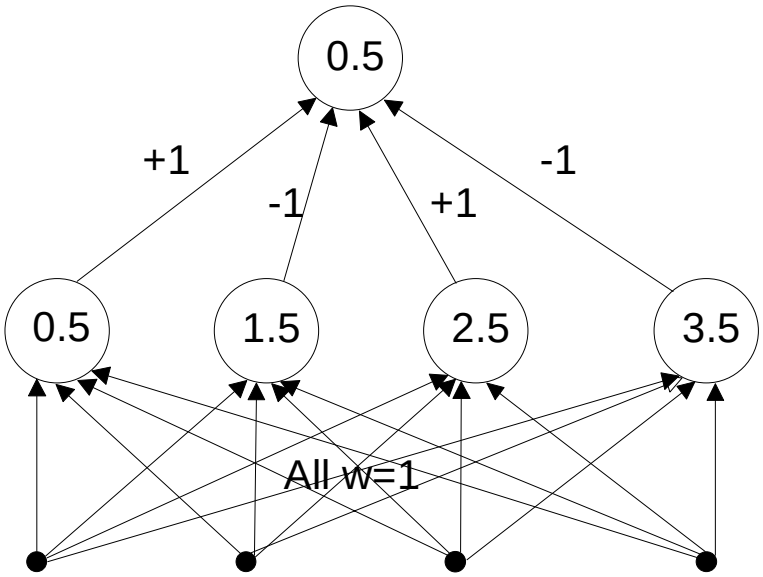


Figure 8.1. A neural network solution to solve the $N = 4$ parity problem with 0/1 threshold nodes.

Figure 8.1 shows that in this solution the weights are all either 1.0 or -1.0, and the hidden units are arranged in such a way that they count the number of inputs. The unit at the far left is on when one or more input units are on, the next comes on if two or more are on, and so on, until all the hidden units are on when all of the input units are on. Hence the first *m* hidden units are on whenever *m* input units are on. The hidden units connect alternately with positive and negative weights, in this way a hidden unit either excites or inhibits the output unit depending on whether *m* is odd or even. Rumelhart and McClelland's solution was reached after 2,825 presentations of each of the 16 patterns with a learning rate of 0.5.

Instead of the back-propagation learning by gradient descent, the OTFM's learning of this solution is by means of searching many possible thickness combinations using either the N-squared Scan Method or the Genetic Algorithm method. Each output determined by the thickness combination is measured by a merit function (see Section 7.2). If it is not matched to the minimal merit requirement (tolerance), this thickness combination is then discarded, and the model moves on to try a new one, until it satisfies the tolerance required by the system.

In this section, experiments using either the encoding methods #1 or #2 described in Section 7.4 were conducted. Encoding method #1 was used as an illustration of how the wavelengths could be used as a way of encoding the input, but in general, the encoding method #2 was preferred

since it uses the "distributed representations" approach (see Section 2.4), which is a fundamental property of a connectionist model.

By using the encoding method #1, the OTFM of 20 layers was tuned so that it produces an optical reflectance value towards the target output. This mapping process has been shown in Table 7.1: a four-bit binary input row was mapped into a decimal number according to the equation (7.5), and 16 such decimal numbers were then mapped into 16 wavelengths by using equation (7.6).

b3	b2	b1	b0	Wavelength (input)	Target	Reflectance (output)
0	0	0	0	0.100	0	0.097453
0	0	0	1	0.105634	1	0.933161
0	0	1	0	0.11194	1	0.934278
0	0	1	1	0.119048	0	0.055822
0	1	0	0	0.127119	1	0.906626
0	1	0	1	0.136364	0	0.128515
0	1	1	0	0.147059	0	0.086174
0	1	1	1	0.159574	1	0.821576
1	0	0	0	0.174419	1	0.856652
1	0	0	1	0.192308	0	0.141524
1	0	1	0	0.214286	0	0.187758
1	0	1	1	0.241935	1	0.846331
1	1	0	0	0.277778	0	0.095377
1	1	0	1	0.326087	1	0.908817
1	1	1	0	0.394737	1	0.93971
1	1	1	1	0.500	0	0.058055

Table 8.4. Training result of the OTFM employing encoding method #1 for the parity problem of 16 four-bit binary numbers. The "Reflectance" column shows the outputs from the training, which were corresponding to the targets in the "Target" column.

Table 8.4 shows the training result using encoding method #1. The wavelengths which were mapped from 16 four-bit binary strings are also shown in Table 8.4. These wavelengths were used as inputs fed into the OTFM for training. The desired targets are shown in "Target" column. After training using the N-squared Scan Method, the trained OTFM produced the reflectance values as outputs shown in "Reflectance" column. If a tolerance of 0.2 is set here, the trained

OTFM can give the correct answer to each corresponding four-bit binary string, that is if the reflectance is smaller than 0.2, then the answer is classified as the even parity, and if the reflectance is greater than 0.8, then the answer is classified as odd parity.

Layer	Refractive index	Thicknesses (μm)	
		Initial	Found
(output)	4.0	-	-
1	3.0	3.0	2.988
2	2.0	3.0	3.007
3	3.0	3.0	2.998
4	2.0	3.0	3.0
5	3.0	3.0	3.001
6	2.0	3.0	3.0
7	3.0	3.0	3.0
8	2.0	3.0	2.98
9	3.0	3.0	3.0
10	2.0	3.0	3.0
11	3.0	3.0	2.999
12	2.0	3.0	3.039
13	3.0	3.0	2.959
14	2.0	3.0	3.025
15	3.0	3.0	2.957
16	2.0	3.0	3.06
17	3.0	3.0	3.018
18	2.0	3.0	3.08
19	3.0	3.0	3.0
20	2.0	3.0	3.0
(Input)	1.0	-	-

Table 8.5. Optical description of a 20-layer OTFM using encoding method #1 for solving the four-bit parity problem. λs are given in Table 8.4, in column "Wavelength"; $N_\lambda = 16$; $\Delta = 0.4$. The N-squared Scan Method was used.

The optical description of this trained OTFM is shown in Table 8.5, where the "Found" thickness values represent a solution for this four-bit parity problem (by using this set of found thickness values, the reflectance values shown in Table 8.4 can be produced).

The above example for solving the four-bit parity problem is the only experiment where the encoding method #1 was used. As we discussed in Section 7.4, the encoding method #1 is more

limited than the encoding method #2, for instance, the wavelengths cannot be used as adjustable parameters, because they are used as inputs. The encoding method #2 does not have such limitations, therefore it is regarded as the more general approach for encoding input values (see Section 7.4). In all other experiments described in this chapter the encoding method #2 was adopted.

Layer	Refractive index	Thicknesses (µm)	
		Initial	Found
(output)	4.0	-	-
1	1.2	0.8	1.592516
2	2.5	1.6	1.599463
3	1.2	0.8	2.087996
4	2.5	1.6	1.447014
5	1.2	0.8	0.78065
6	2.5	1.6	1.602062
7	1.2	0.8	2.603325
8	2.5	1.6	1.6
9	1.2	0.8	0.8
10	2.5	1.6	1.6
11	1.2	0.8	2.595908
12	2.5	1.6	2.465077
13	1.2	0.8	2.567174
14	2.5	1.6	1.301312
15	1.2	0.8	0.79269
16	2.5	1.6	1.6
17	1.2	0.8	0.782267
18	2.5	1.6	1.6
19	1.2	0.8	0.776034
20	2.5	1.6	1.599329
21	1.2	0.8	0.770175
22	2.5	1.6	1.594742
23	1.2	0.8	2.604114
24	2.5	1.6	1.579699
25	1.2	0.8	2.744598
(Input)	1.0	-	-

Table 8.6. Optical description of a 25-layer OTFM using encoding method #2 for solving the four-bit parity problem. $\lambda = 4.4\mu m; N_\lambda = 1; \Delta = 0.5$. The N-squared Scan Method was used.

Training the OTFM using the encoding method #2 to solve the four-bit parity problem was also conducted. In this training, a 25-layer OTFM was used to obtain the desired reflectance values using a light wave at wavelength 4.4μm. 0.5 was chosen as the scaling factor Δ. As shown in Table 8.6, using the N-squared Scan optimization method with the initial thicknesses as shown in column "Initial", we found an optimal set of thickness values as shown in the "Found" column, which results in a solution of the parity problem shown in Table 8.7.

Four-bit Binary Number as Input				Desired Output	OTFM Reflectance (25 layers)	Neural Network Output node (with 4 hidden nodes)
0	0	0	0	0	0.24631	0.02224
0	0	0	1	1	0.64019	0.99903
0	0	1	0	1	0.596955	0.99902
0	0	1	1	0	0.286317	0.00921
0	1	0	0	1	0.822227	0.99903
0	1	0	1	0	0.214709	0.0092
0	1	1	0	0	0.213929	0.00918
0	1	1	1	1	0.8325	0.96059
1	0	0	0	1	0.784272	0.99903
1	0	0	1	0	0.207857	0.00921
1	0	1	0	0	0.14581	0.00918
1	0	1	1	1	0.822444	0.9604
1	1	0	0	0	0.285641	0.00918
1	1	0	1	1	0.600092	0.96021
1	1	1	0	1	0.553203	0.96004
1	1	1	1	0	0.342155	0.08257

Table 8.7. Training result for solving the four-bit parity problem.

As shown in Table 8.7, if a tolerance of 0.5 is set here, the trained OTFM can easily give the correct answer corresponding to the inputs, that is if the reflectance is smaller than 0.5, the answer is classified as even parity, and if the reflectance is greater than 0.5, the answer is classified as odd parity. Though the desired targets were not as well approximated by these reflectance values as those outputs produced by the trained neural network model (as shown in Table 8.7), it has accomplished the required computation. In fact, we can always use a nonlinear function (e.g. a sigmoid function with adjustable steepness, see Figure 2.6) to map these reflectance values to some values further closer to the desired target, as those produced by the trained neural network

model. Therefore a direct comparison of the output values produced by these two models cannot be used as a justifiable measurement of performance. Experiments were also conducted using the Genetic Algorithms, and a similar result was obtained.

The solving of the parity problem by the OTFM has shown that it has good self-organizing ability – adjusting itself to achieve the required computation. This can be done by either using a range of wavelengths as the encoded input (encoding method #1), or by using refractive indices of individual layers as the distributed inputs (encoding method #2).

8.3 PATTERN RECOGNITION

Among the most common learning tasks for connectionist models are those in the area of pattern recognition or classification. The goal is to find a set of parameter values of the model so that for instance in a neural network model, whenever a particular pattern is presented to the input nodes, the associated pattern will appear at the output nodes. The inputs (corresponding to the input nodes in a neural network model) here are associated with the individual thin-film layers in the OTFM, and similarly the outputs are associated with the reflectance values over a range of wavelengths. The recognition task is accomplished by training the OTFM, i.e. searching for an appropriate set of layer thickness values in the layer thickness search space.

8.3.1 Four Sample 5-by-5 Grid Pattern Recognition

A four sample 5-by-5 grid pattern recognition example was used for training in this experiment. As discussed in Section 7.2, the optical output of the OTFM can be taken to be the reflectance (real-value) or the reflection coefficient (complex-value). The simpler approach in terms of optical measurement of using the optical reflectance (equation (7.1)) was used as the system output for this training. Figure 8.2 shows the four sample 5-by-5 grid patterns that the model was trained to recognize.

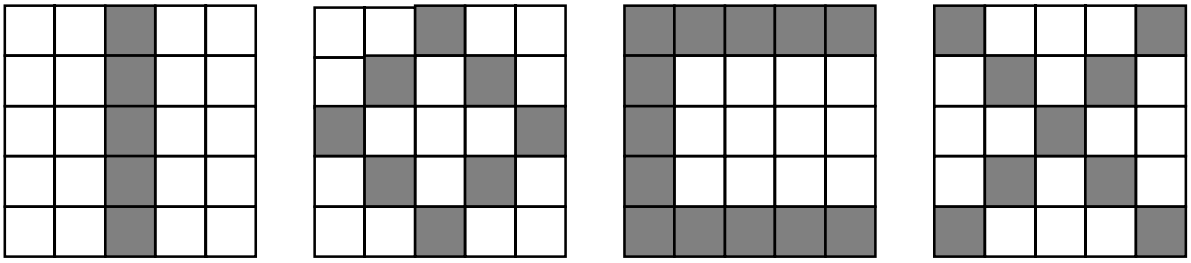


Figure 8.2. Four sample 5-by-5 grid patterns: "I", "O", "C" and "X". A shaded pixel represents 1.0, and a blank pixel represents 0.0.

A multilayer stack of 25 layers was trained for this recognition task (more layers could be used if necessary). The 25 layers were specified with values of the base refractive index n_{Bi} for each layer alternating between 1.2 for the odd-number layers and 4.0 for the even-number layers. The scaling factor Δ for the inputs was 0.4, so the inputs were encoded as small values to be added to n_{Bi} – either 0.0 or 0.4, depending on whether a pixel in the 5-by-5 grid is shaded or blank (each pixel represents either 1.0 (shaded) or 0.0 (blank) in a pattern, so 25 pixels of a pattern represent 25 digits of binary numbers).

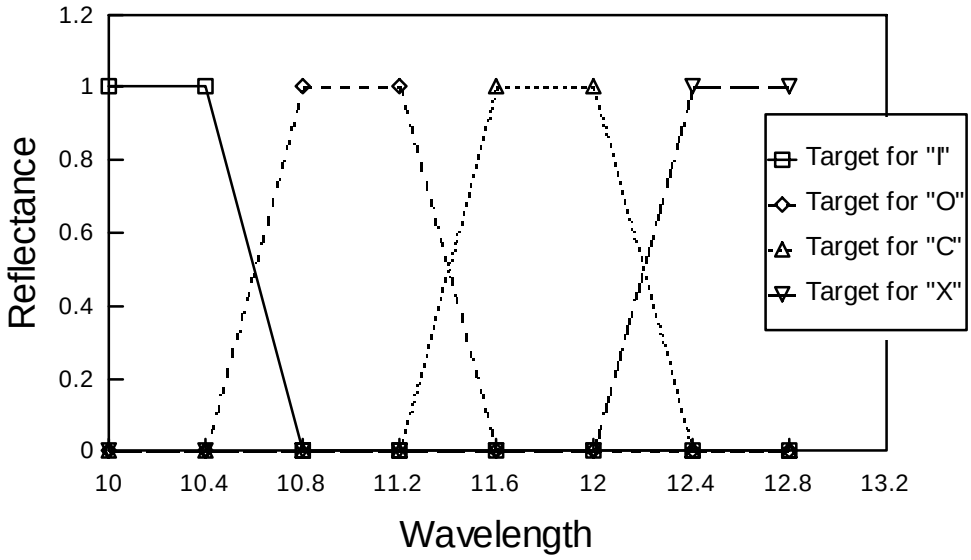


Figure 8.3. Training targets specified for four different patterns respectively, over wavelengths ranging from 10.0 to 12.8μm.

For each pattern, the OTFM was trained with 8 target reflectance values over 8 different wavelengths ranging from 10.0 to 12.8µm. Each of the 8 target reflectance values was specified differently for each pattern as shown in Figure 8.3, i.e. the target spectral response for the pattern "I" has a peak value of 1.0 in the wavelength range of 10.0 to 0.4, descends to 0.0 at a wavelength of 10.8, and then remains steady at 0.0 out to the wavelength of 12.8. The target response for the pattern "O" has peak values in the wavelength range of 10.8 to 11.2 and is zero for wavelengths below 10.4 and above 11.6. The target response for the pattern "C" has peak values for wavelengths from 11.6 to 12.0, with zero values below a wavelength of 11.2 and above a wavelength of 12.4. The target response for the pattern "X" has peak values for wavelengths between 12.4 and 12.8, with zero values for wavelengths below 12.0. The goal was to train the model to produce reflectance values as close as possible to the target reflectance values, so it would be able to distinguish these patterns to a maximum degree after training.

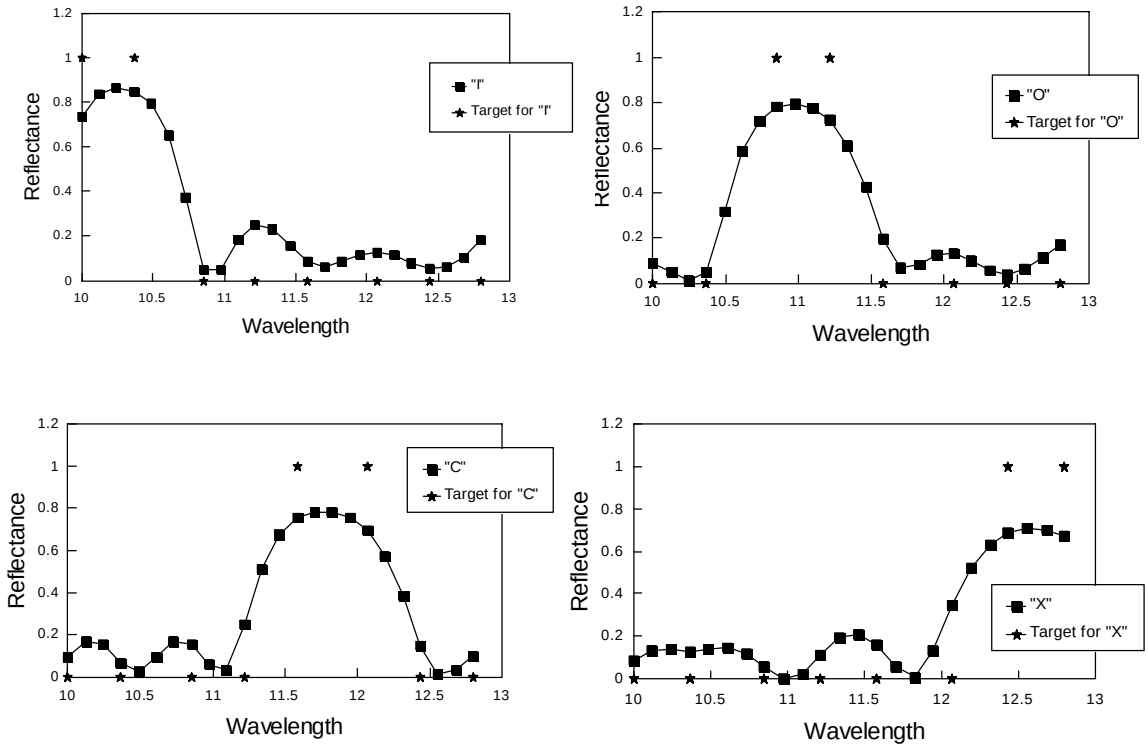


Figure 8.4. Plots of the OTFM training results for "I", "O", "C" and "X".

The training result can be seen in Figure 8.4, where the trained OTFM produces high and low values of reflectance values that approximate the specified targets. The target values which are normally specified as 1.0 (high) and 0.0 (low) were approximated with sufficient accuracy to enable the patterns to be clearly distinguished. Table 8.8 gives the optical description of the trained thin-film stack.

Layer	Refractive index	Thicknesses (µm)	
		Initial	Found
(output)	4.0	-	-
1	1.2	0.4	0.398066
2	4.0	0.3	1.531319
3	1.2	0.4	0.034815
4	4.0	0.3	0.262105
5	1.2	0.4	1.01166
6	4.0	0.3	0.174733
7	1.2	0.4	1.392043
8	4.0	0.3	0.405879
9	1.2	0.4	0.562877
10	4.0	0.3	0.212455
11	1.2	0.4	4.280041
12	4.0	0.3	0.212398
13	1.2	0.4	0.501413
14	4.0	0.3	1.605182
15	1.2	0.4	4.597433
16	4.0	0.3	1.6084
17	1.2	0.4	0.562763
18	4.0	0.3	1.621467
19	1.2	0.4	4.991588
20	4.0	0.3	0.04879
21	1.2	0.4	0.820317
22	4.0	0.3	0.024398
23	1.2	0.4	0.455675
24	4.0	0.3	0.090589
25	1.2	0.4	4.999958
(Input)	1.0	-	-

Table 8.8. Optical description of a 25-layer OTFM using encoding method #2 for solving the 4 pattern recognition problem. λ_{min} = 10.0µm; λ_{max} = 12.8µm; N_λ = 8; Scaling factor Δ = 0.4. The N-squared Scan optimization method was used.

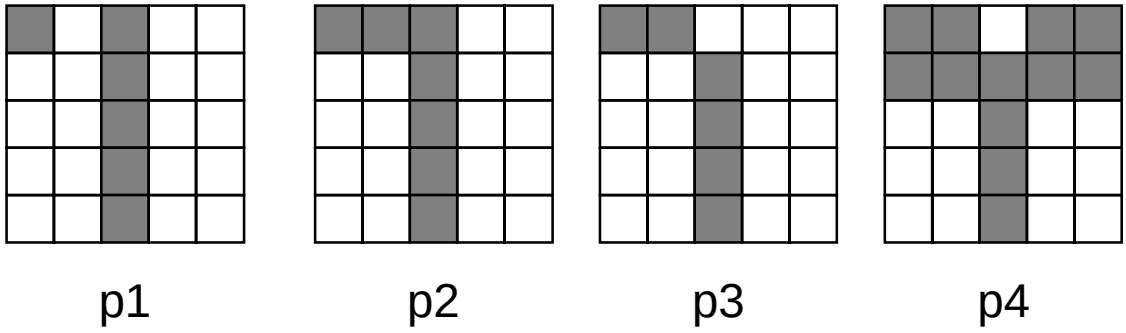


Figure 8.5. Four noisy versions of pattern “I”.

To examine the response of the trained OTFM in the presence of noise, four noisy versions of pattern “I” (as shown in Figure 8.5), p1, p2, p3, and p4, were tested on the trained model. In these four patterns, the majority of shaded and blank pixels of the pattern “I” remained unchanged. Noise was added to the pattern by changing some the original pixels of “I” from shaded to blank or from blank to shaded, with noise increasing gradually from p1 to p4: the pattern p4 is considered to be the worst pattern of “I”. The use of noise like this can test the model's robustness and how gradually its performance degrades.

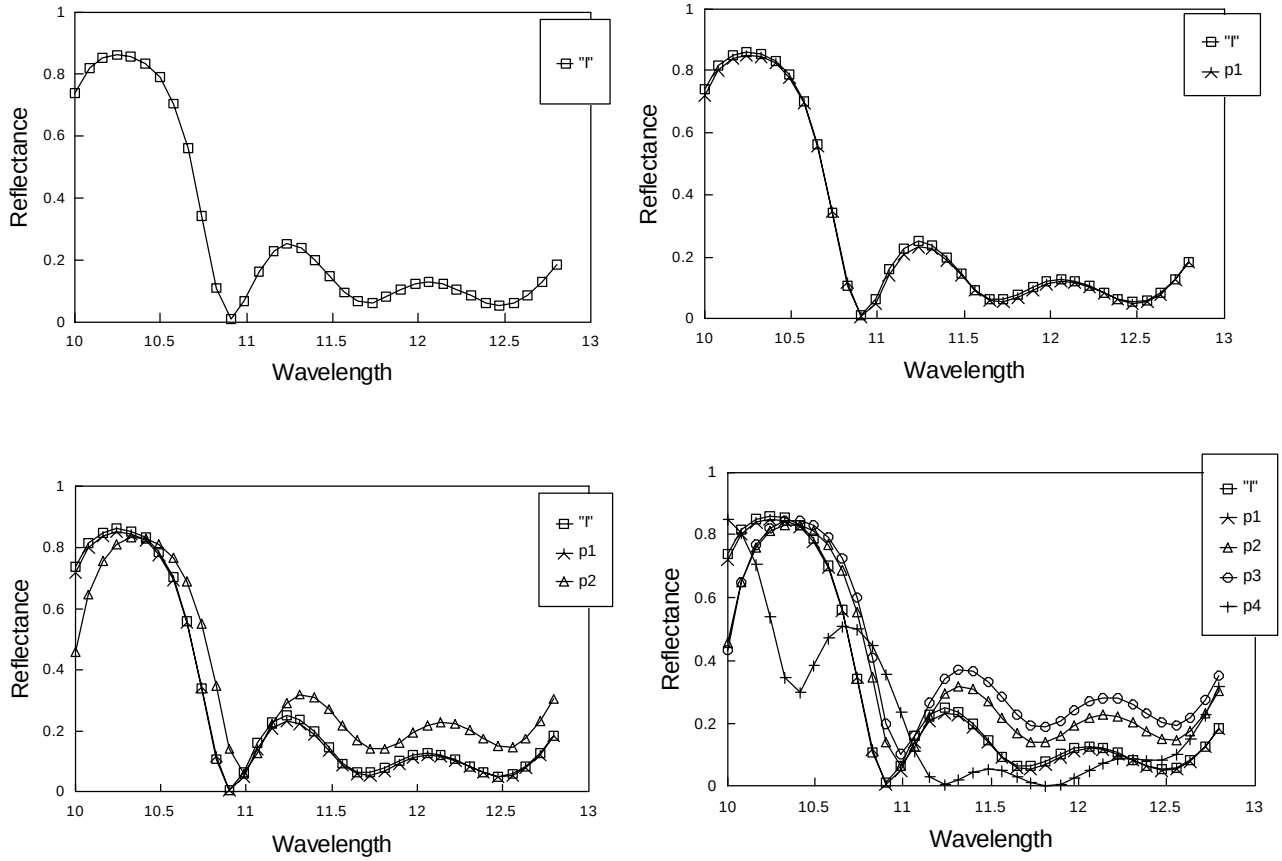


Figure 8.6. Plotted graphs of the test results on noisy versions of "I", p1, p2, p3 and p4.

The test results are shown in Figure 8.6. Depending on the tolerance required, noisy patterns of "I", p1, p2, p3 and p4 can be classified as "I" or a "wrong" pattern. For example, if the tolerance is set to 0.2, only p1 is qualified to be "I", and p2, p3 and p4 are all classified as "wrong" patterns. As the noise increases from p1 to p4, the resulting reflectance curve becomes more distant from the target reflectance curve (p4 is the worst one), indicating that the system degrades gracefully with increasing noise. Even for p4, its curve can still be seen to be closer to "I" than the reflectance pattern of "O", "C" and "X" in Figure 8.4.

The OTFM training for this experiment took 16 minutes, while a feed-forward neural network using back-propagation learning took 3 minutes to complete a satisfactory training for this four-

pattern recognition problem. However if fewer wavelengths are used for the OTFM, the training time can be reduced substantially.

To reduce training time, the OTFM was trained with the same four patterns using complex reflection coefficients specified at only two different target wavelengths (see equation (7.2)). Compared with the previous training, the performance of the OTFM showed output values even closer to the targets, and the training took only 4 minutes (since only two wavelengths were used, resulting a substantial saving of computation). A similar experiment with this approach, in which 8 patterns were used for training, is described in the following section.

8.3.2 Eight Sample 5-by-5 Grid Pattern Recognition

For the training of a pattern recognition problem involving eight sample 5-by-5 grid patterns, a computationally more efficient approach was employed than that used in the previous section. This approach uses the merit function equation (7.2). Assume that $R_0 = |j_0 + ik_0|^2$ represents the target reflectance and $R = |j + ik|^2$ represents the computed complex reflectance at a wavelength (j and k are the real and imaginary parts of a complex number). Rather than training the model by comparing R_0 and R, it can be done by directly comparing j_0 with j, and k_0 with k. Since there are now two comparisons made for each wavelength, the number of wavelengths needed in the computation is reduced.

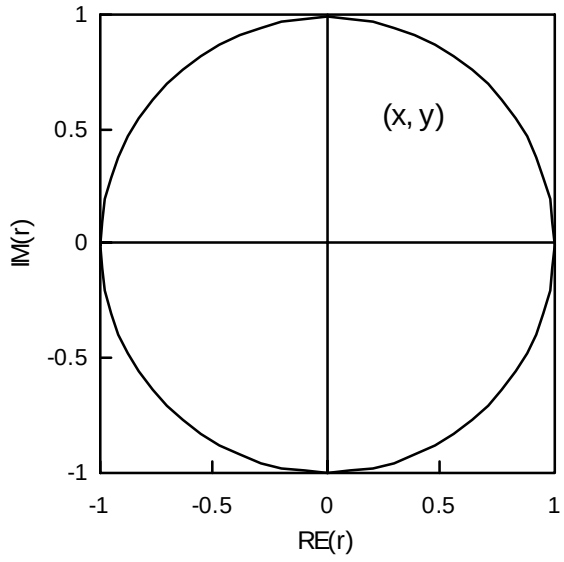


Figure 8.7. Targets can be specified within the circle where reflection coefficients (complex numbers) are distributed.

Figure 8.7 shows that a target can be specified as a point (x, y) at a particular wavelength, within the circle that has a radius of 1.0. More targets can be specified if necessary. Since there is now an additional dimension for training, fewer wavelengths are needed for the same kind of computational tasks, resulting in a significant amount of computational saving. In this section, the eight sample 5-by-5 grid pattern recognition example is used to illustrate this new training approach. The four patterns used in the previous section were again used here, plus another four patterns shown in Figure 8.8.

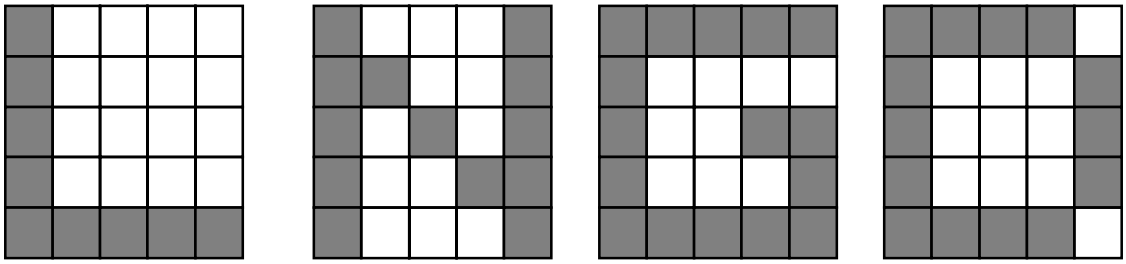


Figure 8.8. The additional four patterns used for the training of the eight sample 5-by-5 grid pattern recognition.

A 25-layer stack was also used here for the eight pattern recognition task. The 25 layers were specified with values of the base refractive index n_{Bi} for each layer alternating between 1.2 for the odd-number layers and 4.1 for the even-number layers. The refractive index scaling factor Δ for the inputs was 0.4, so the inputs were encoded as small values to be added to n_{Bi} – either 0.0 or 0.4, depending on whether a pixel in the 5-by-5 grid is shaded or blank. The stack was trained at a single wavelength of 8.0μm (note that in the experiment of the four pattern recognition 8 wavelengths were used, see the previous section). The specified target reflection coefficient for each pattern “I”, “O”, “C”, “X”, “L”, “N”, “G” and “D”, was distributed evenly around the perimeter of the unit circle of the complex plane, as shown in Figure 8.9. Note that only one target point was used for training for a particular pattern. Any other points within the circle could have been chosen as target points, and it is a subject of further research to determine how best to distribute target points on the complex plane. Before the training started, the thickness of each layer was initialised with the same value of 0.3, then the N-squared Scan optimization method was used for this training.

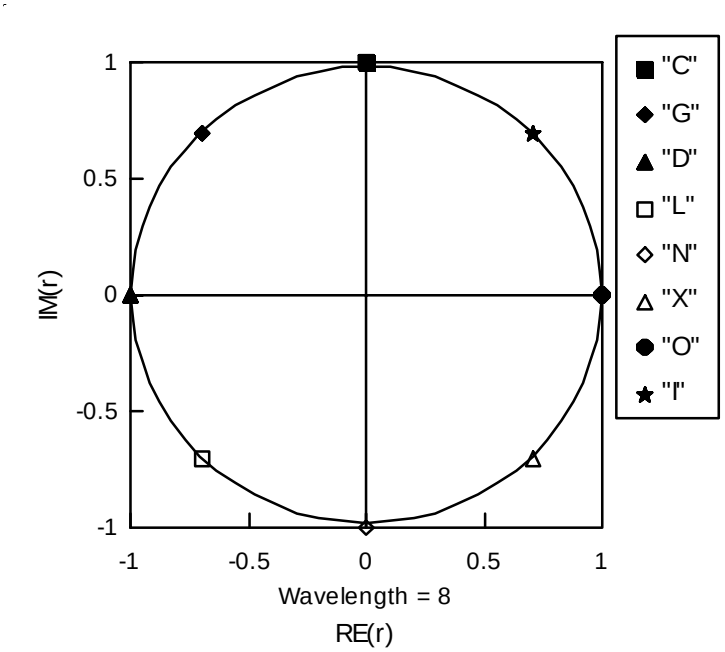


Figure 8.9. Targets specified for 8 different patterns.

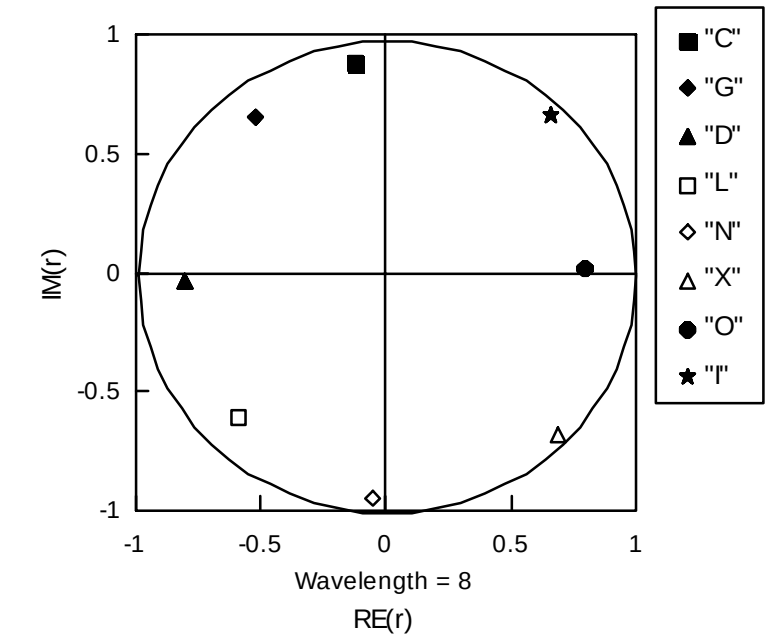


Figure 8.10. Training result for the 8 pattern recognition problem.

After training the 25-layer stack for the specified targets, the stack yielded the reflection coefficients for the 8 patterns shown in Figure 8.10. Each of the resulting reflection coefficient

points is the closest point to its corresponding target. When a new input pattern is presented to the trained OTFM, the distance to the nearest target reflection coefficient can be calculated to determine which pattern is "closest" to the input pattern. Table 8.9 shows the optical description of the trained OTFM for this experiment.

Layer	Refractive index	Thicknesses (μm)	
		Initial	Found
(output)	4.0	-	-
1	1.2	0.3	1.438981
2	4.1	0.3	0.523892
3	1.2	0.3	0.397019
4	4.1	0.3	0.212129
5	1.2	0.3	2.678683
6	4.1	0.3	0.249907
7	1.2	0.3	0.274978
8	4.1	0.3	0.315715
9	1.2	0.3	0.147542
10	4.1	0.3	1.223734
11	1.2	0.3	0.286095
12	4.1	0.3	0.248752
13	1.2	0.3	0.30495
14	4.1	0.3	1.343421
15	1.2	0.3	0.324305
16	4.1	0.3	0.339666
17	1.2	0.3	3.237168
18	4.1	0.3	0.247052
19	1.2	0.3	0.30255
20	4.1	0.3	1.289482
21	1.2	0.3	1.605683
22	4.1	0.3	0.327578
23	1.2	0.3	0.296175
24	4.1	0.3	0.293578
25	1.2	0.3	0.000502
(Input)	1.0	-	-

Table 8.9. Optical description of a 25-layer OTFM using encoding method #2 for solving the eight sample 5-by-5 grid pattern recognition problem. λ = 8.0μm; N_λ = 1; Scaling factor Δ = 0.4. The N-squared Scan optimization method was used.

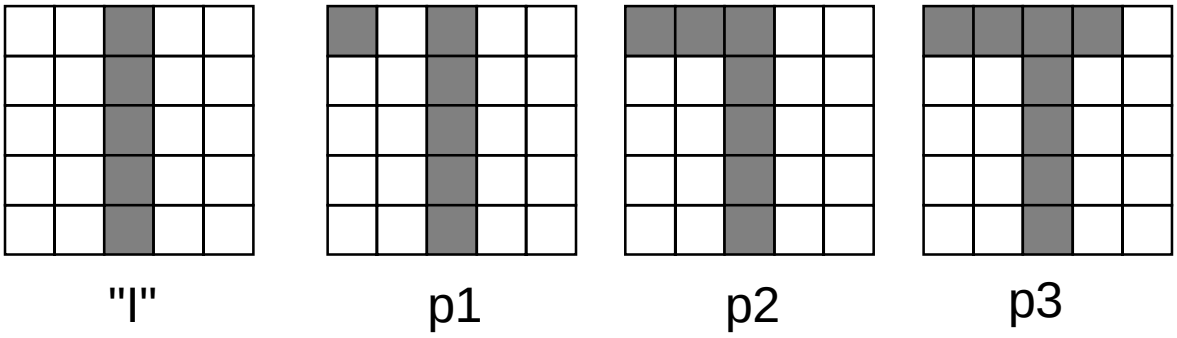


Figure 8.11. Noise increasing steadily from “I”, p1, p2 and p3.

Noisy versions of pattern “I” were also used to test how the model deals with noise. As shown in Figure 8.11, additional shaded pixels as noise have been added to the “I” pattern to create the noisy patterns of “I”.

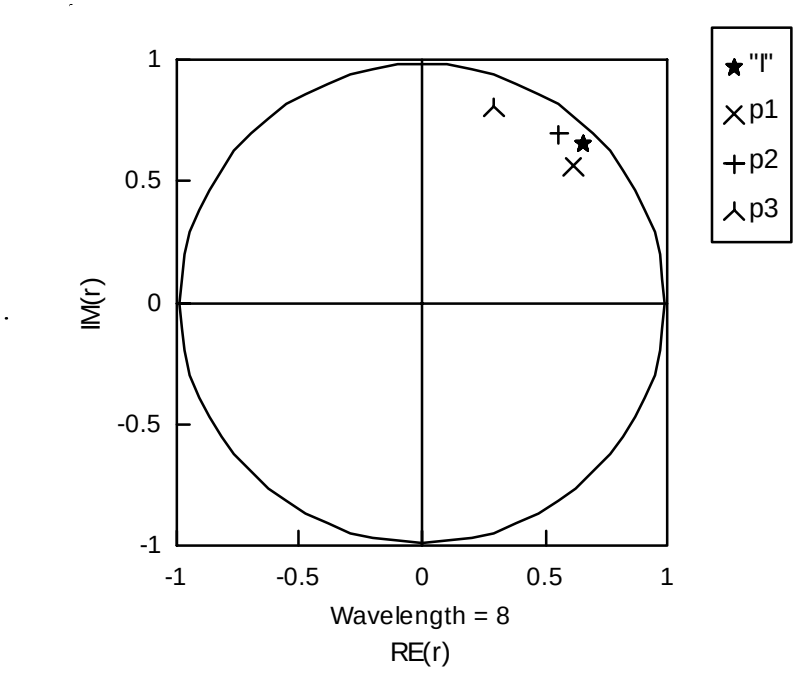


Figure 8.12. Test result on noisy versions of “I”.

The result of applying these noisy input patterns to the trained OTFM is shown in Figure 8.12. As noise increased, the resulting reflection coefficient points moved away gradually from the reflection coefficient point of the original “I” pattern point. Thus it can be seen that the trained

OTFM degraded gracefully with addition of noise to the inputs.

With only a single wavelength used, the OTFM's performance nearly matched the feed-forward neural network model in terms of training time, as well as the classification error rate. The trained OTFM was also robust and showed a graceful degradation, when adding to the system with increasing noise.

8.4 REAL-WORLD PROBLEMS

In previous sections we have only used artificial data sets, which use only binary bits as inputs, and therefore they may not be representative of real-life examples. In this section we selected problems that are typical of many real-world applications that deal with uncertainty, such as contradictory or missing information, noise, etc (see Section 7.6). These data sets normally use continuous-valued data. We chose data sets that tend to be relatively small, but have been previously analysed in the literature, so that comparison can be easily made. We consider three real-world application problems that include:

a) the prognostic evaluation of breast cancer recurrence;
b) the discrimination of three types of iris plants based on their physical features; and
c) the gas furnace time series prediction.

8.4.1 Breast Cancer Prognosis

A data set for evaluating the prognosis of breast cancer recurrence was analysed by Michalski (Michalski *et al.* 1986). It contains 286 descriptions of female patients, classified as either developing or not developing a recurrence of breast cancer after a five-year period following the first surgery. The data consists of 2 output classes for a total of 286 samples, with 9 attributes. The data were found to be inconsistent, meaning that some patients having exactly the same description were classified differently. There are also 9 missing attribute values. Such situations place an extra burden on the learning system. Since the data use descriptive symbolic language, samples need to be converted into numerical values and can then be encoded into the learning system. For example, with the attribute "age", value "10-19" is converted to 0.1, and others like "90-99" is converted to 0.9, to reflect their relative magnitude. In other cases, conversions are

done in an unordered fashion to avoid generating biased data. For example, the attribute "menopause" have three different values, "lt40", "ge40" and "premeno". The data can be coded into variables shown in Table 8.10.

Attribute	Variables
lt40	1 0 0
ge40	0 1 0
premeno	0 0 1

Table 8.10. Conversion of input attributes.

All the data were normalised in the range of 0.0 to 1.0. There are 9 missing attribute values in the breast cancer data set. We replaced each missing value with the means of values for all the samples of that attribute (see Section 7.6).

Following the sampling method used by Michalski (1986), 70% of examples were randomly selected for training and the remaining 30% were used for testing. This process was repeated 4 times.

For each randomly selected training set, the thin-film stack was then trained using the Genetic Algorithm optimization method, with 30 thin-film layers, but only 12 of the layer thickness values were encoded as a chromosome with length 180. The population size was from 200 to 400, with a crossover probability 0.6 and mutation probability 0.033. Table 8.11 gives a detailed description of the configuration of this GA-Based OTFM for the 4 different experiments using 4 randomly selected data sets.

GA-Based OTFM's Parameters	Sample #1	Sample #2	Sample #3	Sample #4
Population Size	200	400	300	300
Chromosome Length	180	180	180	180
Crossover Probability	0.6	0.6	0.6	0.6
Mutation Probability	0.033	0.033	0.033	0.033
Number of Layers	30	30	30	30
Number of Wavelengths	2	2	2	2
Lowest Wavelength	5.0	5.0	5.0	5.0
Highest Wavelength	5.8	5.8	5.8	5.6
Scaling Factor Δ	0.5	0.5	0.5	0.5

Table 8.11. GA-Based OTFM's parameter configuration for the 4 different experiments using 4 randomly selected breast cancer data sets.

It must be pointed out that these 4 experiments explored only “the tip of the iceberg” of the parameter space, considering the possible configurations in the system parameter space. More experiments are needed to further explore different configurations. After training, we tested the trained GA-Based OTFM on the corresponding test data set (30% out of total data), which was the remaining data after the random selection of the training data (70% out of total data). The test results were shown in the Table 8.12. We have not shown the found optimal set of layer thickness values as the solution for these 4 experiments (they can be easily reproduced).

Sampled Data	Correct classification rate on each test data set
randomly selected sample #1	62.8%
randomly selected sample #2	59.3%
randomly selected sample #3	59.3%
randomly selected sample #4	65.1%

Table 8.12. Test result on each breast cancer test data set.

As shown in Table 8.12, the average of these 4 correct classification rates on the test data set is 61.5%, slightly below the 64%, the correct prognosis by human experts, and also lower than 66% which is produced by an inductive learning system AQ15 reported by Michalski (1986). Since the original data are symbolic attributes, they must be converted into numerical ones before feeding them into the OTFM as input data for training. This might cause difficulty cross-validating our results with other researchers' work, since the conversion could be done differently, and it was also possible to introduce bias in the converted data. In this experiment, training and testing data were partitioned and generated by using the same resampling method as Michalski's. The trained OTFM shows it has learned a mapping relationship within the data, though its performance is not as good as that produced by the AQ15 reported by Michalski (1986).

In the work reported by Weiss and Kapouleas (1989), the best result using this cancer data set was obtained by training a back-propagation neural network model with zero hidden nodes. The error rate obtained on the training data set was 0.236. This suggests that it is likely that whatever regularity pattern exists in the data, it is linear (that is why using hidden nodes does not necessarily improve the training result), so the OTFM, which in most cases is well suited to learning nonlinear mapping tasks, may not necessarily be best suited to this type of problems.

8.4.2 Iris Data Classification

The iris data was first used by Fisher (Fisher 1936), and is still frequently used today as a standard discriminant analysis example (Weiss & Kapouleas 1989). The data set contains three classes of iris plants that are to be discriminated using four continuous-valued features, petal length, petal width, sepal length, and sepal width, that represent physical characteristics of the plants. There are 150 instances in the data set, with 50 instances for each class. One class is linearly separable from the other two, while those other two are not linearly separable from each other.

Each of the four features for the iris data set has a different range of values, so we used a separate scaling factor Δ for each feature and multiply the input value by the appropriate value of Δ and add the product to the base value of the layer refractive index. The Δ scaling factor for this experiment were 0.03, 0.08, 0.03, 0.04 for each of the four features respectively (we later tried just a single Δ value for all the four inputs, and the OTFM performed equally well).

The training approach was still the same as for the eight sample 5-by-5 grid pattern recognition (using merit function (7.2)), but three wavelengths were used for training on each pattern, instead of just one. Using extra wavelengths provides additional discriminatory power, as the wavelength parameter may be considered to offer an extra dimension for training. Since the 2nd and 3rd classes are not linearly separable, it is necessary to train the model on more wavelengths (dimensions) to discriminate these two classes. The thin-film stack consisted of 4 layers, with values of the base refractive index n_{Bi} for the 1st and 3rd layer being 1.2, and the 2nd and 4th layer being 4.1. The training was carried out over three wavelengths, 5.0, 5.3 and 5.6µm. The target reflection coefficients (real and imaginary components) were {0.2, 0.2}, {0.3, 0.3} and {0.4, 0.4} for class 1, and {-0.5, 0.1}, {-0.4, 0.3} and {-0.3, 0.4} for class 2, and {0.1, -0.5}, {0.3, -0.4} and {0.4, -0.3} for class 3, as shown in Figure 8.13.

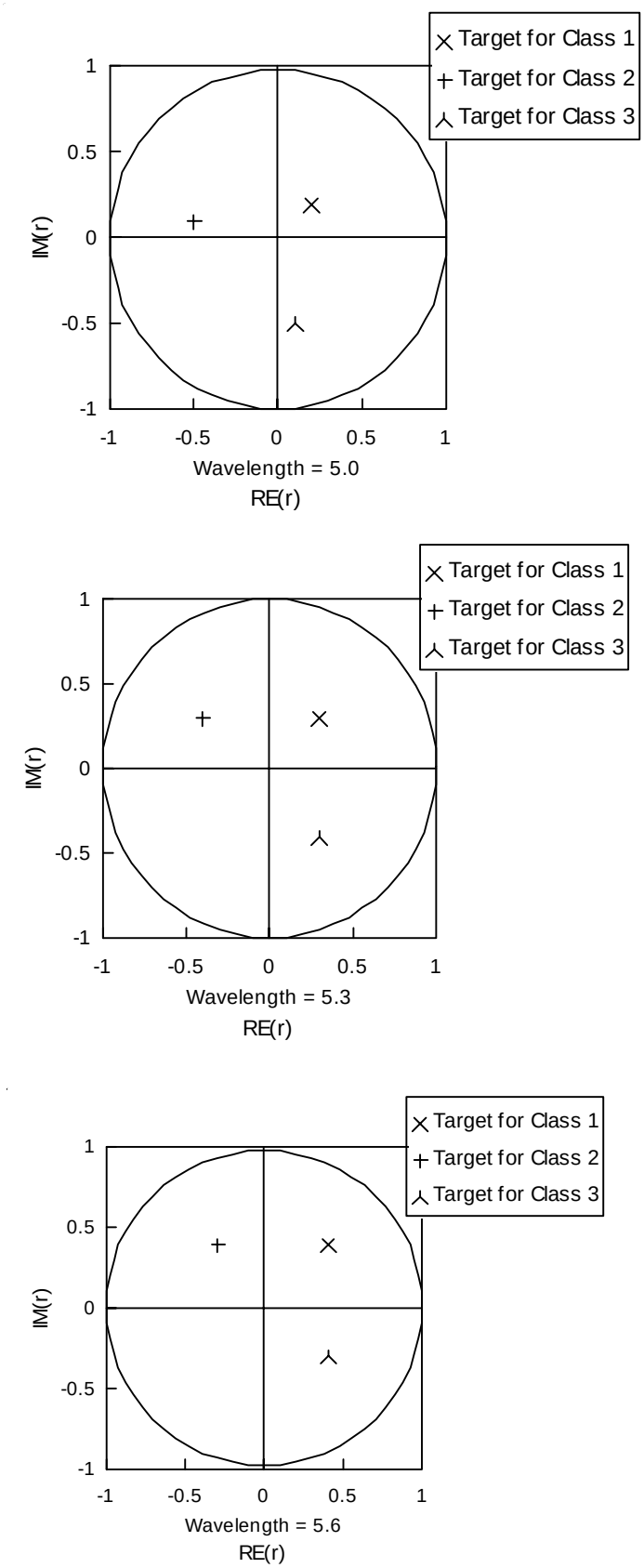
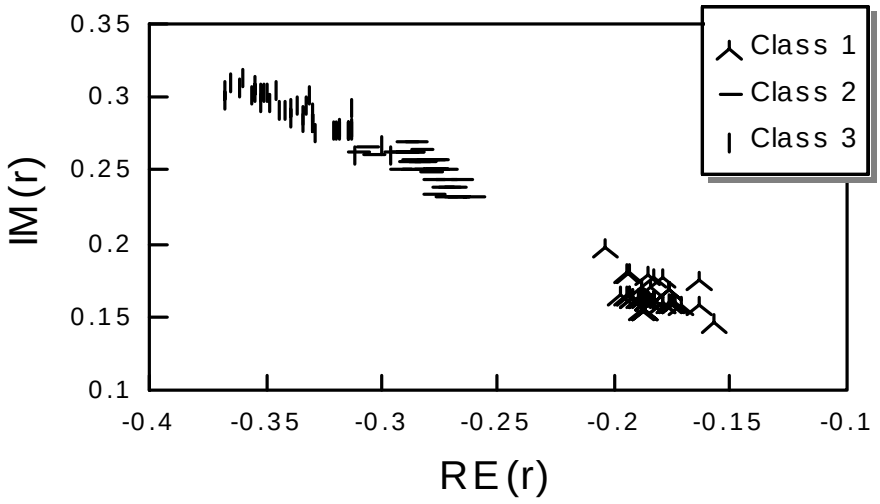
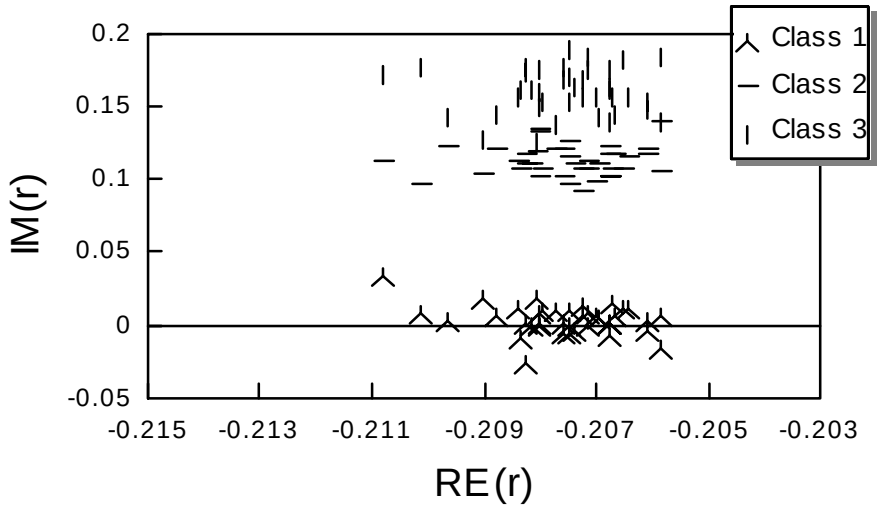


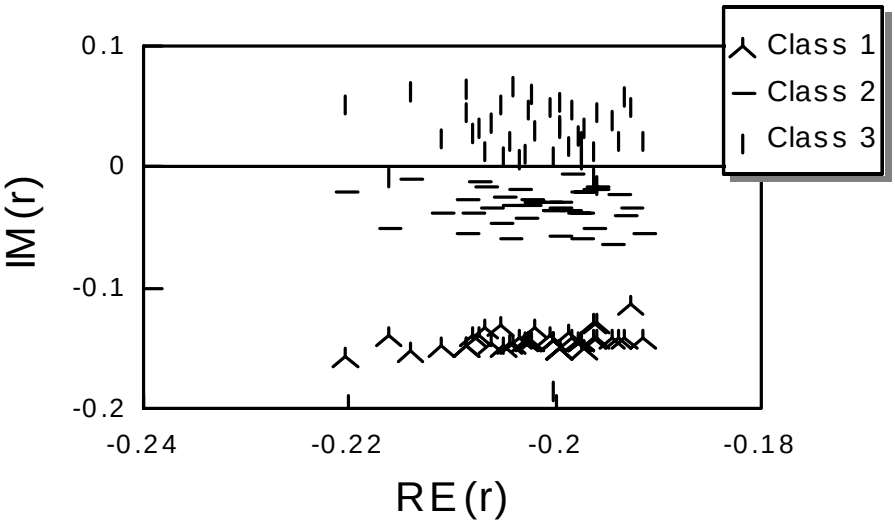
Figure 8.13. The target reflection coefficient points for the iris classification at three different wavelengths.



a) Training result at wavelength = 5.0µm.



b) Training result at wavelength = 5.3µm.



c) Training result at wavelength = 5.6µm.

Figure 8.14. The training result with 120 iris training examples.

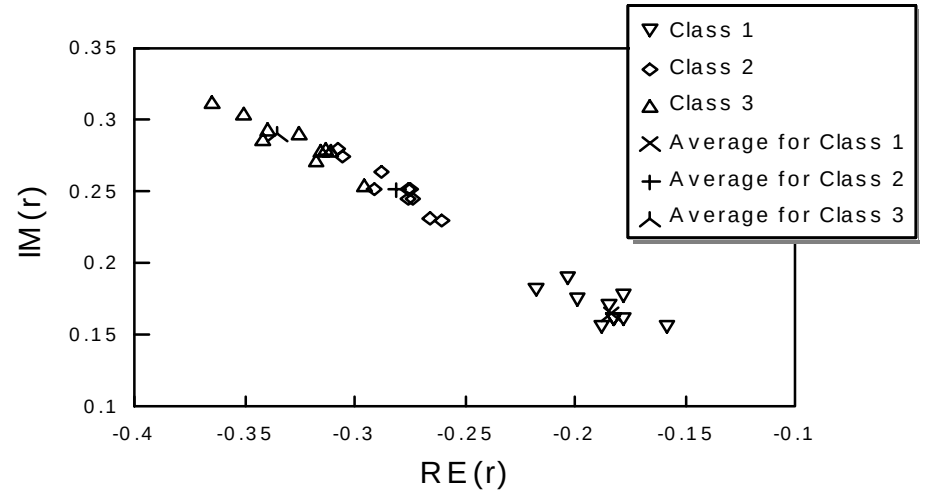
The 4-layer stack was trained to move reflection coefficient points as close as possible to the three target points over the three different wavelengths. The goal was to discriminate them to the maximal extent. As shown in Figure 8.14, class 2 and class 3 always have a number of points overlapped into each other's regions, but class 1 is completely separated from the other two classes.

For the iris classification problem, a 4-layer thin-film stack was sufficient to achieve satisfactory classification result. Its optical description is shown in Table 8.13.

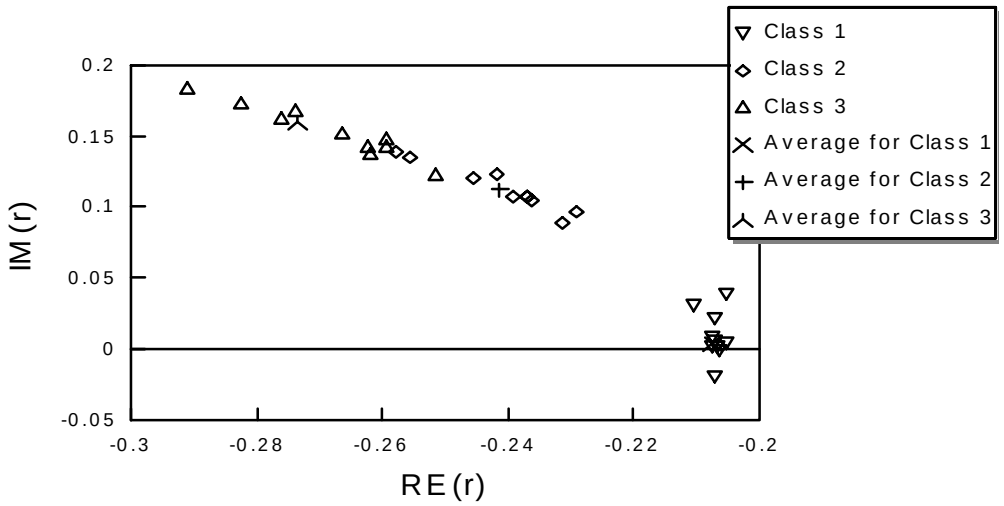
Layer	Refractive index	Thicknesses (μm)	
		Initial	Found
(Output)	4.0	-	-
1	1.2	0.5	0.146978
2	4.1	1.1	0.775223
3	1.2	0.5	0.426208
4	4.1	1.1	0.700982
(Input)	1.0	-	-

Table 8.13. Optical description of a 4-layer OTFM using encoding method #2 for solving the iris classification problem. λ_{min} = 5.0μm, λ_{max} = 5.6μm, N_λ = 3, Scaling factors Δ are 0.03,0.08,0.03 and 0.04 respectively. The N-squared Scan optimization method was used.

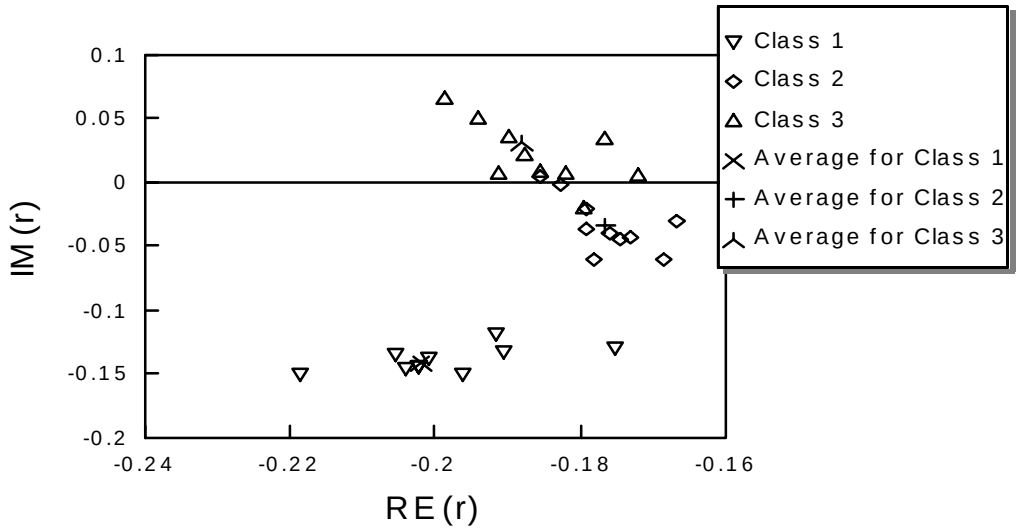
After the OTFM was trained with 120 examples from the iris data set, it was tested with the remaining 30 novel test examples. A “winner-take-all” approach was adopted (Hart 1992) that first takes the average value of all the points for each training class at three different wavelengths. During testing this value is compared with the actual output; the one having the minimum difference is chosen as the classified type. Figure 8.15 shows the test results at the three different wavelengths.



a) Test result at wavelength = 5.0µm.



b) Test result at wavelength = 5.3µm.



c) Test result at wavelength = 5.6µm.

Figure 8.15. OTFM test results on 30 novel iris test examples.

The OTFM stack classified 2 out of the 30 test examples incorrectly in this experiment. When all the 150 iris examples were presented, and 5 were classified incorrectly. The trained model has shown satisfactory classification on the training data set, and also generalized well on novel data from the test data set.

Ripley (1993), Weiss and Kapouleas (1989) reviewed some of the classification results on the iris data using different methods. The best method makes 3 misclassifications out of 150 examples. A feed-forward neural network with 4 hidden nodes classifies 5 wrongly out of 150. The trained OTFM of the above experiment and the feed-forward neural network model tested in this research had the same classification result, 5 out of 150 incorrect. It is interesting to note that the same 5 examples which were classified incorrectly by the feed-forward neural network model were also classified incorrectly by the OTFM. This is doubtless associated with the fact that in both training cases, the same training examples were used: both models reached the same conclusion about the internal relationships among the training data, then classified and generalized in the same way on the test data as well.

Using the Genetic Algorithm for training the OTFM to solve the iris classification problem was also conducted with a 4-layer stack. The 4 layers were specified with values of the base refractive index for each n_{Bi} alternating between 1.2 and 2.5. The training was again carried out for the optical wavelength values 5.0,5.3 and 5.6μm, and the target reflection coefficient values used were the same as those shown in Figure 8.13. Table 8.14 shows the Genetic Algorithm configuration used for this training. Chromosome length 60 was chosen, since only 4 layer thicknesses were mapped into a binary string (i.e. 15 bits for each layer thickness). After 6 generations, with initial population size 300, crossover probability 0.6 and mutation probability 0.033, the training had the effect of moving the response reflection coefficient points as close as possible to the 3 target reflection coefficients in an effort to separate the collection of output reflection coefficient values as much as possible.

Population Size	300
Chromosome Length	60
Crossover probability	0.6
Mutation Probability	0.033
Generation Runs	6

Table 8.14. Genetic Algorithm configuration for solving the iris classification.

Layer	Refractive Index	Found Thicknesses (μm)
(Output)	4.0	-
1	1.2	0.02373
2	2.5	2.441406
3	1.2	3.108105
4	2.5	0.002539
(Input)	1.0	-

Table 8.15. Optical description of a 4-layer OTFM using encoding method #2 for solving the iris classification problem. λ_{min} = 5.0μm, λ_{max} = 5.6μm, N_λ = 3, Scaling factors Δ is 0.1 for all the input attributes. The Genetic Algorithm optimization method was used.

The optical description of the GA training for solving the iris classification problem is given in Table 8.15. After training, the same "winner-take-all" approach used in the previous training for solving the iris classification was again used[1].

[1]i.e. Find the average values of all the output points from the training phase. When testing, compare the output with these three values, and take the closest one as the corresponding classified category.

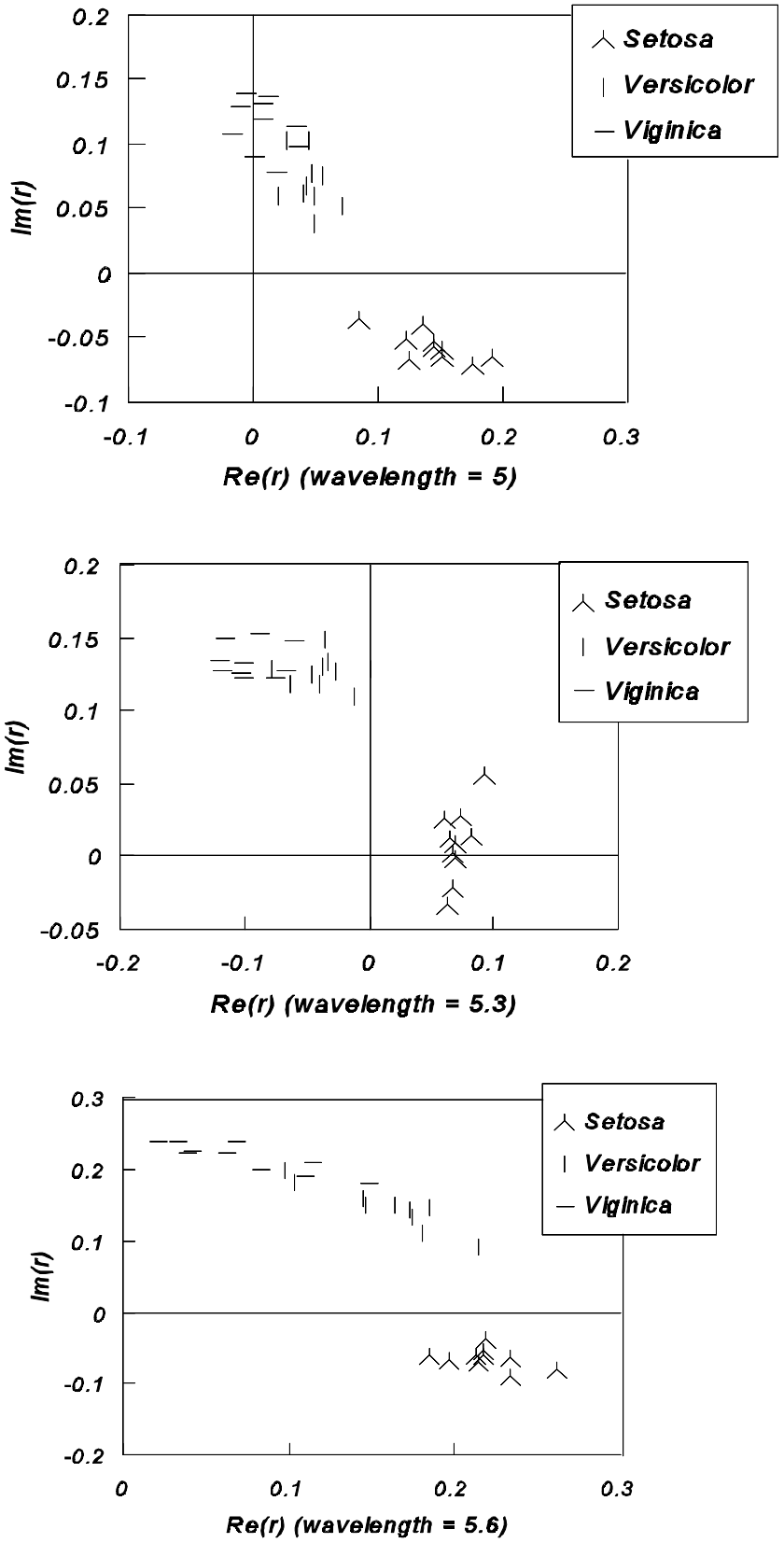


Figure 8.16. Test results on 30 test examples (over 3 wavelengths, 5.0,5.3 and 5.6μm) for solving the iris classification using GA.

Figure 8.16 shows the output reflection coefficient points of 30 test examples (note that we use the real class name here, i.e. “setosa” for class 1, “versicolor” for class 2 and “viginica” for class 3). Out of 30 novel examples only 3 were misclassified by the trained OTFM. When the system was given all 150 examples, 8 were misclassified.

We also conducted an experiment on a 25-layer stack using the GA method, with the same initial setup as those used for the 4-layer stack (see Table 8.15). A comparison of the training results is shown in Table 8.16. The 4-layer stack took only 3 minutes to converge at a satisfactory solution and it took only the first GA generation, while the 25-layer stack took much longer, about 240 minutes to converge at a similar solution after 23 generations. One possible explanation is that compared with the search space for the 4-layer stack, there were more learning parameters (i.e. 25 thickness values) for the 25-layer stack, resulting in a larger search space where a large number of local minima existed. The GA method had to search in a much larger search space, and spent more effort to step out of those local minima, in order to reach a satisfactory solution.

No. Of Layers	Converged at generation #	Time taken to converge (minutes)	Misclassification on the training set	Misclassification on the test set
4	1	3	5	3
25	23	240	3	4

Table 8.16. Comparison of the training results of the 4-layer stack and the 25-layer stack respectively, between two GA runs.

Although compared with the 4-layer stack, the 25-layer stack had a better classification on the training set after converging to a satisfactory solution, its classification performance on the novel examples from the test data set got worse. This can be seen as a problem of overfitting (Hertz, Krogh and Palmer 1991), as illustrated in Figure 8.17, where the fit is perfect on the “training set” (x's), but is likely to be poor on a “test set” represented by the circle. This is because the system is trying to follow all the small details or noise, and might fit well with the training set, but will behave poorly on the test data set.

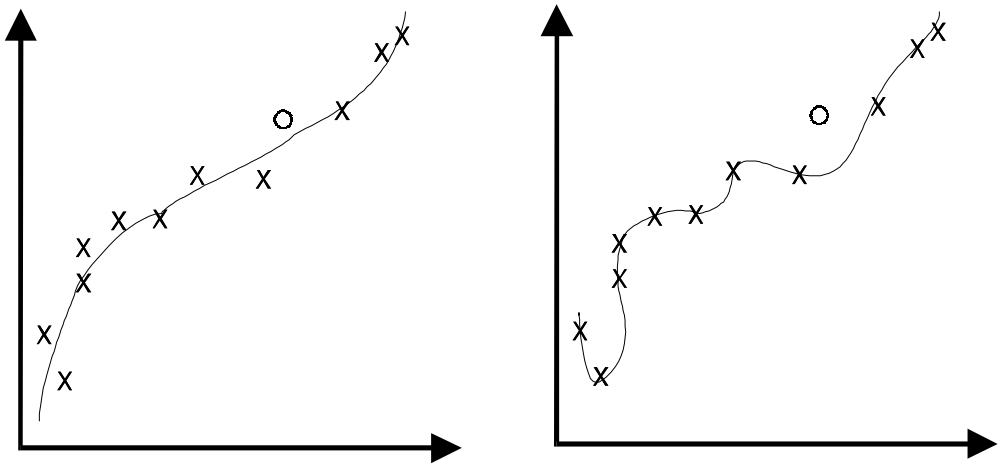


Figure 8.17. (a) A good fit to noisy data. (b) Overfitting of the same data.

8.4.3 Gas Furnace Time Series Data

In this experiment, we consider the qualitative modelling of a dynamical process using a well-known example given by Box and Jenkins (1970). It is a gas furnace system where air and methane are combined to form a mixture of gases containing CO_2 (carbon dioxide). Air fed into the gas furnace is kept constant, while the methane feed rate can be varied in any desired manner. After that, the resulting CO_2 concentration is measured in the off-gases at the outlet of the furnace. The time series used for prediction purposes consists of 292 pairs of observation: the methane gas feed rate and the CO_2 concentration in the off gases recorded at intervals of 9 seconds (see Figure 8.18).

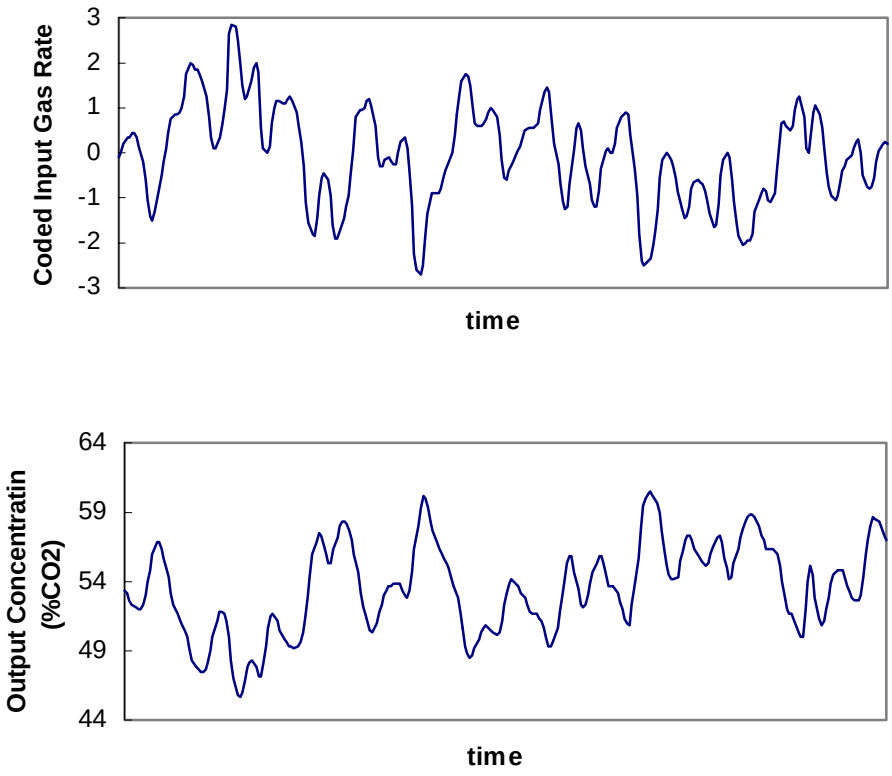


Figure 8.18. Input gas rate and output CO_2 concentration from a gas furnace.

We can consider the gas feed rate *u(t)* as a single input and the CO_2 concentration *y(t)* as an output of this dynamical process. *t* denotes the time.

In order to train the OTFM to produce an output as close as possible to the *y(t)* (time series prediction curve), we first considered two variables *y(t-1)* and *u(t-1)* as input attributes. All the 292 such data pairs were used for this prediction task. We then used a 2-layer stack to encode these two input variables into the refractive indices of the 1st and 2nd layers (using encoding method #2, see Section 7.4). The optical description of such a 2-layer stack is shown in Table 8.17.

Layer	Refractive Index	Found thicknesses (μm)
(Output)	4.0	-
1	1.4	0.61322
2	3.2	0.17041
(Input)	1.0	-

Table 8.17. Optical description of a 2-layer OTFM for solving the gas furnace time series prediction. $\lambda = 5.0\mu m$; $N_\lambda = 1$; Scaling factors Δ were 0.01 for *y(t)* and 0.1 for *u(t)* respectively. Found thicknesses were obtained by using the Genetic Algorithm.

Training was carried out at a single optical wavelength of 5.0μm. The target reflectance values are *y(t)*s as given in the 292 training data pairs, which have been scaled down to be in the range 0.0 to 1.0. The Genetic algorithm was used for this training, with a population size of 500, a chromosome length of 50 which can be interpreted as 2 thickness values, crossover probability 0.6 and mutation probability 0.033. After 39 generations, the population converged to a solution which can be interpreted as the two found layer thickness values, shown in the column of "Found Thicknesses" of Table 8.17.

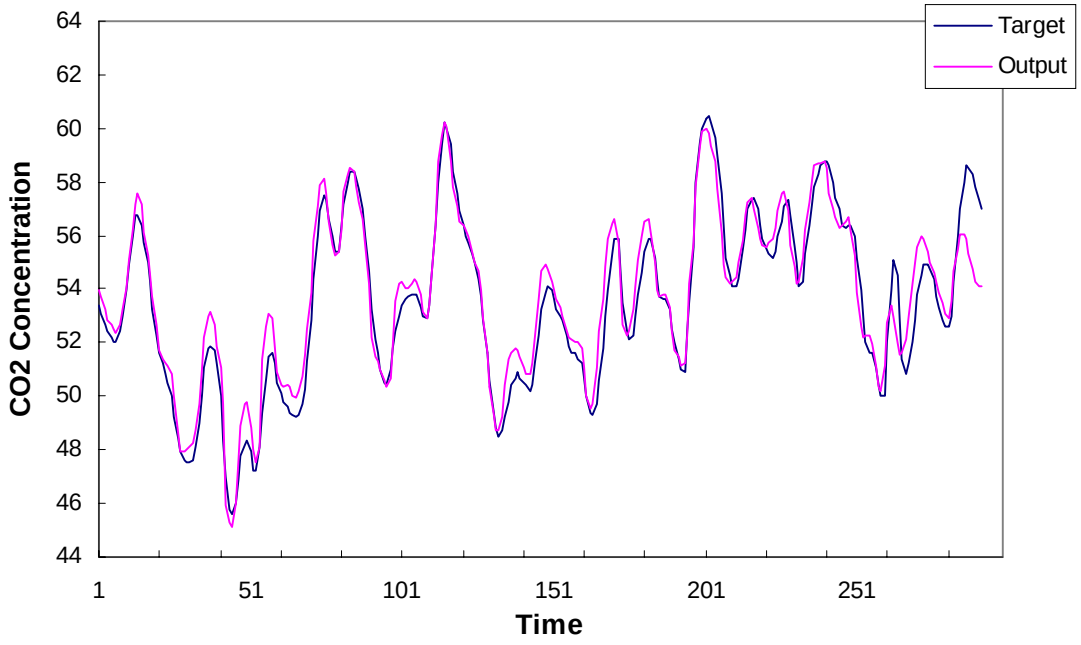


Figure 8.19. Training result of a 2-layer stack using all the 292 data pairs. The Mean Squared Error (MSE) was 0.837.

Figure 8.19 shows the training result. The Mean Squared Error (MSE) over all the 292 training data was 0.837. Optimizing on the only two thickness values (i.e. only two learning parameters), the trained OTFM was capable of accomplishing a good approximation to the overall prediction trend, though the last 6 predictions were not as close as the other predictions.

In order to further investigate the prediction performance of the OTFM training on this data set, we considered 4 variables, *y(t-1)*, *y(t-2)*, *u(t-1)* and *u(t-2)*, as input attributes. We divided the 292 data pairs into a training data set and a test data set: 250 were used for training and the remaining 42 were used for testing. A 4-layer thin-film stack was used for this task (the minimum number of layers required was 4, because 4 input attributes must be encoded into the refractive indices of the 4 layers of the thin-film stack). Its optical description, as well as the found optimal set of layer thickness values are given in Table 8.18.

Layer	Refractive Index	Found thicknesses (μm)
(Output)	4.0	-
1	1.4	0.543683
2	3.2	0.247239
3	1.4	0.457652
4	3.2	0.487074
(Input)	1.0	-

Table 8.18. Optical description of a 4-layer OTFM for solving the gas furnace time series prediction. λ = 12.0μm; N_λ = 1; Scaling factors Δ were 0.01 for *y(t)* and 0.1 for *u(t)* respectively. Found thicknesses were obtained by using the Genetic Algorithm.

Training was carried out at a single optical wavelength 12μm, and the target reflectance values were those of *y(t)* as given in the 250 training data pairs, which have been scaled down to be within the range 0.0 to 1.0. The Genetic Algorithm approach was used for this training, with a population size of 400, a chromosome length of 100 (encompassing the 4 layer thickness values), crossover probability 0.6 and mutation rate 0.033.

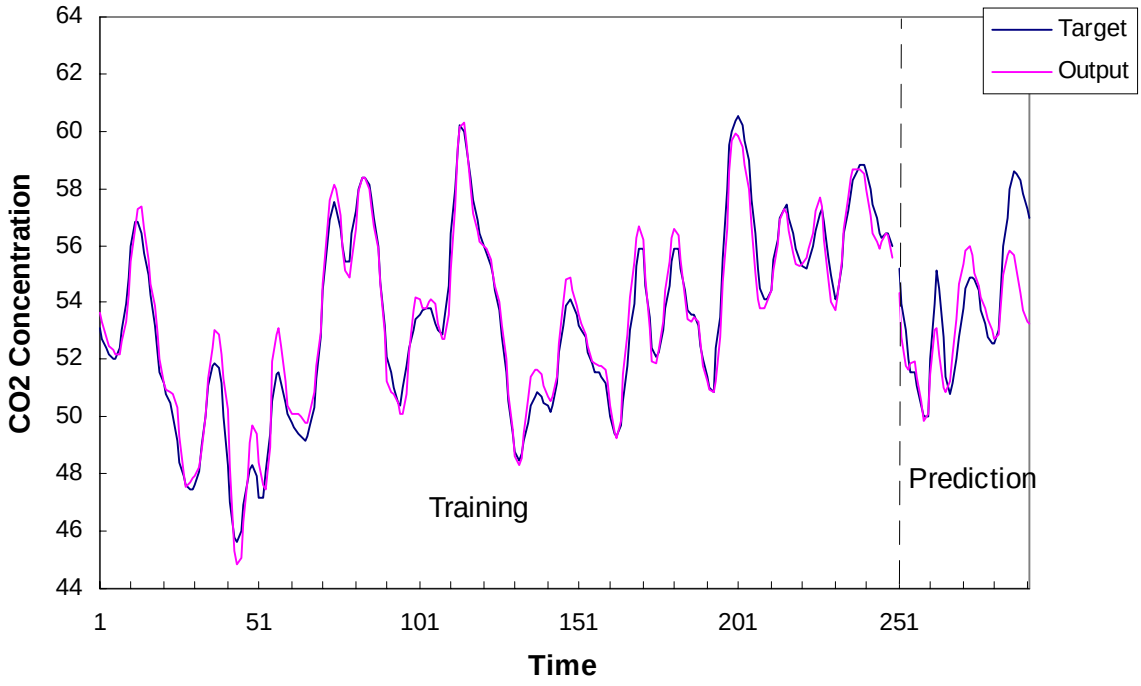


Figure 8.20. Training result on the gas furnace time series data using 4 input attributes, and its prediction on 42 test data pairs. The MSE on the training data set was 0.384.

The training result is shown in Figure 8.20. The Mean Squared Error (MSE) on the training data was 0.384, which is comparable to the result 0.395, reported by Pedrycz, Lam and Rocha (1995). In their experiment, the same time delay and 4 input variables as those in our experiment were used. A neural network model using back-propagation learning with 7 hidden node was used in their training. We have shown again that with a simple 4-layer thin-film stack, we can achieve almost the same training result. Although the MSE on the test data is larger: 2.15, the training result still indicates that the OTFM has learned the hidden pattern from limited previous data to predict the future output trend.

As we can notice in Figure 8.20, though the trained OTFM approximates well the target curve on the training data, its output does not match the test data set (predict the future) as closely. In order to further improve the performance of the OTFM on the prediction set, more historical data were considered as input variables for training. We considered 10 variables as input attributes, *y(t-1), ..., y(t-4), u(t-1), ..., u(t-6),* to affect the present output *y(t)*, as suggested by Sugeno and Yasukawa (1993). This time training data consisted of 245 sets of the input variables (10 for each

set as described above). The first target is the *y(t)* of the 6th data pair from the beginning of the training set, as the time delay is 6 steps back and all 5 data pair before the current one have been used for training to predict the current *y(u)*. However for the 42 test data pairs there is no such problem, as we can always use the last few data pairs from the training set to predict the first 5 targets of the test data set. This time as more input attributes were used, the number of the layers in the stack was increased. Table 8.19 shows the 20-layer thin-film stack used for this training and the found optimal set of layer thickness values by GA as the solution.

Layer	Refractive Index	Found thicknesses (μm)
(Output)	4.0	-
1	1.4	0.503875
2	3.6	0.385072
3	1.4	0.617846
4	3.6	0.576687
5	1.4	0.625
6	3.6	0.055364
7	1.4	0.899861
8	3.6	0.539047
9	1.4	0.520535
10	3.6	0.462766
11	1.4	1.085743
12	3.6	0.29726
13	1.4	0.481615
14	3.6	1.197523
15	1.4	0.771915
16	3.6	0.753187
17	1.4	0.943736
18	3.6	0.694948
19	1.4	1.001625
20	3.6	0.253046
(Input)	1.0	-

Table 8.19. Optical description of a 20-layer OTFM for solving the gas furnace time series prediction. λ=12.0μm; N_λ=1; Scaling factors Δ were 0.01 for *y(t)* and 0.1 for *u(t)* respectively. Found thicknesses were obtained by using the Genetic Algorithm.

The Genetic Algorithm approach and a single optical wavelength 12µm were again used for training the OTFM. With a population size of 400, a chromosome length of 500 that covers 20 thickness values (25 for each), crossover probability 0.6 and mutation rate 0.033, the OTFM was trained again. The Figure 8.21 shows the Mean Squared Error (in this case the MSE is the same as the merit function value, see Section 7.2), based on the 20 thickness values which were found by the fittest chromosome selected from the population, decreased from 3.8136 to 0.9185 after 500 generations of GA runs.

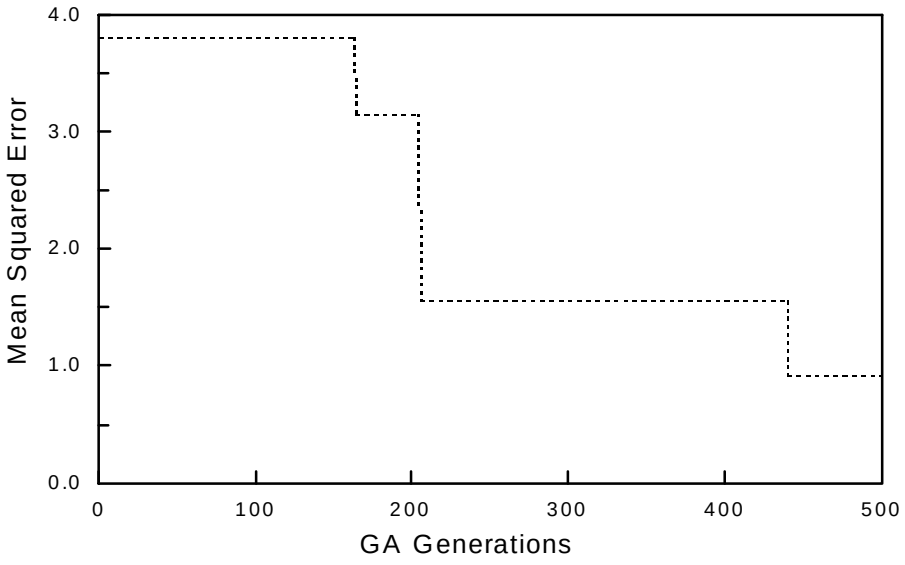


Figure 8.21. The Mean Squared Error decreased from 3.8136 to 0.9185 after 500 generations of GA runs.

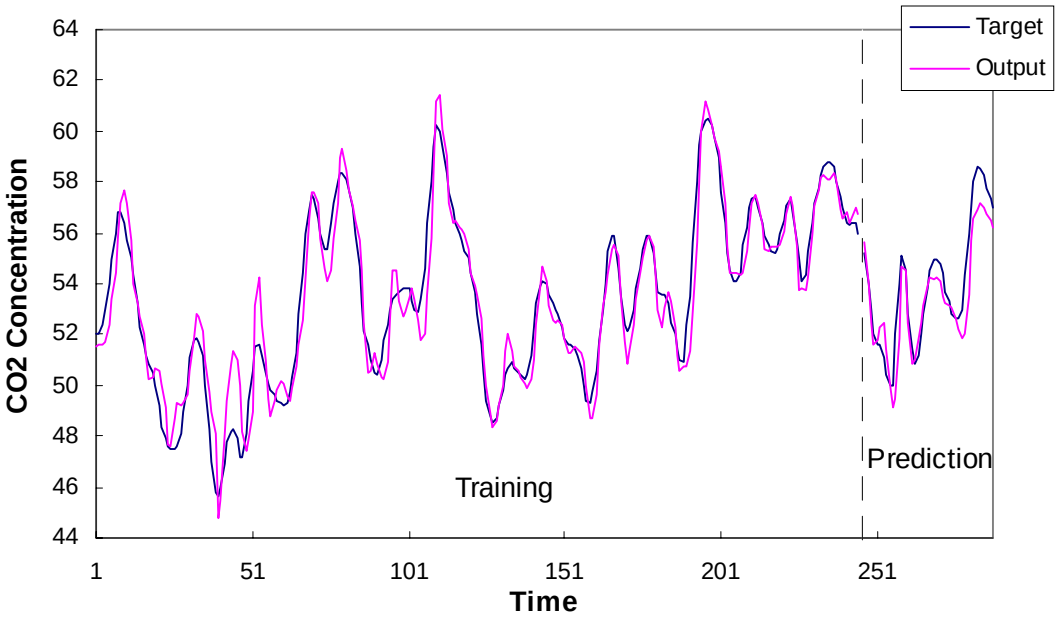


Figure 8.22. Training result on the gas furnace time series data using 10 input variables. The MSE was 0.9185 on the training data set and 1.072 on the test data set.

The training result on this 20-layer stack is shown in Figure 8.22. Though the MSE on the training set is higher than that of Figure 8.20, the MSE on the prediction set has been lowered from 2.15 to 1.072. These training results have shown clearly that by using more historical data for training, the OTFM has learned to predict better on the prediction data set.

8.5 Summary

The learning examples used in this chapter have been extensively studied in the field of connectionist models. This has made it easier to compare their OTFM experimental results with those done by other researchers using different approaches. The experiments described in this chapter have demonstrated that the OTFM's learning capability of accomplishing some complex computational tasks is comparable to those of conventional connectionist models, in particular, to that of the widely used feed-forward neural network model employing the back-propagation learning algorithm.

It must be pointed out that the total experimentation with the OTFM in connectionist learning thus far is only in its initial stages, compared with the vast scale and extent of the empirical and theoretical research conducted on the widely used neural network models. The OTFM is definitely worth further investigation in many respects. Some pointers for future directions of research in this regard will be provided in Chapter 10.

In the next chapter, an analysis derived from the OTFM experimental results will be presented.

Chapter 9
Analysis of Results

9.1 Introduction

The preceding chapter described various individual experiments conducted on the OTFM in detail. In order to evaluate the performance of the OTFM in these experiments, this chapter presents a more collective analysis of these experimental results, particularly the issues concerning the OTFM system parameters, training algorithms, accuracy of classification, performance on noisy data and training time.

One should be aware that the analysis of all the experimental results in this chapter is based on an experience with the OTFM model that is necessarily limited when compared with the vast amount of accumulated research that has been conducted in connection with the conventional feed-forward neural network model over the past decade. There are other alternative ways of conducting these experiments which have not been tried as yet. For example, a different set of starting values for system parameters might yield a different training result; choice of different search algorithms may determine significantly how well the OTFM is trained.

9.2 System Parameters

From a dynamical systems perspective, learning in a connectionist model is associated with adaptive modification of parameters (Serra & Zanarini 1990). For example, in a neural network, these parameters can be regarded as those related to individual nodes and those related to the connections. In the OTFM, learning takes place by modifying its system parameters as well, not only by adjusting connection-like layer thicknesses, but also by adjusting other parameters, such as using different light wavelengths. This means that a thickness combination pattern can be used over more than one wavelength for learning. In a neural network model, learning is achieved by adjusting connection weights, while all the other system parameters relevant to this connection weight pattern are normally fixed.

OTFM	
N	The number of thin-film layers in the stack
d_i	The thickness of layer i
n_i	The refractive index of layer i
λ	The wavelength
numlam	The number of wavelengths
d_inc	The incremental value of a thickness

Table 9.1. System parameters to be initialized in a N-layer OTFM, i = 1, 2, 3, ..., N.

Feed-forward Neural Network Model	
hidden	The number of hidden nodes
num_hidden	The number of hidden layers
rate	The learning rate of weight pattern
momentum	The momentum of the weight changes
$weight_{ij}$	The connection weight between nodes i and j

Table 9.2. System parameters to be initialized in a feed-forward neural network.

Table 9.1 and 9.2 show the system parameters of both the OTFM and a feed-forward neural network model in general. The parameters related to the adopted search algorithms are not given. The reader can refer to Section 5.3.

In a feed-forward neural network model, parameters such as the number of hidden nodes must be chosen before training. Other parameters including initial connection weights, the learning rate and the momentum must also be specified. Performance may depend greatly on the initial setting of these parameters. The difficulty is that these parameters are usually meaningless to the people running the experiments. In many cases, the only way to discover the preferred setup of the parameters is to try each configuration in turn. This can be very unpredictable and time consuming.

Just like a feed-forward neural network model, the OTFM also needs many trials to find a preferred setup of parameter values. When the OTFM starts learning, or searching for an appropriate set of thickness values to optimise the merit function, the initial values for parameters n_j, d_j and λ (of *j-th* layer) are determined by the optical thin-film model designer. The range $0 \leq 4\pi n_j d_j / \lambda \leq 2\pi$ is normally considered (Case 1983), since the reflection coefficient is invariant with respect to a change in $n_j d_j$ by the amount $\lambda/2$(see Section 4.3.2 and Figure 4.4). For instance, if we take the base refractive index n_{Bi} for each layer in the OTFM alternating between 1.2 for odd-number layers and 4.0 for the even-number layers (see the pattern recognition example in Section 8.3), and d_j is chosen to be in the range 0.0 to 5.0, then we can determine that λs in the range somewhere around 12.0 would be appropriate, at least for the odd-number layers with n_j of 1.2, though it would be unnecessarily large for the even-number layers with n_j of 4.0. The overall objective is to choose layer thickness ranges that result in the most efficient use of computational resources for searching the parameter space.

The final thicknesses found by the OTFM learning are influenced by a combination of many factors, including the choice of search algorithms, the initial setup for the OTFM parameters and those of the search algorithm. Any change in the initial setup of these parameters can result in a different OTFM parameter search space, therefore it may well lead to a different solution.

9.3 Training Procedures

As discussed above, the OTFM training or learning is usually carried out by searching for a set of appropriate layer thicknesses to satisfy the target condition of the model. The search algorithms adopted in the OTFM have been the N-squared Scan Method and the Genetic Algorithm described in Section 5.3. However, the N-squared Scan Method has some drawbacks as a training procedure when dealing with problems that require a larger number of thin-film layers.

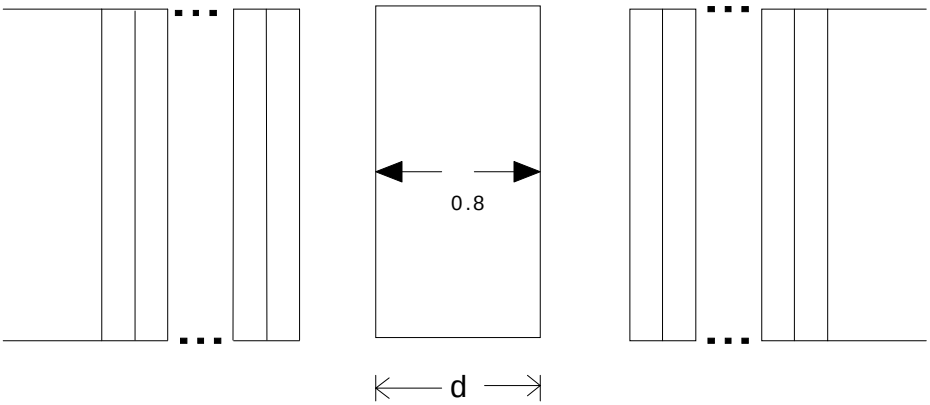


Figure 9.1. Choosing a thickness value on a single layer.

Using the N-squared Scan Method, the search steps and parameter ranges are determined by the designer. Search takes place with respect to one variation of layer thickness at a time. As shown in Figure 9.1, if the initial thickness for *i*-th layer d_i is 0.8, and the incremental thickness Δd = 0.05 is the search step specified, and the number of searches on each layer q is 20, then the search conducted is centred at d_i = 0.8, and takes on values on either side of that center value in steps of 0.05, i.e. the thicknesses used for the calculation are in turn 0.75, 0.70, 0.65, ..., 0.3, and 0.85,0.90,0.95, ..., 1.3. The smaller the value of the incremental thickness (Δd) and the greater the number of searches on each layer (q) are, the more thorough the search would be. However this would increase the amount of computation involved, since more evaluations are required. So long as time is affordable, a better experimental result would be achieved with a finer Δd and a larger q. Since the search is carried out on every layer with a fixed number of evaluations, if an evaluation is better than the previous one, the OTFM search procedure will save the values of this new set of learning parameters; if not, the search procedure simply discards this evaluation, and moves on to the next trial. Compared with the gradient descent search, e.g. the back-propagation learning in a neural network model, where each search step is in the right direction of reducing the merit function value, the N-squared Scan Method involves “false steps” that are in directions not necessarily reducing the merit function value. However, on the other hand, for problems that have a large number of minima, the N-squared Scan Method can easily avoid being trapped in local minima by using larger search steps, whereas it might be difficult for the gradient descent search to avoid such traps. We compared both the methods (N-squared Scan Method and gradient descent) on learning the iris classification using a feed-forward neural network model (so the N-squared Scan Method was used to optimize all the connection weights between the input and

hidden layer, and between the hidden and output layer). Both of them converged to a similar solution, but the N-squared Scan Method took longer. It is possible that the iris data does not have many local minima, therefore there was no difficulty for the gradient descent to locate the best overall minimum, whereas for the N-squared Scan Method, there were many unnecessary "false steps" involved during the search. Thus especially when applying the N-squared Scan Method to a large number of thin-film layer thicknesses, because of its trials of many "false steps", it can be very computationally demanding.

Compared with the N-squared Scan Method, the Genetic Algorithm appears to be more efficient with respect to global search. The GA method searches simultaneously in multiple dimensions of the parameter space and eventually focuses on a promising region in order to converge to a minimum. However when fine-tuning the model to reach a minimum, the N-squared Scan Method is superior to the GA, as the designer has complete control over how finely spaced the search steps should be. Overall, the two approaches are not noticeably different in efficiency when dealing with a smaller number of layers because of the limited number of evaluations, but the difference is obvious when the number of layers becomes larger. The N-squared Scan Method may consume considerably more computation and take much longer to achieve training comparable to that of the GA method. Thus for some complex tasks, the best approach is to combine both approaches, searching first globally using the GA method and then using the N-squared Scan Method to fine-tune the model to reach the best value possible.

In contrast, in the training of a feed-forward neural network model using the gradient descent search algorithm, every evaluation of a new trial is certainly a step in the direction of improving the current learning. If an evaluation does not contribute to learning, then the training stops, thus there is no waste of evaluation. Other global or regional search algorithms (Dobrowolski & Kemp 1990) may be worth more investigation with respect to the OTFM since they might help shed light on the OTFM parameter structure and search more efficiently, leading to improved learning ability. The gradient descent search has been tried previously by a few researchers on the thin-film design problem (it is very complicated and difficult to derive), but because the problems they dealt with were much more complex in the learning parameter space (i.e. many local minima exist), convergence of such parameters using such a technique has been difficult to achieve. This suggests that gradient descent approaches will not yield satisfactory results when complicated learning structures (either OTFM or neural network) that are likely to have many local minima are

evaluated. For these types of problems, the N-squared Scan and GA methods are preferable, because both of them can find ways of stepping out of local minima (the gradient descent methods need to employ noise or annealing techniques to deal with this problem).

9.4 Accuracy of Classification

In general, the OTFM's ability to represent concepts and correctly classify novel examples appears to be comparable with that of the feed-forward neural network model. For instance, when the trained OTFM was used to classify the iris test data, which includes 30 previously unseen examples, only two examples was classified incorrectly. The same result was obtained from using the feed-forward neural network model employing back-propagation (but it took somewhat longer to train the neural network model). Note that the same 5 examples classified incorrectly by the feed-forward neural network model were also classified incorrectly by the OTFM. Since in both training cases, the same training examples were used, one possible explanation for this similarity in performance is that both trained models reached the same conclusion about the internal relationships among the training data, then classified and generalized in the same way on the test data as well.

9.5 Performance on Noisy Patterns

Our initial experiment suggests that the OTFM degrades gracefully when noise in the input pattern increases. As shown in Figure 8.6, p1, p2, p3 and p4 are noise-increased versions of pattern “I”. The output curve (reflectance values) of the OTFM for each pattern changes gradually from p1 to p4, and p4 is obviously the worst one according to the desired target reflectance. However compared with the patterns “C”, “O”, and “X”, p4 still has a curve closer to the target reflectance of pattern “I”. These “noisy” curves show that although they do not perfectly fit the target, they are sufficiently close so that they can be distinguished from alternative targets. This preliminary result shows how the trained OTFM performs well not only on specified target patterns, but also on their noisy versions.

9.6 Training Time

Training time is determined by a number of factors, including the hardware requirements, the iteration number, the number of wavelengths used for training and the number of layers used in the OTFM. For example, with the N-squared Scan Method, the OTFM training on the iris data set took about 16 minutes on a PC, and the feed-forward neural network model employing the back-propagation took 50 minutes or more to achieve a similar result. However, the GA-based OTFM reached a satisfactory solution only after the first generation run, for about 3 minutes on a Sun SPARC station 5. The iteration number was set by the model designer to determine the number of runs of search procedures in the OTFM. Using more wavelengths means a greater number of calculations and evaluations to be done in total. For example, training on the eight sample 5-by-5 grid pattern recognition at a single wavelength took about 3 minutes for the system platform to achieve a satisfactory result, while training on the four sample 5-by-5 grid pattern recognition over 8 wavelengths took 16 minutes. Using more layers in the OTFM also increases the training time substantially. For instance, when using the same initial setup for training the OTFM on the iris data employing the GA search method, a 4-layer stack took only 3 minutes to converge to a satisfactory solution, but a 25-layer stack took about 4 hours to converge to a similar solution (see Section 8.4.2).

9.7 Summary

The overall performance of the OTFM on various learning problems has been shown to be comparable with a feed-forward neural network model in learning many complex computational tasks, especially when it is applied to learning problems that are nonlinear and have a moderate number of input attributes (i.e. it is not necessary to have a large number of layers for encoding the input attributes). In some experiments, the OTFM could achieve performance that is even better than that of the feed-forward neural network using back-propagation algorithm. Its comparable learning capability has been demonstrated through the problem solving examples that are commonly used in the field of connectionist models. Consequently this research has demonstrated that the OTFM can be used as an alternative learning model to other conventional connectionist models, including the widely used feed-forward neural network models.

The next chapter will conclude the research result and provide pointers for future research directions.

Chapter 10 Conclusion

10.1 Summary

In this research we have proposed a novel connectionist learning architecture based on an *Optical Thin-Film Multilayer* (OTFM) model. To the author's knowledge, the research communities of the optical thin-film design and connectionist models have not been in close touch with each other. By viewing the optical thin-film model from the standpoint of connectionist models and identifying their common characteristics, this research has attempted to bring together knowledge from both areas. Using such an approach, we have developed a novel connectionist learning architecture that is different from the biologically neuron-inspired models that have always been the most common connectionist models. Because the proposed model is based on the optical thin-film technology, it has a unique architecture and some distinct characteristics in contrast to neuron-inspired models. Thus for implementing learning tasks it provides an alternative to more conventional connectionist models such as the artificial neural network models.

In this study our work has been focused on developing the proposed model and investigating its capability to accomplish complex computational tasks, as well as comparing its performance with the conventional connectionist models, in particular, the widely used feed-forward neural network model. Developing such a thin-film based connectionist model has involved complex algorithm design and implementation. We adopted an object-oriented modelling approach for the implementation, since it is particularly suitable for constructing connectionist models, and furthermore it makes modelling elements easily reusable for future research. To investigate the proposed model, we have selected some of the most widely-used learning examples in the connectionist model field. Experiments were carried out in such a way that comparison could be readily made with the experimental results from other researchers in the field.

Another issue considered for this learning system has been the choice of search algorithms. A search algorithm can profoundly impact the performance of a learning system. We have surveyed the optimization methods that have been used for solving the thin-film design problems. Though we did not adopt any optimization methods reviewed by Dobrowolski (1990), it does not imply

that they are not feasible (in fact, the suitability of search algorithms is very much application-dependent in many cases). The overall objective is to search well globally, as well as locally in the system parameter space for the proposed thin-film multilayer model. Two search algorithms, the N-squared Scan Method and the Genetic Algorithm have been adopted and they have been found to be sufficient in dealing with many learning tasks.

Conducting experiments on the proposed connectionist model has played an important part in this research. However up to now, there has not been a commonly accepted set of unified practices and methodologies for conducting experiments in the connectionist model field. As a result, it has been difficult for researchers to cross-validate their experimental results (Flexer 1996). Furthermore this has made it hard to evaluate how well a connectionist model performs. We have made attempts to address this issue by reviewing the most widely-used approaches and techniques in this area. Although we did not use all the techniques and requirements suggested by Flexer (1996), in most of the cases we adopted the same approaches that were used by the researchers who applied the same learning problems to their connectionist models.

For the ease of comparison of the OTFM's performance with more conventional neural network models, experimental examples used in conjunction with the OTFM were those mostly widely-used in the field of connectionist models. By illustrating how the OTFM finds solutions for these learning examples, it has been demonstrated that the OTFM's capability in learning many complex computational tasks - i.e. supervised learning tasks in this research, is comparable to that of a feed-forward neural network model. This suggests that it can be used as an alternative learning model to the more conventional connectionist models for solving various supervised learning problems.

In the following sections we first summarise the scientific contributions of this research and the results of the investigation on the use of the proposed *Optical Thin-Film Multilayer* (OTFM) model as a connectionist model. Secondly we discuss the strengths and limitations of the proposed OTFM. The Chapter ends with suggestions of possible future research and further improvements of this model.

10.2 Main Contributions

The connectionist model proposed in this research was originally an idea derived from the field of optics, with its unique architecture and computational process that differ from those of the neuron-inspired connectionist models. This work is original, and there has not been any previous published work in this area by other authors. The research was carried out in two phases: the first phase was the development and implementation of the simulation model and the second was the exploration and investigation of the model's learning capability in comparison with conventional connectionist models, in particular the feed-forward neural network model. The main contributions of this research can be summarized as follows:

- An alternative connectionist learning architecture based on an Optical Thin-Film Multilayer (OTFM) model has been proposed and developed. This model has a unique architecture and computational properties compared with other connectionist models. It has been demonstrated that this architecture and optical properties can be viewed from the perspective of connectionist models, therefore it can be regarded as a novel connectionist learning architecture.

  A summary of the associated work in proposing and developing the OTFM is as follows:

  - An original approach of feeding inputs to a connectionist model and measuring its outputs has been proposed and developed for the proposed optical thin-film model (OTFM). By using unique optical thin-film properties and structure, this approach is different from that of a conventional neural network model.

  - The lack of a commonly accepted set of standards for experimental validation in the connectionist research community has been addressed by reviewing current experimental techniques and approaches that are in common use.

  - Optimization methods for solving the thin-film design problems have been reviewed. Two search algorithms, the N-squared Scan Method and the Genetic Algorithm, have been adopted in this research.

- In order to conduct experiments on the OTFM and evaluate its performance in accomplishing various learning tasks, a prototype software simulation of the model has been built from first principles.

- A variety of widely-used supervised learning examples in connectionist learning have been used for conducting experiments on the OTFM. It has been shown that the OTFM is capable of performing some complex learning tasks that are typical of those to which conventional connectionist models are applied. The OTFM can be used as a generic learning system and applied to a wide range of applications that are not only limited to the thin-film domain.

    - An analysis of the experimental results has been presented and compared with the results by using the feed-forward neural network model. This analysis has shown that the OTFM has a performance in accomplishing many complex supervised learning tasks that is comparable to those of a typical feed-forward neural network model.

In short, proposing and developing this new connectionist model based on an optical thin-film multilayer model, and investigating its learning capability are the major contributions of this research. In addition, building the prototype of the simulation model and conducting various experiments on it has also been important components of the research, since the simulation model is the testbed for the proposed OTFM and the experimentation is the demonstration of the OTFM's learning capability as a connectionist model.

10.3 Strengths and Limitations

Having analysed all the experimental results in Chapter 9, we have observed some of the strengths and limitations that the OTFM exhibits. Some strengths are summarised as follows:

- The OTFM provides a potentially valuable alternative to the current connectionist learning models. It is useful especially when dealing with nonlinear mapping problems, in particular supervised learning problems.

- With only few layers in the stack, the OTFM is capable of accomplishing a learning task equivalent to that done by a feed-forward neural network model of a much more complex network structure. This has been demonstrated by the problem solving in the iris classification and gas furnace time series prediction (see Sections 8.4.2 and 8.4.3).

- The OTFM is noise resistant. It degrades gracefully when noise-added input is fed into the trained model. This has been illustrated by the solving of pattern recognition problems (see Section 8.3).

The following two limitations have been observed in connection with the OTFM:

- It has been observed that using a large number of layers, e.g. on the scale of 100 or more, in the OTFM results in a complicated search space with a great many local minima, making the searching task lengthy and difficult.

- The OTFM, as it currently stands, only uses learning algorithms that do not require the use of derivatives, since calculating the gradient information for the optical thin-film model involves very complicated calculations and can be computationally expensive. This has limited us from using gradient descent learning as used for a feed-forward neural network model employing the back-propagation learning algorithm.

10.4 Recommendations for Future Research

As a new approach to connectionist learning, the OTFM is still at its initial stages of development and exploration. In view of the vast number of theoretical and experimental studies that have significantly advanced the area of conventional feed-forward neural network models, it is unlikely that this initial research on the OTFM has explored the full extent of its potential. Further experiments with different learning problems are worth pursuing, in order to gain a better understanding of the OTFM system behaviour in different circumstances. Many factors can impact greatly on the performance of the OTFM and it could possibly be improved in various aspects. A number of possible improvements of the OTFM itself and other related recommended future work are discussed in the following:

- More investigation from the optical perspective is worthwhile. In particular, the high nonlinear characteristics of thin-film multilayer should be studied further, as it is the essence of this connectionist model. In this research only one optical thin-film calculation algorithm was used, but there are many other calculation algorithms that have yet to be tried.

- Many search algorithms have yet to be investigated. There is a good review on thin-film multilayer optimization procedures by Dobrowolski and Kemp (1990). Other approaches such as simulated annealing (Davis & Steenstrup 1987) might be worth trying as well. An appropriate search algorithm is crucial for satisfactory training in the OTFM. The "N-squared Scan Method" was found useful when applied to smaller data sets, but can be computationally expensive on larger data sets. Efficient search algorithms could improve the performance of the OTFM dramatically.

- Different techniques for calculating the merit function value and estimating the true error rate may prove to be useful (see Sections 7.2 and 7.7). There is a review on different types of merit functions which have been used in the past in optical thin-film calculations (Dobrowolski *et al*. 1989). Further investigation in this area is of value. With respect to estimating the true error rate, a single training data set was used for training during problem solving in the experiments described in Chapter 8 (except the breast cancer prognosis, where multiple random training and test sets were used for training and testing). Resampling techniques such as the *k*-fold cross-validation that may produce more accurate estimations on error rate are worthy of investigation in this respect (Weiss & Kapouleas 1989; Cohen

1995).

- The learning explored in this research are so far supervised learning tasks only. Other types of learning such as unsupervised or reinforcement learning are worth investigation (Hertz, Krogh & Palmer 1991).

- Experiments on solving a wider range of learning problems are of value. Many other artificial as well as "real-world" data sets have been used for training a feed-forward neural network employing back-propagation or other learning methods, in addition to those that have already been used for training of the OTFM. The ability to learn solutions for large complex "real-world" problems is important for benchmarking a new learning model.

- Most of the learning problems described in Chapter 8 have only 3 or 4 categories to be classified (only the Eight 5-by-5 Grid Pattern Recognition has eight patterns to be classified). However when the number of categories (classes) in the domain increases (e.g. 15 different categories), more interference among different classes rises and thus more discrimination is needed. The OTFM that may perform well for few-category learning may fall short for many-category learning tasks, so experiments in this area are needed.

- Learning in the OTFM has the same "black box" effect as other connectionist models. In order to understand how the OTFM achieves a learning process in a more meaningful way, rule extraction for the OTFM would be very valuable. This would require an in-depth knowledge of the optical thin-film system design. One such possibility that might be worth investigation has been described by Dobrowolski and Piotrowski (1982): "it is always possible to duplicate the normal incidence performance of any multilayer system consisting of many different materials by one consisting of only the lowest and highest refractive-index materials employed in the original system." It is then possible to simplify the thin-film model by replacing the old system with the new one that has only layers with the lowest and highest refractive-index. Simplifying the OTFM structure (another possibility is to combine a number of layer thicknesses into one) would make it easier to explain the trained OTFM in more meaningful terms, such as rules. It is similar to the rule extraction of a trained neural network by pruning the network structure first, such as eliminating the insignificant inputs and connections (i.e. those with very small weights). The reader can refer to (Andrews,

Diederich & Tickle 1995) and (Gallant 1988) for further background information in this area.

- The OTFM requires a large number of optical system parameters to be specified. If Genetic Algorithms are used, there would be many GA parameters that must be specified (some parameters in GAs are continuous probability values). Under such conditions, it seems highly unlikely that the optimal set of system parameter values will be selected from a single run. This is the reason that on some occasions we noticed an improved performance after slightly changing the initial configuration. More extensive experimentation would help to gain a better understanding of the sources of such variations.

- It might be possible to use the thin-film multilayer structure as a recurrent network, which is a more general network than a feed-forward network. In a recurrent network, connections are allowed both ways between a pair of units, and even from a unit to itself (Hertz, Krogh & Palmer 1991: pp.163). In a thin-film multilayer structure, any change made in a thin-film layer thickness affects all the reflection coefficients at both boundaries of each layer, so each layer can be viewed as being connected to all other layers, including itself. The multilayer structure can be then seen as a fully connected network. Generally such a recurrent network can be used to learn to recognize or reproduce time sequences.

- The GA adopted in this research uses only binary representation for potential solutions, however other types of representation, such as the floating point representation of thin-film layer thicknesses, might prove to be more appropriate for some problem solving. The reader can refer to (Michalewicz 1996) for further background information.

- In order to improve further the performance of the GA-Based OTFM, attempts to incorporate problem specific knowledge into Genetic Algorithms might be of value. For example, it may be possible to incorporate problem specific knowledge into the chromosome's data structures, or define recombination operators that take problem specific knowledge into account when constructing offspring structures from parents. The evaluation of structures could also utilize problem specific knowledge. The interested reader can refer to (Michalewicz 1996) for further background information in this regard.
- It would be useful to have a software OTFM test-bed environment that is even more

interactive, than the one built by the author. This would provide the following features:

- The interface should enable the user to retrieve input data and generate the output data in a standard format.

- All these data should be easily normalized to meet the optical requirement (e.g. a reflectance value is always within the range of 0.0 to 1.0).

- It should be able to promptly display the intermediate training and testing results, or those after a training run is completed.

One desirable feature for such a tool is that whenever a layer's thickness is altered, the tool should be able to display the movement of reflection coefficient points on a reflection coefficient complex plane (see Section 4.3.2). This will enable us to see where those calculated reflection coefficient points are distributed on the complex plane during the training, and therefore may help us to have a better understanding of how the learning proceeds and to decide which layer thickness should be varied and how much variation should be made.

- The OTFM simulation model developed in this research has shown that it may stand as a viable computational learning model in its own right, regardless of its feasibility for an optical realization. Nevertheless, such an optical realization would offer the possibility of more or less instantaneous evaluations of the "trained" thin-film stack (dynamically varying thickness values of thin-film layers during the training phase would be difficult for an optical implementation, though it can be easily carried out on the OTFM simulation model). Also, because optical beams at ordinary intensities do not interfere with each other, the trained OTFM could support the simultaneous evaluation of multiple signals. Variations in the refractive index can be achieved for some materials by means of elastooptic, electrooptic, and magnetooptic effects, though the magnitude of the effects is often relatively small (Levi 1980). This is an area that may offer promise for additional experimental investigation.

Appendix A

Code Listing for the OTFM

Using the N-squared Scan Method

The OTFM using the N-squared Scan Method was coded in C++. For a detailed description of its various components, which were constructed using an object-oriented modelling approach, the reader may refer to Chapter 6. The adopted search algorithm, N-squared Scan Method, is described in Section 5.3.1.

```
////////////////////////////
//         Slayer.h         //
////////////////////////////
// Class defined for a single thin-film layer. It can be used as a basic building component
// for a multilayer model.
#ifndef __SLAYER
#define __SLAYER

struct Complex{ double Re;   double Im;};

class TBlock;
#include "public.h"
#define LayerSize  32
#define WaveSize   5
#define  TwoPi 6.28318530717958648

_CLASSDEF(TSingleLayer)
class TSingleLayer :  public Object
{
    friend class TBlock;
    friend class Item;
    protected:
                double thickness, *Lambdas;
                Complex ref_index, *rx, *tx, f;
                int numlam;
                TPublic_funs PubFuns;
    public:
                TSingleLayer(Complex Ref_index, double Thick,int NumLam, double *Lambdas);
                ~TsingleLayer();

                // Calculate for a single layer.
                void Calc_layer_x(double lambdas, Complex &Rx, Complex &Tx);
                // these funcs are from Object
                virtual classType isA() const { return __firstUserClass; }
                virtual Pchar nameOf() const { return "TSingleLayer"; }
                virtual hashValueType hashValue() const { return 0; }
                virtual int isEqual(RCObject ATSingleLayer) const
                { return thickness == ((RTSingleLayer)ATSingleLayer).thickness
                 && ref_index.Re == ((RTSingleLayer)ATSingleLayer).ref_index.Re
                 && ref_index.Im == ((RTSingleLayer)ATSingleLayer).ref_index.Im; }
                virtual void printOn(Rostream outputStream) const
                { outputStream << "(" << thickness << ","
                   << ref_index.Re << "," << ref_index.Im << ")"; }
};

#endif __SLAYER
```

```
////////////////////////////
//        Block.h         //
////////////////////////////
// Class defined for multiple thin-film layers.  It includes an array of thin-film layers
// and variables that store calculated values for such a multilayer structure.
#ifndef __BLOCK
#define __BLOCK
#include <array.h>

#define  Pi 3.14159265358979324
#define  TwoPi 6.28318530717958648
#define  FourPi  12.56637061435917296

_CLASSDEF(TMultiLayers)
class TMultiLayers : public Array
{
 public:
    TMultiLayers(int upper, int lower = 0, sizeType aDelta = 0)
                : Array(upper, lower, aDelta){};
    virtual classType isA() const { return __firstUserClass + 1; }
    virtual Pchar nameOf() const { return "TMultiLayers"; }
 //            void Delete_Entry(int loc){removeEntry(loc);};
};

_CLASSDEF(TBlock)
class TBlock
{
    friend class TSingleLayer;
    friend class Item;
 protected:
    PTMultiLayers MLayers;
    int NumItem, numlam;
    double *d_sav, *lambdas;
    Complex *n, Out_F, In_F, Base, *alfa;
    Complex *r_o, *t_o, *r_o_back, *t_o_back;
    TPublic_funs PubFuns; // Access to complex number calculations.
 public:
    TBlock(int NumLam, int NumLay, double *Lambdas, Complex *N,
                  double *D_sav, Complex In, Complex Out);
    ~TBlock();
    void AlfaParams();
    void Create_One_Layer(int layer);
    void setup_supers();
    void setup_subs();
    void update(double *D_sav);
    void Add_one(int layer);
    void calc_superstructure(int wav);
    void substructure(BOOL direction, int wav);
    void BackOff_one(int layer);
    void Zstak(int Layer, Complex &z, Complex &t, Complex *theta, Complex *alfa,double Nin,
    BOOL want_transmission);
};

#endif __BLOCK
```

```
//////////////////////////
//        Linklist.h      //
//////////////////////////
// During the training, when an input example is presented to the OTFM, we use
// an instance of class Item to store this example, conduct necessary calculation
// and store the results. Item has a pointer (*next) of its own type, so these
// instances can be stored on a linked-list, List, which is also defined here.
#ifndef __LINKLIST_H
#define __LINKLIST_H

#include "slayer.h"
#include "block.h"

class Item
{
    friend class List;
    friend class TOptThinFilm;
public:
    void Initialize(int layer_to_start, double *d_sav);
    void Adjust(int lay_row, int layer_to_vary, BOOL r_u_updated, double *d_sav);
    double GetMerit(double dx, int layer_to_vary);
protected:
    TBlock *SubStrate, *Supers;
    double *d_sav, *lambdas;
    Complex *r_combo, *next_r_u, *n;
    Complex *r, *r_best, *r_overallbest;
    double Rtarget;
    TPublic_funs PubFuns;
    Complex Ref_calc(Complex &r_combo, Complex r_u,Complex rx, Complex tx,
                Complex r_o, Complex t_o, Complex r_o_back, Complex t_o_back);
private:
  int numlam, maxlay;
    Item(int numlam, int maxlay, double *lambdas, Complex *n,
                  double *d_sav, double Rtarget, Item *item = 0);
    ~Item();
    Item *next;
};

class List
{
public:
    List(){head = 0; at_end = 0;}
    ~List(){ remove();}
    void append(Item *item);
    void remove(); // remove all items
    BOOL is_empty();
    void display();
    void readin(ifstream s);
    double TotalMerit(double dx, int layer_to_vary);
protected:
    Item *head, *at_end;
};

#endif
```

```
////////////////////////////////
//          Optfilm.h         //
////////////////////////////////
// Class represents the entire optical thin-film multilayer model(OTFM).
#ifndef __NSQUARE
#define __NSQUARE
#include "Linklist.h"

class TOptThinFilm : public List
{
 protected:
    BOOL WantPrint, SaveStack, r_u_updated;
    int max_film_thickness;
    int i,iter, lay, next_layer_to_vary,layer_to_vary, layer_to_start, lay_row, lay_col,
        thick_count, wav,numlam, maxlay, num_iterations,numthix, numthix_plus1;
    double Merit, BestMerit, OverallBestMerit, maxlam, minlam,scale_down,d_start,dx,
           lambda, midlam;
    Complex r, temp;
    // declare some arrays here.
    int *iteration;
    double *d_sav, *d_orig, *opthix_range, *d_inc;
    double *lambdas, *d_best;
    Complex *n, *n_opt;
    char Name[50];
    int FirstIter, SecondIter, ThirdIter, NumberOfInput;
    double AcceptableResult, delta, Pat[30];
    Complex aver1[WaveSize], aver2[WaveSize], aver3[WaveSize];
    TPublic_funs PubFuns;
 public:
    TOptThinFilm();
    ~TOptThinFilm();
    void calc_d_inc(double *d_inc, int maxlay, int numthix, double opthix_range[], Complex ndx[]);
    double calc_dstart(double d_saved, double d_inc, int numthix, double max_film_thick);
    void CopyComplex(int MaxLay, Complex *n, Complex *beta);
    void CopyStack(int MaxLay, double d_from[], double *d_to);
    BOOL encode();
    double training();
    void ReadInput();
    void recall(Complex *r_u_test, Complex *n);
    void ReadTest();
    void savethick();
    void result();
    void correct();
    void setup_subs(Complex *r_u, Complex *t_u, int NumLam,Complex Ndx[]);
};

#endif __NSQUARE
```

```
//////////////////////////
//          Public.h      //
//////////////////////////
// Functions for complex number and thin-film model related calculations. These functions
// will be frequently used throughout the thin-film model calculations.

#ifndef __PUBLIC
#define __PUBLIC

class TSLayer;
class TBlock;
class TOptThinFilm;

class TPublic_funs
{
    friend class TSLayer;
    friend class TBlock;
    friend class TOptThinFilm;
    friend class Item;
 public:
    Complex Cprod(Complex A, Complex B);
    Complex Cdiv(Complex D, Complex E);
    Complex CExponJ(Complex A);
    double Cmagsq(Complex x);
    Complex Comp_conj(Complex x);
    Complex Csqrt(Complex z);
    Complex Fresnel(Complex N);
    Complex Compos(Complex A, Complex B);
    double Atan_ns(double x, double y);
    Complex Clog(Complex z);
    void LayerParameters(int MaxLay, Complex Ndx[],Complex *F, Complex *alfa);
    void Add_a_module(Complex &Z, Complex r_over[], Complex r_back_over[], Complex t_over[],
                  Complex t_back_over[], int Wav, double Nin);
    void Zstak(int Layer, Complex &z, Complex &t, Complex *alfa, Complex *f,
                  Complex *theta, int MaxLay, double Nin, BOOL want_transmission);
};

#endif __PUBLIC
```

```
//////////////////////////////
//          Optfilm.cpp      //
//////////////////////////////
// Main program, which basically runs first from model.encode()(see Section 6.4 for detail),
// then save the training result and the found layer thickness values.
// If there are test data, model.ReadTest() and model.recall() can be used to
// read in the test data and run them on the trained model.
#define WIN30

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <owl.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <float.h>
#include <assert.h>
#include <iostream.h>
#include <fstream.h>
#include "optfilm.h"

// The starting point of the program. The model.encode() is where the training is conducted.
void main()
{
    TOptThinFilm model;
    //double percent =  model.training();
    struct time start_t;
    gettime(&start_t);

    cout << "starting time: " << (int)start_t.ti_hour << ":" << (int)start_t.ti_min
            << ":" << (int)start_t.ti_sec << endl;

    if (model.encode());

    model.result();
    model.savethick();
    model.correct();  // used for testing data.
    //model.ReadTest();
    //test.recall();
}

// Make a copy of an array of complex numbers.
void TOptThinFilm::CopyComplex(int MaxLay, Complex *n, Complex *beta)
{
    for(int lay=0; lay <= MaxLay+1; lay++)
    {
               beta[lay].Re = n[lay].Re;
               beta[lay].Im = n[lay].Im;
    }
}

// Make a copy of an array of thin-film thicknesses.
void TOptThinFilm::CopyStack(int MaxLay, double d_from[], double *d_to)
{
    for(int lay=0; lay <= MaxLay+1; lay++)
    d_to[lay] = d_from[lay];
}

// Initialization of the model is done here. It also read in the input data for training.
ToptThinFilm::TOptThinFilm()
{
    max_film_thickness = 5;
    FirstIter = 59;     //11 + random(50);
    SecondIter = 17;
    ThirdIter = 7;
    AcceptableResult = 0.25;
    delta = 0.4;
    num_iterations = 3;
    layer_to_start = 2;
    NumberOfInput = 2;
    maxlay = 8;  //25;
    minlam = 4.4; //5;
    maxlam = 5.6;
    numlam = 1;   // At moment it should be less than 4.
    iteration = new int[LayerSize];
    d_sav = new double[LayerSize];
    d_orig = new double[LayerSize];
    opthix_range = new double[LayerSize];
    d_inc = new double[LayerSize];
    d_best = new double[LayerSize];
```

```
    n = new Complex[LayerSize];
    lambdas = new double[WaveSize];
    n_opt = new Complex[LayerSize];

    // read in "n"
    int lay;
    double Nin, Nout;
    cout << endl << "Enter file name: ";
    cin >> Name;
    char fn[50];
    sprintf(fn, "%s.stk", Name);
    ifstream stream(fn, ios::in);
    //open a file
    if (!stream)
    {
               cerr << "Cannot open output file.\n";
               exit(-1);
    }
    stream >> fn
               >> maxlay >> Nin >> Nout;
    for(lay = 0; lay <= (maxlay + 1); lay++)
    {
               stream >> n[lay].Re >> n[lay].Im >> d_orig[lay];
                  n_opt[lay] = n[lay];
    }
    // three more elements added here!!!
    stream >> NumberOfInput;
    stream >> delta;
    stream >> minlam;

    stream.close();//Close StackFile

    double LamFac;
    if (numlam == 1)
               LamFac = 0.0;
    else LamFac = (maxlam - minlam) / (numlam - 1);
    for(wav=1; wav <= numlam; wav++)
               lambdas[wav] = minlam + LamFac*(wav - 1);

    for(lay=1; lay <= maxlay; lay++)
               iteration[lay] = 1;
}

// Free the used memory.
ToptThinFilm::~TOptThinFilm()
{
    delete[] iteration;
    delete[] d_sav;
    delete[] d_orig;
    delete[] opthix_range;
    delete[] d_inc;
    delete[] d_best;
    delete[] lambdas;
    delete[] n;
    delete[] n_opt;
}

double TOptThinFilm::training()
{
    return 1;
}

// This is the basic N-squared Scan Method (see Section 6.4 for detail).
BOOL TOptThinFilm::encode()
{
    OverallBestMerit = 999.9;
    int count = 0;
    CopyStack(maxlay, d_orig, d_sav);
    midlam = 0.5*(lambdas[1] + lambdas[numlam]);
    for(lay=1; lay <= maxlay; lay++)
    {
               opthix_range[lay] = midlam;
               if (opthix_range[lay] < n[lay].Re * d_sav[lay])
                   opthix_range[lay] = 1.2*n[lay].Re*d_sav[lay];
               if (opthix_range[lay] > n[lay].Re*max_film_thickness)
                   opthix_range[lay] = n[lay].Re*max_film_thickness;
    }

    ReadInput();
    for(iter=1; iter <= num_iterations; iter++)
    {
```

```
            switch (iter)
            {
               case 1:   numthix = FirstIter; break;
               case 2:   numthix = SecondIter; break;
               default:  numthix = ThirdIter;
            }
            numthix_plus1 = numthix + 1;
            scale_down = 1.1/numthix;

            //Caculates d_inc's for all layers.
            calc_d_inc(d_inc,maxlay,numthix,opthix_range, n);

 layer_to_vary = layer_to_start;
 for(lay_col=1; lay_col <= maxlay; lay_col++)
 {
            BestMerit = 999.9;
            if (iter==1)
                CopyStack(maxlay, d_orig, d_sav);
            else
               for(lay=1; lay <= maxlay; lay++)
               d_sav[lay] = d_best[lay];   //[lay_col];

            for(Item *pt=head; pt; pt=pt->next)
               pt->Initialize(layer_to_vary, d_sav);

            for(lay_row=1; lay_row <= maxlay; lay_row++)
            {
               r_u_updated = FALSE;
               if (iteration[layer_to_vary] <= iter)
               {
                    dx= calc_dstart(d_sav[layer_to_vary],
               d_inc[layer_to_vary], numthix,max_film_thickness);
                    for(thick_count=1; thick_count <= numthix_plus1; thick_count++)
                    {
                           Merit = 0.0;
                           // Get a local merit
                           Merit = TotalMerit(dx, layer_to_vary);
            //We'll use those having the best merit in the new substructure.
                           if(Merit < BestMerit)
                           {
                                  r_u_updated = TRUE;
                                  BestMerit = Merit;
                                  d_sav[layer_to_vary] = dx;

                                  for(Item *pt=head; pt; pt=pt->next)
                           {
                                  for(wav=1; wav <= numlam; wav++)
                           {
                                  pt->next_r_u[wav] = pt->r_combo[wav];
                                  pt->r_best[wav].Re = pt->r[wav].Re;
                                  pt->r_best[wav].Im = pt->r[wav].Im;
                           }
                                  }
                           }// if merit...

                           dx = dx + d_inc[layer_to_vary];
                     }// For thick_count = ...
               }// if iteration[layer_to_vary]...

               if(BestMerit < OverallBestMerit)
               {
                    OverallBestMerit = BestMerit;

                    for(lay=1; lay <= maxlay; lay++)
                           d_best[lay] = d_sav[lay];

                    for(Item *pt=head; pt; pt=pt->next)
                    {
                    for(wav=1; wav <= numlam; wav++)
                    {
                           pt->r_overallbest[wav].Re = pt->r_best[wav].Re;
                           pt->r_overallbest[wav].Im = pt->r_best[wav].Im;
                    }
                    }
               }
               for(Item *pt=head; pt; pt=pt->next)
               pt->Adjust(lay_row, layer_to_vary, r_u_updated, d_sav);
               layer_to_vary = fmod(layer_to_vary,maxlay) + 1;
            }// lay_row loop.
            layer_to_vary = fmod(layer_to_vary, maxlay) + 1;
}//lay_col loop.
```

```
   cout << "OverallBestMerit: " << OverallBestMerit << endl;
   for(lay=1; lay <= maxlay; lay++)
              opthix_range[lay] = opthix_range[lay] * scale_down;
 }// iteration loop.

 for(lay=1; lay <= maxlay; lay++)
   d_sav[lay] = d_best[lay];

 for(Item *pt=head; pt; pt=pt->next)
 {
   for(wav=1; wav <= numlam; wav++)
   {
              pt->r[wav].Re = pt->r_overallbest[wav].Re;
              pt->r[wav].Im = pt->r_overallbest[wav].Im;
   }
 }

 cout << endl << endl << " Yes, I am terminated!!!";
 if (OverallBestMerit <= AcceptableResult)
   return 1;
 else
   return 0;
}

//Initialize the layers for the next calculation.
void Item::Initialize(int layer_to_start, double *d_sav)
{
    SubStrate->update(d_sav);
    Supers->update(d_sav);

    SubStrate->setup_subs();
    Supers->setup_subs();
    Supers->setup_supers();
    int layer = 1;
    while(layer < layer_to_start)
    {
              SubStrate->Add_one(layer);
              Supers->BackOff_one(layer+1);
              layer++;
    }
}

//Adjust the system parameters and vary the thickness of the next layer.
void Item::Adjust(int lay_row, int layer_to_vary, BOOL r_u_updated, double *d_sav)
{
    int wav, next_layer_to_vary;
    next_layer_to_vary = fmod(layer_to_vary,maxlay) + 1;
    SubStrate->update(d_sav);
    Supers->update(d_sav);
    if (next_layer_to_vary != 1)
    {
              if (r_u_updated)
                 for(wav=1; wav <= numlam; wav++)
                       SubStrate->r_o[wav] = next_r_u[wav];
              else
                 SubStrate->Add_one(layer_to_vary);
              Supers->BackOff_one(next_layer_to_vary);
    }
    else  // next_layer_to_vary = 1.
    {
               SubStrate->setup_subs();
               Supers->setup_subs();
               Supers->setup_supers();
    } // if layer_to_vary.....

    if (lay_row == maxlay)
    //if nest_layer_to_vary = maxlay, then the REAL next_layer_to_vary = 1.
              if (next_layer_to_vary != maxlay)
              {
                 SubStrate->Add_one(next_layer_to_vary);
                 Supers->BackOff_one(next_layer_to_vary + 1);
              } // else....( if layer_row....)
              else   // next_layer_to_vary = maxlay.
              {
                     SubStrate->setup_subs();
                     Supers->setup_subs();
                     Supers->setup_supers();
              }  //if next_layer_to_vary...(and) if lay_row...

} // procedure adjust_fixed_layers.
```

```
// Increamental step of the thickness for the “varying” layer, the layer to be varied
// many times during the training.
void TOptThinFilm::calc_d_inc(double *d_inc, int maxlay, int numthix, double opthix_range[],
Complex ndx[])
{
    for(int lay=1; lay <= maxlay; lay++)
    d_inc[lay] = opthix_range[lay]/(ndx[lay].Re*numthix);
} // procedure calc_d_inc.

// Starting thickness value for the varying layer.
double TOptThinFilm::calc_dstart(double d_saved, double d_inc, int numthix,double max_film_thick)
{
    int thick_count;
    double result;
    // first check top and of range to see if too high. If so, lower dstart.
    result = d_saved + d_inc*(div(numthix, 2).quot);
    while (result > max_film_thick)
               result = result - d_inc;
    // now go and get low value --- the true value of dstart.
    result = result - d_inc*numthix;
    // but make sure that result is not negative.
    while (result < 0)
               result = result + d_inc;
    return result;
};

//Calculate the overall reflection coefficient of a multilayer structure,
//which includes the overlaying, varying and underlying structures.
Complex Item::Ref_calc(Complex &r_comb, Complex r_u,Complex rx, Complex tx,
               Complex r_o, Complex t_o, Complex r_o_back, Complex t_o_back)
{
    Complex temp1, temp2, num, den;
    // calculate r_combination.
    temp1 = PubFuns.Cprod(rx, rx);
    temp2 = PubFuns.Cprod(tx, tx);
    temp1.Re = temp2.Re - temp1.Re;
    temp1.Im = temp2.Im - temp1.Im;
    num = PubFuns.Cprod(temp1, r_u);
    num.Re = rx.Re + num.Re;
    num.Im = rx.Im + num.Im;

    temp1 = PubFuns.Cprod(rx, r_u);
    den.Re = 1 - temp1.Re;
    den.Im = -temp1.Im;

    // r_combo will be used for the next r_u if this turns out be optimal.
    r_comb = PubFuns.Cdiv(num, den);

    // now calculate overall r.
    temp1 = PubFuns.Cprod(r_o, r_o_back);
    temp2 = PubFuns.Cprod(t_o, t_o_back);
    temp1.Re = temp2.Re - temp1.Re;
    temp1.Im = temp2.Im - temp1.Im;
    num = PubFuns.Cprod(temp1, r_comb);
    num.Re = r_o.Re + num.Re;
    num.Im = r_o.Im + num.Im;

    temp1 = PubFuns.Cprod(r_o_back, r_comb);
    den.Re = 1 - temp1.Re;
    den.Im = -temp1.Im;

    temp1 = PubFuns.Cdiv(num, den);
    return temp1;
}

//Calculaton done here for the Substrate, i.e. the underlying structure.
void TOptThinFilm::setup_subs(Complex *r_u, Complex *t_u, int NumLam,Complex Ndx[])
{
    int wav;
    Complex f, denom, two;
    f = PubFuns.Fresnel(Ndx[0]);
    denom.Re = Ndx[0].Re + 1;
    denom.Im = Ndx[0].Im;
    two.Re = 2;
    two.Im = 0;
    for(wav=1; wav <= NumLam; wav++)
    {
               r_u[wav].Re = -f.Re;
               r_u[wav].Im = -f.Im;
               t_u[wav] = PubFuns.Cdiv(two, denom);
    }
```

```
}// fuction setup_subs.

//Save the found thickness values after the training.
void TOptThinFilm::savethick()
{
     char fn[50];
     sprintf(fn,"%s.THK",Name);
     ofstream os(fn, ios :: out);
     os << fn << endl
        << maxlay << " " << n_opt[maxlay+1].Re << " " << n_opt[0].Re << endl;
     for(int lay = 0; lay <= (maxlay + 1); lay++)
     {
       os << n_opt[lay].Re << " " << n_opt[lay].Im << " " << d_sav[lay] << endl;
     }
     os.close();
}

//Save the calculation results.
void TOptThinFilm::result()
{
    struct date d;
    getdate(&d);
    struct time t;
    gettime(&t);

     char fn[50];
     sprintf(fn,"%s.RST",Name);
     ofstream os(fn, ios :: out);
     os << fn << endl;

     os << "File date: " << (int)d.da_day << "/" << (int)d.da_mon
          << "/" << (int)d.da_year << endl
          << "File time: " << (int)t.ti_hour << ":" << (int)t.ti_min
          << ":" << (int)t.ti_sec << endl;

     os << "OverallBestMerit: " << OverallBestMerit << endl
          << "FirstIter: " << FirstIter << endl
          << "Delta value: " << delta << endl
                    << "Lambdas: ";
     for(wav=1; wav <=numlam; wav++)
               os << lambdas[wav] << " : ";
     os << endl;

     int count = 0;

     for(Item *pt=head; pt; pt=pt->next)
     {
               count = count + 1;
               os << pt->Rtarget << ": " ;
               for(wav=1; wav <= numlam; wav++)
                  os << "{" << PubFuns.Cmagsq(pt->r[wav]) << "}";
               os << endl;
     }
     os.close();
}

//Read in input examples from a training data set.
void TOptThinFilm::ReadInput()
{
    // read in a file
    char fn[50], keyword[10];
    sprintf(fn,"%s.fct", Name);
    ifstream s(fn, ios::in);

    double Rtarget;
    int count = 0;

    for(int p=1; p <= 30; p++)
               Pat[p] = 0;
    for(;;)
    {
    // This is an example reading data for the sixteen 4-bit parity problem.
    if (s.eof() || s.fail() || (count==16)) break;
    else
    {
               count = count + 1;
               for(int j=1; j <= NumberOfInput; j++)
                  s >> Pat[j];
               s >> keyword;
               s >> Rtarget;
```

```
                for(j=1; j <= maxlay; j++)
                {
                n[j].Re = n_opt[j].Re + delta*Pat[j];    // rate is the scaling factor.
                }

                Item *pt = new Item(numlam, maxlay, lambdas, n, d_sav, Rtarget);

                   assert(pt != 0);
                   append(pt);
    }
    }
}

// Based on the found solution, the trained model can now be tested with novel
// data examples from testing data set. This is also a basic calculation of the
// overall reflection coefficient of a thin-film multilayer structure.
void TOptThinFilm::recall(Complex *r_u_test, Complex *n)
{
    //CopyComplex(maxlay, n, beta_opt);
    Complex *r_u, *t_u, *f, *alfa;
    r_u = new Complex[LayerSize];
    t_u = new Complex[LayerSize];
    f = new Complex[LayerSize];
    alfa = new Complex[LayerSize];

    PubFuns.LayerParameters(maxlay, n, f, alfa);
    setup_subs(r_u, t_u, numlam, n);
    for(wav=1; wav <= numlam; wav++)
    {
                lambda = lambdas[wav];
                Complex *theta;
                theta = new Complex[LayerSize];
                for(int i=0; i < LayerSize; i++)
                {
                   theta[i].Re = 0;
                   theta[i].Im = 0;
                }
                for(int lay=1; lay <= maxlay; lay++)
                   theta[lay].Re = ((4*Pi)*n[lay].Re)*d_sav[lay]/lambda;
    PubFuns.Zstak(1, r_u[wav], t_u[wav], alfa, f, theta, maxlay, n[0].Re,1);
    }

    for(wav=1; wav <= numlam; wav++)
    {
                //rrst[wav] = PubFuns.Cmagsq(r_u[wav]);
                r_u_test[wav] = r_u[wav];
    }
}


void TOptThinFilm::ReadTest()
{
// If there is a testing data set, then this function is used to read the test data.
}

void TOptThinFilm::correct()
{
    // Calculate how many test examples are correctly classified.
}



////////////////////////////////
//          Slayer.cpp        //
////////////////////////////////
//Calculations for a single thin-film layer, the basic processing element of the OTFM.
#define WIN30
#include <owl.h>
#include <stdio.h>
#include "slayer.h"
#include "public.h"

// Initialization of the variables.
TsingleLayer::TSingleLayer(Complex Ref_index, double Thick,int NumLam, double *Waveths)
{
     ref_index = Ref_index;
     thickness = Thick;
     numlam = NumLam;
     Lambdas = new double[WaveSize];
     rx = new Complex[WaveSize]; // reflection coefficients at different wavelengths.
     tx = new Complex[WaveSize]; // transmission coefficients at different wavelengths.
```

```
     for(int i=1; i <= numlam; i++)
     {
               Lambdas[i] = Waveths[i];
               Calc_layer_x(Lambdas[i], rx[i], tx[i]);
     }
     f = PubFuns.Fresnel(ref_index);
}

TSingleLayer::~TSingleLayer()
{
    delete[] Lambdas;
    delete[] rx;
    delete[] tx;
}

// Calaculation for the reflection and transmission coefficients of a single layer.
void TSingleLayer::Calc_layer_x(double lambda, Complex &Rx,Complex &Tx)
{
    double fac;
    Complex num, den, f, fsq, ep, epsq;
    // calculate the denominator for both reflection and transmission.
    f = PubFuns.Fresnel(ref_index);
    fsq = PubFuns.Cprod(f, f);
    fac = TwoPi*thickness/lambda;
    ep.Re = fac*ref_index.Re;
    ep.Im = fac*ref_index.Im;
    ep = PubFuns.CExponJ(ep);
    epsq = PubFuns.Cprod(ep, ep);
    den = PubFuns.Cprod(fsq, epsq);
    den.Re = 1 - den.Re;
    den.Im = -den.Im;

    //calculate the reflection.
    num = PubFuns.Cprod(epsq, f);
    num.Re = -f.Re + num.Re;
    num.Im = -f.Im + num.Im;
    Rx = PubFuns.Cdiv(num, den);

    //calculate the transmission.
    num.Re = 1 - fsq.Re;
    num.Im = -fsq.Im;
    num = PubFuns.Cprod(ep, num);
    Tx = PubFuns.Cdiv(num, den);
}



//////////////////////////////
//          Block.cpp       //
//////////////////////////////
//Calculations of a multilayer structure.
#define WIN30
#include <owl.h>
#include <array.h>
#include <abstarry.h>
#include "slayer.h"
#include "block.h"
#include "public.h"

// Initialize the multilayer structure with various values, e.g. the number of
// layers, wavelengths, layer thickcness values, input and output medium refractive indices.
Tblock::TBlock(int NumLam, int NumLay, double *Lambdas, Complex *N,double *D_sav, Complex In,
Complex Out)
{
    if (!(MLayers = new TMultiLayers(LayerSize+2, 0, LayerSize+2)))
     {
               cout << "Insufficient memory for TMultiLayers";
               exit(1);
     }
    n = new Complex[LayerSize];
    alfa = new Complex[LayerSize];
    d_sav = new double[LayerSize];
    lambdas = new double[WaveSize];
    r_o = new Complex[WaveSize];
    r_o_back = new Complex[WaveSize];
    t_o_back = new Complex[WaveSize];
    t_o = new Complex[WaveSize];
    NumItem = NumLay;
    numlam = NumLam;
    Base = Out;
```

```
    In_F = PubFuns.Fresnel(In);
    Out_F = PubFuns.Fresnel(Out);
    Complex denom, two;
    denom.Re = Out.Re + 1;
    denom.Im = Out.Im;

    two.Re = 2;
    two.Im = 0;
    for(int lay=0; lay<=NumLay+1; lay++)
    {
               n[lay] = N[lay];
               d_sav[lay] = D_sav[lay];
    }
    for(int wav=1; wav<=numlam; wav++)
               lambdas[wav] = Lambdas[wav];
    setup_subs();
}

TBlock::~TBlock()
{
    delete MLayers;
    delete[] n;
    delete[] alfa;
    delete[] d_sav;
    delete[] lambdas;
    delete[] r_o;
    delete[] t_o;
    delete[] r_o_back;
    delete[] t_o_back;
}

// Update the thickness values with ones that produce a better merit.
void TBlock::update(double *D_sav)
{
    for(int lay=0; lay<=NumItem+1; lay++)
               d_sav[lay] = D_sav[lay];
}

// Calculation of Alfa values for all the layers.
void TBlock::AlfaParams()
{
       Complex negF, temp_F;
       if(NumItem > 0)
       {
    RObject LayerObject = MLayers->operator [](2);
    alfa[2] = ((PTSingleLayer)(&LayerObject))->f;
    for(int lay=3; lay<=NumItem; lay++)
    {
               RObject LayerObjectPrev = MLayers->operator [](lay-1);
               negF.Re = -((PTSingleLayer)(&LayerObjectPrev))->f.Re;
               negF.Im = -((PTSingleLayer)(&LayerObjectPrev))->f.Im;
               RObject LayerObjectCurr = MLayers->operator [](lay);
               temp_F = ((PTSingleLayer)(&LayerObjectCurr))->f;
               alfa[lay] = PubFuns.Compos(temp_F, negF);
    }
    RObject LayerObjectLast = MLayers->operator [](NumItem);
    negF.Re = -((PTSingleLayer)(&LayerObjectLast))->f.Re;
    negF.Im = -((PTSingleLayer)(&LayerObjectLast))->f.Im;
    alfa[NumItem+1] = PubFuns.Compos(In_F, negF);
        }
}

//Necessary for the varying layer (the thickness is varied constantly).
void TBlock::Create_One_Layer(int layer)
{
    TSingleLayer *ALayer;
    ALayer = new TSingleLayer(n[layer], d_sav[layer], numlam, lambdas);
    MLayers->addAt(*ALayer, layer);
}
```

```
////////////////////////////
//          Engine.cpp       //
////////////////////////////
//Functions used for calculations on a multilayer structure.

#define WIN30
#include <owl.h>
#include <math.h>
#include <float.h>
#include "slayer.h"
#include "block.h"

// Calculate substructure r and t -- return in rsub, and tsub
// used by calc_substructure and calc_superstructure.
void TBlock::substructure(BOOL direction, int wav)
{
    double pseud_nin = 1.0;
    int lay, level, Maxlay_plus1;
    int bottom_level, l_bias;
    double thet_fac;
    Complex f[LayerSize], alfa[LayerSize], Ndx[LayerSize], theta[LayerSize];

    for(int i=0; i < LayerSize; i++)
    {
               Ndx[i].Re = 0;
               Ndx[i].Im = 0;
               f[i] = alfa[i] = theta[i] = Ndx[i];
    } // initialize them with 0.

    bottom_level = 2;
    if (!direction)
               l_bias = NumItem + bottom_level;
    else
               l_bias = 0;
    thet_fac = FourPi / lambdas[wav];
    Maxlay_plus1 = NumItem + 1;

    Ndx[Maxlay_plus1].Re = 1.0;
    Ndx[Maxlay_plus1].Im = 0.0;
    Ndx[bottom_level - 1].Re = 1.0;
    Ndx[bottom_level - 1].Im = 0.0;

    theta[Maxlay_plus1].Re = 0.0;
    theta[Maxlay_plus1].Im = 0.0;

    if (bottom_level <= NumItem)
    {
               for(lay=bottom_level; lay <= NumItem; lay++)
               {// The array index below enables calculation of the layer parameters.
                // in reverse order(for back reflection) when do_trans = false.
                level = l_bias + (2*direction - 1)*lay;
                Ndx[level] = n[lay];
                if (direction)
                      Create_One_Layer(lay);
                theta[level].Re = thet_fac*n[lay].Re*d_sav[lay];
                theta[level].Im = thet_fac*n[lay].Im*d_sav[lay];
               }
               PubFuns.LayerParameters(NumItem, Ndx, f, alfa);
               // 'do_trans' determines whether transmission will be calculated.
               if (direction)
PubFuns.Zstak(bottom_level, r_o[wav], t_o[wav], alfa, f, theta, NumItem, pseud_nin, TRUE);
               else
PubFuns.Zstak(bottom_level, r_o_back[wav], t_o_back[wav], alfa, f, theta, NumItem, pseud_nin,
TRUE);
    }
}

// Calculate the overlaying structure (see Figure 4.9)
void TBlock::calc_superstructure(int wav)
{
    BOOL normal = TRUE;
    BOOL reverse = FALSE;
    // calculate r_o, and t_o.
    substructure(normal, wav);
    // Calculate r_o_back, and t_o_back.
    substructure(reverse, wav);
}// procedure calc_superstructuer.
```

```
// store variable values at different wavelengths.
void TBlock::setup_supers()
{
    for(int wav=1; wav <= numlam; wav++)
                calc_superstructure(wav);
}// procedure setup_supers.

// A single layer is created before added to the multilayer.
void TBlock::Add_one(int layer)
{
    double Nin = 1.0;
    int lay, wav;
    Complex betax;
    Complex Ndx[3], f[3], alfa[3], theta[3];

    for(int i=0; i < 3; i++)
    {
                Ndx[i].Re = 0;
                Ndx[i].Im = 0;
                f[i] = alfa[i] = theta[i] = Ndx[i];
    } // initialize them with 0.

    Ndx[2].Re = 1.0;
    Ndx[2].Im = 0.0;
    Ndx[1] = n[layer];
    betax = Ndx[1];
    theta[2].Re = 0.0;
    theta[2].Im = 0.0;
    PubFuns.LayerParameters(1, Ndx, f, alfa);
    Create_One_Layer(layer);
    for(wav=1; wav <= numlam; wav++)
    {
                theta[1].Re = FourPi*betax.Re*d_sav[layer] / lambdas[wav];
                theta[1].Im = FourPi*betax.Im*d_sav[layer] / lambdas[wav];
                PubFuns.Zstak(1, r_o[wav], t_o[wav], alfa, f, theta, 1, Nin, TRUE);
    }
}// procedure add_one_to_r_u.

//When a layer is taken off the overlaying part.
void TBlock::BackOff_one(int layer)
{
    double Nin = 1.0;
    int lay, wav;
    double theta_fac, d_factor, thick;
    Complex c_fac, num, r, t, temp, rxx, txx, N;
    Complex Ndx[3], f[3], alfa[3], theta[3];
    for(int i=0; i < 3; i++)
    {
                Ndx[i].Re = 0;
                Ndx[i].Im = 0;
                f[i] = alfa[i] = theta[i] = Ndx[i];
    } // initialize them with 0.

    if((layer > 1) && (layer <= NumItem))
    {
    // calculate r_o_back and t_o_back first.
    // so add a negative thicknes of next layer to r_o, etc.
    RObject LayerObject = MLayers->operator [](layer);
    N = ((PTSingleLayer)(&LayerObject))->ref_index;
    thick = ((PTSingleLayer)(&LayerObject))->thickness;
    Ndx[2].Re = 1.0;
    Ndx[2].Im = 0.0;
    Ndx[1] = N;
    theta[2].Re = 0.0;
    theta[2].Im = 0.0;
    PubFuns.LayerParameters(1, Ndx, f, alfa);
    d_factor = -FourPi*thick;
    for(wav=1; wav <= numlam; wav++)
    {
                theta_fac = d_factor / lambdas[wav];
                theta[1].Re = theta_fac*N.Re;
                theta[1].Im = theta_fac*N.Im;
                PubFuns.Zstak(1, r_o_back[wav], t_o_back[wav], alfa,f, theta, 1, Nin, TRUE);

                // Now go and calculate r_o and t_o.
                // First get new rxx and txx, the rx and tx for Layer_plus1.
                //Calc_layer_x(rxx, txx, Ndx[1], betax[1], d_sav[layer], lambdas[wav);
                rxx = ((PTSingleLayer)(&LayerObject))->rx[wav];
                txx = ((PTSingleLayer)(&LayerObject))->tx[wav];

                r = r_o[wav];
```

```
                t = t_o[wav];
                c_fac = PubFuns.Cprod(r_o_back[wav], rxx);
                c_fac.Re = 1 - c_fac.Re;
                c_fac.Im =  - c_fac.Im;
                temp = PubFuns.Cprod(t, c_fac);
                t_o[wav] = PubFuns.Cdiv(temp, txx);
                num = PubFuns.Cprod(t_o[wav], t_o[wav]);
                num = PubFuns.Cprod(num, rxx);
                temp = PubFuns.Cdiv(num, c_fac);
                r_o[wav].Re = r.Re - temp.Re;
                r_o[wav].Im = r.Im - temp.Im;
    }
    MLayers->destroy(layer);   // delete it from MLayers.
    }// if layer = ....
} // procedure Add_BackOff_one.

// Initialization of the substructure (i.e. the underlying structure).
void TBlock::setup_subs()
{
    int wav;
    Complex f, denom, two, temp;
    denom.Re = Base.Re + 1;
    denom.Im = Base.Im;
    two.Re = 2;
    two.Im = 0;
    for(wav=1; wav <= numlam; wav++)
    {
                r_o[wav].Re = -Out_F.Re;
                r_o[wav].Im = -Out_F.Im;
                temp = PubFuns.Cdiv(two, denom);
                t_o[wav].Re = temp.Re;
                t_o[wav].Im = temp.Im;
                r_o_back[wav].Re = 0.0;
                r_o_back[wav].Im = 0.0;
                t_o_back[wav].Re = 1.0;
                t_o_back[wav].Im = 0.0;
    }
}// fuction setup_subs.

//Recursive function for the calculation of the reflection coefficient of a
//multilayer structure. It is called by recall() or reflectance(...) Of the
//GA based OTFM. Also see equation (4.11).
void TBlock::Zstak(int Layer, Complex &z, Complex &t, Complex *alfa,Complex *theta, double Nin,
                   BOOL want_transmission)
{
    double n_fac;
    Complex p, p_factor, z_factor, F, negF, half_thet, den, temp;
    if (Layer <= (NumItem+1))
    {
                p = PubFuns.Compos(alfa[Layer], z);
                z = PubFuns.Cprod(PubFuns.CExponJ(theta[Layer]), p);
                if (want_transmission)
                {
                   if (Layer < (NumItem+1))
                   {
                        half_thet.Re = 0.5*theta[Layer].Re;
                        half_thet.Im = 0.5*theta[Layer].Im;

                        RObject LayerObject = MLayers->operator [](Layer);
                        F = ((PTSingleLayer)(&LayerObject))->f;

                        p_factor = PubFuns.Cprod(F, p);
                        p_factor.Re = 1 - p_factor.Re;
                        p_factor.Im = -p_factor.Im;
                        z_factor = PubFuns.Cprod(F, z);
                        z_factor.Re = 1 - z_factor.Re;
                        z_factor.Im = -z_factor.Im;
                   temp = PubFuns.Cprod(t, PubFuns.Cdiv(p_factor, z_factor));
                        t = PubFuns.Cprod(PubFuns.CExponJ(half_thet), temp);
                   }
                   else   // Layer = MaxLay + 1
                   {
                        negF.Re = -In_F.Re;
                        negF.Im = -In_F.Im;
                        den = PubFuns.Compos(negF, p);
                // ' den ' at this point is really 'r' for the previous layer.
                        den = PubFuns.Cprod(In_F, den);
                        den.Re = 1 + den.Re;
                        temp = PubFuns.Cdiv(t, den);
                        n_fac = 2*Nin/(Nin+1);
                        t.Re = n_fac*temp.Re;
```

```
                    t.Im = n_fac*temp.Im;
                } //if at last layer.
            }  // if want_transmission.
            Zstak(Layer+1, z, t, alfa, theta, Nin, want_transmission);
    }
}
// module nsqmod.


//////////////////////////
//          List.cpp      //
//////////////////////////
// Calculation on a list of Items, which are associated with training examples.
// The merit corresponding to each example is also calculated here.

#define WIN30
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <owl.h>
#include <assert.h>
#include "Linklist.h"

Item::Item(int NumLam, int MaxLay, double *Lambdas, Complex *N,double *D_sav, double RTarget, Item
*item)
{
    numlam = NumLam;
    maxlay = MaxLay;

    lambdas = new double[WaveSize];
    d_sav = new double[LayerSize];
    r_combo = new Complex[WaveSize];
    next_r_u = new Complex[WaveSize];
    n = new Complex[LayerSize];
    r = new Complex[WaveSize];
    r_best = new Complex[WaveSize];
    r_overallbest = new Complex[WaveSize];
    Rtarget = RTarget;
    for(int wav=1; wav <= numlam; wav++)
    {
            lambdas[wav] = Lambdas[wav];
    }

    for(int i=0; i <= maxlay+1; i++)
    {
            d_sav[i] = D_sav[i];
            n[i] = N[i];
    }

    Complex Out={4.0, 0.0}, In={1.0, 0.0};
    SubStrate = new TBlock(numlam, maxlay, lambdas, n, d_sav, In, Out);
    Supers = new TBlock(numlam, maxlay, lambdas, n, d_sav, In, In);

    next = item;
}


Item::~Item()
{
    delete[] lambdas;
    delete[] d_sav;
    delete[] r_combo;
    delete[] next_r_u;
    delete[] n;
    delete SubStrate;
    delete Supers;
    delete[] r;
    delete[] r_best;
    delete[] r_overallbest;
}

// Calculation of the merit when the “varying” layer is varied.
double Item::GetMerit(double dx, int layer_to_vary)
{
    PTSingleLayer ALayer;
    ALayer = new TSingleLayer(n[layer_to_vary], dx, numlam, lambdas);
    double lambda;
    double Merit = 0.0;
    for(int wav=1; wav <= numlam; wav++)
    {
            lambda = lambdas[wav];
```

```
                r[wav]=Ref_calc(r_combo[wav],SubStrate->r_o[wav],
                        Alayer->rx[wav],ALayer->tx[wav],Supers->r_o[wav],
                        Supers->t_o[wav], Supers->r_o_back[wav],Supers->t_o_back[wav]);
                Merit = Merit  + ((Rtarget-PubFuns.Cmagsq(r[wav]))
                        * (Rtarget-PubFuns.Cmagsq(r[wav])));
    }
    delete ALayer;
    return Merit;
}

BOOL List::is_empty()
{
    return head == 0 ? TRUE : FALSE;
}

// Add another element in the linked-list.
void List::append(Item *pt)
{
    if (head == 0)
                head = pt;
    else
                at_end->next = pt;

    at_end = pt;
}

void List::display()
{
}

// Remove an element from the list.
void List::remove()
{
    Item *pt = head;
    while(pt)
    {
                Item *tmp = pt;
                pt = pt->next;
                delete tmp;
    }
    head = at_end = 0;
}

// The merit for all the training examples which are presented to the OTFM.
double List::TotalMerit(double dx, int layer_to_vary)
{
    int count = 0;
    double Merit, TMerit = 0.0;
    for(Item *pt=head; pt; pt=pt->next)
    {
                count = count + 1;
                if (pt==0) break;

                Merit = pt->GetMerit(dx, layer_to_vary);
                TMerit = TMerit + Merit;
    }
    return TMerit/count;
}

void List::readin(ifstream s)
{
}


////////////////////////////////
//          Public.cpp        //
////////////////////////////////
// Complex number and thin-film model related calculations. Functions here are used
// frequently throughout the program.

#define WIN30
#include <owl.h>
#include <float.h>
#include <math.h>
#include "slayer.h"
#include "block.h"

Complex TPublic_funs::Cprod(Complex A, Complex B)
{//Complex multiplication -- minimun # of floating point operations required.
    double M1, M2, M3;
    M1 = (A.Re + A.Im)*(B.Re + B.Im);
```

```
    M2 = A.Re*B.Re;
    M3 = A.Im*B.Im;
    Complex Cprod_temp;

    Cprod_temp.Re = M2 - M3;
    Cprod_temp.Im = M1 - M2 - M3;
    return Cprod_temp;
}

Complex TPublic_funs::Cdiv(Complex D, Complex E)
{ // Complex Division.
    double R;
    Complex Q;
    E.Im = -E.Im;
    Q = Cprod(D,E);
    R = (E.Re)*(E.Re) + (E.Im)*(E.Im);
    Complex Cdiv_temp;

    Cdiv_temp.Re = Q.Re/R;
    Cdiv_temp.Im = Q.Im/R;
    return Cdiv_temp;
} // Procedure Cdiv.

Complex TPublic_funs::CExponJ(Complex A)
{
// 'EXP' evaluation with complex argument
//   z = exp(j*a)
//   where j = sqrt(-1)
    double Atten;
    Complex CExponJ_temp;
    if (A.Im == 0.0)
    {
               CExponJ_temp.Re = cos(A.Re);
               CExponJ_temp.Im = sin(A.Re);
    }
    else  // A.Im != 0.0
    {
               Atten = exp(-A.Im);
               CExponJ_temp.Re = cos(A.Re)*Atten;
               CExponJ_temp.Im = sin(A.Re)*Atten;
    }
    return CExponJ_temp;
} // Procedure CExponJ.

double TPublic_funs::Cmagsq(Complex x)
{
    // return the square of the abolute magnitude of a complex number.
    return ((x.Re)*(x.Re) + (x.Im)*(x.Im));
} // procedure cmagsq.

Complex TPublic_funs::Comp_conj(Complex x)
{
    // return complex conjugate of a complex number.
    Complex Comp_conj_temp;
    Comp_conj_temp.Re = +x.Re;
    Comp_conj_temp.Im = -x.Im;
    return Comp_conj_temp;
}// function comp_conj.

Complex TPublic_funs::Csqrt(Complex z)
{
    // return the square root of a complex number.
    double Temp;
    Temp = z.Re + sqrt(Cmagsq(z));
    Complex Csqrt_temp;
    if (Temp > 0)
    {
               Temp = sqrt(0.5*(Temp));
               Csqrt_temp.Re = Temp;
               Csqrt_temp.Im = 0.5*z.Im/Temp;
    }
    else
    {
               Csqrt_temp.Re = 0.0;
               Csqrt_temp.Im = sqrt(-z.Re);
    }
    return Csqrt_temp;
} // function Csqrt.

Complex TPublic_funs::Fresnel(Complex N)
{
```

```
    // Evaluates Fresnel Coefficient: f = (n-1) / (n+1).
    Complex F, Num, Den;
    Num.Re = N.Re - 1;
    Num.Im = N.Im;
    Den.Re = N.Re + 1;
    Den.Im = N.Im;
    return Cdiv(Num, Den);
} // Procedure Fresnel.

Complex TPublic_funs::Compos(Complex A, Complex B)
{
    // Linear fractionl transformation: c = (a+b) / (1+a*b).
    Complex X, Y;
    X.Re = A.Re + B.Re;
    X.Im = A.Im + B.Im;
    Y = Cprod(A, B);
    Y.Re = 1.0 + Y.Re;
    return Cdiv(X, Y);
}// Procedure Compos.

double TPublic_funs::Atan_ns(double x, double y)
{
    //double Pi = 3.14159265358979324;
    double HalfPi = 1.57079632679489662;
    double a;
    if (x == 0.0)
    {
               if (y == 0.0)
                  return 0.0;
               else  //y!=0
               if (y > 0.0)
                  return HalfPi;
               else // y < 0
                      return (Pi + HalfPi);
    }
    else  // x != 0.0
    if (y == 0.0)
               return 0.0;
    else // x and y <> 0
    {
               a = atan(y/x);
               if (x < 0.0)
                  return (Pi + a);
               else  // x >= 0.0
                  if (a < 0.0)
                       return (2*Pi + a);
                  else // a >= 0.0
                       return a;
    }  // x nad y <> 0
} // Atan

Complex TPublic_funs::Clog(Complex z)
{
    // return the natural logarithm of a complex number.
    Complex Clog_temp;
    Clog_temp.Re = Atan_ns(z.Re, z.Im);
    Clog_temp.Im = -0.5*log(Cmagsq(z));
    return Clog_temp;
} // function clog.

void TPublic_funs::LayerParameters(int MaxLay, Complex Ndx[],Complex *F, Complex *alfa)
{
    // procedure supplies alfa and F (fresnel) coefficients for each layer.
    int lay;
    Complex negF;
    for(lay=0; lay <= (MaxLay+1); lay++)
               F[lay] = Fresnel(Ndx[lay]);
    alfa[1] = F[1];
    for(lay=2; lay <= (MaxLay+1); lay++)
    {
               negF.Re = -F[lay-1].Re;
               negF.Im = -F[lay-1].Im;
               alfa[lay] = Compos(F[lay], negF);
    }
} // procedure layerparameters.

void  TPublic_funs::Add_a_module(Complex &Z,  Complex r_over[],  Complex r_back_over[],  Complex
                      t_over[],Complex t_back_over[], int Wav, double Nin)
{
    // add a film module for a particular wavelength to a substructure's Z.
    Complex num, den, dum, tsq, rsq;
```

```
    // transform Z from input medium to vacuum medium.
    dum.Re = -(Nin - 1) / (Nin + 1);
    dum.Im = 0.0;
    Z = Compos(dum, Z);
    tsq = Cprod(t_over[Wav], t_back_over[Wav]);
    rsq = Cprod(r_over[Wav], r_back_over[Wav]);
    num.Re = tsq.Re - rsq.Re;
    num.Im = tsq.Im - rsq.Im;
    num = Cprod(num, Z);
    num.Re = r_over[Wav].Re + num.Re;
    num.Im = r_over[Wav].Im + num.Im;
    den = Cprod(r_back_over[Wav], Z);
    den.Re = 1 - den.Re;
    den.Im = -den.Im;
    Z = Cdiv(num, den);
}// procedure add_a_module.


void TPublic_funs::Zstak(int Layer, Complex &z, Complex &t, Complex *alfa, Complex *f,
                  Complex *theta, int MaxLay, double Nin, BOOL want_transmission)
{
    // Recursive function for calculation of reflection and transmission coefficients
    // of a multilayer structure (see equation 4.11).
    double n_fac;
    Complex p, p_factor, z_factor, negF, half_thet, den, temp;
    if (Layer <= (MaxLay+1))
    {
               p = Compos(alfa[Layer], z);
               z = Cprod(CExponJ(theta[Layer]), p);
               if (want_transmission)
               {
                  if (Layer < (MaxLay+1))
                  {
                       half_thet.Re = 0.5*theta[Layer].Re;
                       half_thet.Im = 0.5*theta[Layer].Im;
                       p_factor = Cprod(f[Layer], p);
                       p_factor.Re = 1 - p_factor.Re;
                       p_factor.Im = -p_factor.Im;
                       z_factor = Cprod(f[Layer], z);
                       z_factor.Re = 1 - z_factor.Re;
                       z_factor.Im = -z_factor.Im;
                       temp = Cprod(t, Cdiv(p_factor, z_factor));
                       t = Cprod(CExponJ(half_thet), temp);
                  }
                  else   // Layer = MaxLay + 1
                  {
                       negF.Re = -f[Layer].Re;
                       negF.Im = -f[Layer].Im;
                       den = Compos(negF, p);
               // ' den ' at this point is really 'r' for the previous layer.
                       den = Cprod(f[Layer], den);
                       den.Re = 1 + den.Re;
                       temp = Cdiv(t, den);
                       n_fac = 2*Nin/(Nin+1);
                       t.Re = n_fac*temp.Re;
                       t.Im = n_fac*temp.Im;
                  } //if at last layer.
               }  // if want_transmission.
    Zstak(Layer+1, z, t, alfa, f, theta, MaxLay, Nin, want_transmission);
    }
}
```

Appendix B

Code Listing for the GA-Based OTFM

The GA-based OTFM code here is application dependant. The reader should refer to the SGA-C (Smith *et al.* 1994), a C version of Goldberg's Pascal SGA (Goldberg 1989), for a complete GA source code. The following program was coded specifically for the iris classification.

```
/*////////////////////////////////
//              sga.h           //
////////////////////////////////*/
/*--------------------------------------------------------------------------*/
/* sga.h - global declarations for main(), all variable declared herein must  */
/*         also be defined as extern variables in external.h !!!               */
/*--------------------------------------------------------------------------*/

#define LINELENGTH 80               /* width of printout */
#define BITS_PER_BYTE 8            /* number of bits per byte on this machine */
#define UINTSIZE (BITS_PER_BYTE*sizeof(unsigned))    /* # of bits in unsigned */
#include <stdio.h>

/* file pointers */
FILE *outfp, *infp;

/* Global structures and variables */
struct individual
{
    unsigned *chrom;                  /* chromosome string for the individual */
    double   fitness;                           /* fitness of the individual */
    int      xsite;                              /* crossover site at mating */
    int      parent[2];                  /* who the parents of offspring were */
    int      *utility;           /* utility field can be used as pointer to a */
               /* dynamically allocated, application-specific data structure */
};
struct bestever
{
    unsigned *chrom;        /* chromosome string for the best-ever individual */
    double   fitness;                  /* fitness of the best-ever individual */
    int      generation;                       /* generation which produced it */
};

struct individual *oldpop;                 /* last generation of individuals */
struct individual *newpop;                 /* next generation of individuals */
struct bestever bestfit;                        /* fittest individual so far */
double sumfitness;                     /* summed fitness for entire population */
double max;                                  /* maximum fitness of population */
double avg;                                  /* average fitness of population */
double min;                                  /* minumum fitness of population */
float  pcross;                                    /* probability of crossover */
float  pmutation;                                  /* probability of mutation */
int    numfiles;                                      /* number of open files */
int    popsize;                                            /* population size */
int    lchrom;                  /* length of the chromosome per individual */
int    chromsize;         /* number of bytes needed to store lchrom string */
int    gen;                                  /* current generation number */
int    maxgen;                               /* maximum generation number */
int    run;                                            /* current run number */
int    maxruns;                           /* maximum number of runs to make */
int    printstrings = 1;  /* flag to print chromosome strings (default on) */
int    nmutation;                            /* number of mutations */
int    ncross;                               /* number of crossovers */

/* Application-dependent declarations go after here... */
#define lay_size   35         /* maximum number of thin-film layers */
#define wav_size   10         /* maximum number of wavelengths can be used */

double Pi = 3.14159265358979324;
int max_film_thickness = 6;
enum BOOL {FALSE, TRUE};
```

```
struct Complex
{ double Re;
  double Im;
};

/* Thin-film data file*/
char StackFile[15];

/* global variables. */
int numlam, maxlay, NumPat, InputBit;
double MinLam, MaxLam, Delta, Bit[35];

/* declare some arrays here.*/
double d_orig[lay_size], lambdas[lay_size], Rtarget[lay_size];
struct Complex r_u[lay_size], t_u[lay_size], n[lay_size], f[lay_size], alfa[lay_size];

/* this array is used to store results.*/
double R_u[lay_size];

/* declare arrays to store iris Rtargets.*/
Complex Rtarget1[5], Rtarget2[5];

/* a linklist to store refractive indices as inputs.*/
typedef struct item {
               struct Complex Ndx[lay_size];
               struct item *next;
} ITEM, *PITEM;

PITEM current, first;
```

```
/*////////////////////////////
//         OTFM code        //
////////////////////////////*/
/* some calculations related to complex number are not included here,      */
/* as they have been described in Appendix A (see Public.cpp)              */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "external.h"
#define NEW(PP) ((PP) = (PITEM)malloc(sizeof(ITEM)))
#define TEST(PP) if (NEW(PP) == NULL) \
               { printf("memory error\n"); return;}

extern struct Complex Cdiv(struct Complex D, struct Complex E);
extern double Cmagsq(struct Complex x);
extern void LayerParameters(int maxlay, struct Complex Ndx[],
                            struct Complex *f,  struct Complex *alfa);
extern struct Complex Fresnel(struct Complex N);
extern Zstak(int Layer, struct Complex *z, struct Complex *t,
               struct Complex *alfa,struct Complex *f, struct Complex *theta,
               int maxlay, double Nin,enum BOOL want_transmission);

/*setting up the Substrate, i.e. the underlying structure */
void setup_subs( struct Complex *r_u, struct Complex *t_u, int NumLam, struct
    Complex Ndx[])
{
    int wav;
    struct Complex f, denom, two;
    f = Fresnel(Ndx[0]);
    denom.Re = Ndx[0].Re + 1;
    denom.Im = Ndx[0].Im;
    two.Re = 2;
    two.Im = 0;
    for(wav=1; wav <= NumLam; wav++)
    {
               r_u[wav].Re = -f.Re;
               r_u[wav].Im = -f.Im;
               t_u[wav] = Cdiv(two, denom);
    }
}


/* calculation of the overall reflection coefficient of a multilayer */
/* structure */
void reflectance(double *lambdas, struct Complex *n, double *d_orig,
                  struct Complex *r_u, struct Complex *t_u,
                  struct Complex      *alfa, struct Complex *f)
{
    double lambda;
    int wav, i, lay;
    struct Complex theta[lay_size];

    LayerParameters(maxlay, n, f, alfa);
    setup_subs(r_u, t_u, numlam, n);
    for(wav=1; wav <= numlam; wav++)
    {
               lambda = lambdas[wav];

               for(i=0; i < lay_size; i++)
               {
                  theta[i].Re = 0;
                  theta[i].Im = 0;
               }
               for(lay=1; lay <= maxlay; lay++)
                  theta[lay].Re = ((4*Pi)*n[lay].Re)*d_orig[lay]/lambda;
    Zstak(1, &r_u[wav], &t_u[wav], alfa, f, theta, maxlay, n[0].Re,TRUE);

    }
}

/* read in training examples */
void Getinput(struct Complex *Ndx, double *d_orig, double *lambdas,
           double *Rtarget)
{
    int i, wav, j, lay;
    double Nin, Nout,LamFac;
    char StackFileName[11], keyword[18];
    FILE *datafile;

    fprintf(outfp, " Data file(parameters for thin-film)---> ");
    fscanf(infp, "%11s", StackFileName);
```

```
strcat(StackFileName,".stk");

if ((datafile = fopen(StackFileName, "r"))==NULL)
{
           printf("File cannot be opened\n");
           return;
}

fscanf(datafile, "%15s", &StackFile);

fscanf(datafile, "%d%lf%lf", &maxlay, &Nin, &Nout);

/* Assign layers with various reflective indices. */
for(lay = 0; lay <= (maxlay + 1); lay++)
{
           if (lay%2)
              Ndx[lay].Re = 1.2;
           else
              Ndx[lay].Re = 4.1;

           Ndx[lay].Im = 0.0;
              d_orig[lay] = 1.0;
}

Ndx[0].Re = Nout;
Ndx[maxlay+1].Re = Nin;

fscanf(datafile, "%18s%d", &keyword, &numlam);
fscanf(datafile, "%18s%lf%lf", &keyword, &MinLam, &MaxLam);
fscanf(datafile, "%18s%d", &keyword, &NumPat);
fscanf(datafile, "%18s%lf", &keyword, &Delta);
fscanf(datafile, "%18s%d", &keyword, &InputBit);

for(lay = 0; lay <= (maxlay + 1); lay++)
           Bit[lay] = 0;

/*create the list*/
TEST(first)
current = first;

for(i=1; i <= NumPat; i++)
{
           TEST(current->next)
           current = current->next;
           for(lay = 0; lay <= (maxlay + 1); lay++)
           {
              current->Ndx[lay].Re =  Ndx[lay].Re;
              current->Ndx[lay].Im =  Ndx[lay].Im;
           }
     for(j=1; j <= InputBit; j++)
     {
           fscanf(datafile, "%lf", &Bit[j]);
           current->Ndx[j].Re =  Ndx[j].Re + Delta*Bit[j];
     }
    fscanf(datafile, "%3s", &keyword); /*skip a comma.*/
}
current->next = NULL;

fscanf(datafile, "%18s", &keyword);

/* when using iris data, Rtarget is a complex number.*/
for(i=1; i <= 3; i++)
{
  fscanf(datafile, "%lf", &Rtarget1[i].Re);
  fscanf(datafile, "%lf", &Rtarget1[i].Im);
}

for(i=1; i <= 3; i++)
{
  fscanf(datafile, "%lf", &Rtarget2[i].Re);
  fscanf(datafile, "%lf", &Rtarget2[i].Im);
}

for(i=1; i <= 3; i++)
{
  fscanf(datafile, "%lf", &Rtarget3[i].Re);
  fscanf(datafile, "%lf", &Rtarget3[i].Im);
}

fclose(datafile);
```

```
    if (numlam == 1)
               LamFac = 0.0;
    else LamFac = (MaxLam - MinLam) / (numlam - 1);

    for(wav=1; wav <= numlam; wav++)
               lambdas[wav] = MinLam + LamFac*(wav - 1);
}


/* The code here is used for the iris classification*/
void TestData()
{
    /* read in the iris test data file */
    int j, wav, lay, count;
    double localMerit1, localMerit2, localMerit3;
    struct Complex Ndx[lay_size];
    char Test[25], keyword[10];
    FILE *testfile;

    sprintf(Test,"%s.tst", StackFile);

    if ((testfile = fopen(Test, "r"))==NULL)
    {
    printf("File cannot be opened\n");
    return;
    }

    for(count=1; count <= 150; count++)
    {
       if (count==31)
        fprintf(outfp, "Results on training data class1:\n");
       else if (count==71)
        fprintf(outfp, "Results on training data class2:\n");
       else if (count==111)
        fprintf(outfp, "Results on training data class3:\n");

       for(lay = 0; lay <= (maxlay + 1); lay++)
               Ndx[lay] = n[lay];

       for(j=1; j <= InputBit; j++)
       {
               fscanf(testfile, "%lf", &Bit[j]);
               Ndx[j].Re = Ndx[j].Re + Delta*Bit[j];  */
       }

       fscanf(testfile, "%3s", &keyword); /* skip a comma.*/

       reflectance(lambdas, Ndx, d_orig, r_u, t_u, alfa, f);

       localMerit1 = localMerit2 = localMerit3 = 0.0;
       for(wav=1; wav <= numlam; wav++)
       {
          localMerit1 = localMerit1
               + ((r_u[wav].Re - aver1[wav].Re)*(r_u[wav].Re - aver1[wav].Re)
               +(r_u[wav].Im - aver1[wav].Im)*(r_u[wav].Im - aver1[wav].Im))/numlam;
          localMerit2 = localMerit2
               + ((r_u[wav].Re - aver2[wav].Re)*(r_u[wav].Re - aver2[wav].Re)
               +  (r_u[wav].Im - aver2[wav].Im)*(r_u[wav].Im - aver2[wav].Im))/numlam;
          localMerit3 = localMerit3
               + ((r_u[wav].Re - aver3[wav].Re)*(r_u[wav].Re - aver3[wav].Re)
               +  (r_u[wav].Im - aver3[wav].Im)*(r_u[wav].Im - aver3[wav].Im))/numlam;
        fprintf(outfp, "{%lf, %lf}", r_u[wav].Re, r_u[wav].Im);
    }

    fprintf(outfp, "Merit: %lf : %lf : %lf", localMerit1, localMerit2, localMerit3);
    if ((localMerit1 < localMerit2) && (localMerit1 < localMerit3))
       fprintf(outfp, " Plant1\n");
    if ((localMerit2 < localMerit1) && (localMerit2 < localMerit3))
       fprintf(outfp, " Plant2\n");
    if ((localMerit3 < localMerit1) && (localMerit3 < localMerit2))
       fprintf(outfp, " Plant3\n");

  }
  fclose(testfile);
}
```

```
/*////////////////////////////////
//            app2.c            //
////////////////////////////////*/

/*-------------------------------------------------------------------------*/
/* application dependent routines, change these for different problem       */
/*                                                                         */
/* This example application interprets chromosomes as concatenated strings */
/* of binary integers of user-specified length (field_size).                */
/* Fitness is simply the length of this integer vector squared.          */
/*-------------------------------------------------------------------------*/

#include <stdlib.h>
#include <math.h>
#include "external.h"

static int field_size, vec_size;
extern double Cmagsq(struct Complex x);
extern void reflectance(double *lambdas, struct Complex *n, double *d_orig,
              struct Complex *r_u, struct Complex *t_u, struct Complex *alfa, struct Complex *f);
extern void Getinput(struct Complex *Ndx, double *d_orig, double *lambdas,double *Rtarget);

void application()
/* This routine should contain any application-dependent      */
/* computations that should be performed before each GA cycle */
/* called by main() */
{
  int wav;

  for(wav=1; wav <= numlam; wav++)
  {
    aver1[wav].Re = aver1[wav].Im = 0.0;
    aver2[wav].Re = aver2[wav].Im = 0.0;
    aver3[wav].Re = aver3[wav].Im = 0.0;
  }
}


void app_data()
/* application dependent data input, called by init_data() */
/* ask your input questions here, and put output in global variables */
/* In this example application, the utility pointer of each individual */
/* is assigned a vector of integers that will be used to store the */
/* interpreted chromosome. */
{
    int size = UINTSIZE;

    if(lchrom < UINTSIZE) size = lchrom;

   /* user must specify length of concatenated integers in the chromosome. */
    fprintf(outfp," Enter field size (must be less than %d) ->", size);
    fscanf(infp,"%d",&field_size);
    vec_size = lchrom/field_size;
    if((((float)lchrom/(float)field_size)-vec_size) > 0.0) vec_size++;
    /* here added is for data of thinfilm layers.  */
    Getinput(n, d_orig,lambdas,Rtarget);

}


void app_free()
/* This routine should free any memory allocated */
/* in the application-dependent routines, called by freeall() */
{
    int i;
    for(i = 0; i < popsize; i++)
    {
        free(newpop[i].utility);
        free(oldpop[i].utility);
    }
}


void app_init()
/* Application dependent initialization routine called by initialize().   */
{
}

void app_initreport()
/* Application-dependent initial report called by initialize() */
{
```

```
    if(vec_size > lchrom/field_size)
    {
fprintf(outfp," Each chromosome interpreted as %d %d-bit integers",vec_size-1, field_size);
fprintf(outfp," and one %d-bit integer.\n",lchrom-((vec_size-1)*field_size));
    }
    else
fprintf(outfp,"Each chromosome interpreted as %d %d-bit integers.\n",vec_size, field_size);
}

app_malloc()
/* application dependent malloc() calls, called by initmalloc() */
{
    unsigned nbytes;
    int i;

    nbytes = vec_size * sizeof(int);
    for(i = 0; i < popsize; i++)
    {
        if((newpop[i].utility = (int *) malloc(nbytes)) == NULL)
       nomemory(stderr,"newpop utility");
        if((oldpop[i].utility = (int *) malloc(nbytes)) == NULL)
        nomemory(stderr,"oldpop utility");
    }
}


app_report()
/* Application-dependent report, called by report() */
{
    int i, j, wav, lay;

    fprintf(outfp, "******Found solution --- Layer thicknesses:******\n");
    for(lay=1; lay <= maxlay; lay++)
        fprintf(outfp,"%lf  ", d_orig[lay]);

    /* Print the initial setup of thin-film model.*/
     fprintf(outfp, "\n******Thin-film model initialised with data file =%15s******", StackFile);
    fprintf(outfp, "\nmaxlay = %d  numlam = %d ", maxlay, numlam);
    fprintf(outfp, " MaxLam = %f   MinLam = %f", MaxLam, MinLam);
    fprintf(outfp, " Scaling factor (Delta) = %f", Delta);
    fprintf(outfp, "\nRefractive indecies %lf and %lf used", n[1].Re, n[2].Re);
    fprintf(outfp, "  OverAllBestMerit = %f", (10 - bestfit.fitness));
    fprintf(outfp,"\n-----------------------------------------------------------------------\n");
    for(wav=1; wav <= numlam; wav++)
         fprintf(outfp,"\nTargets for class 1 is %f", Rtarget1[wav]);

    for(wav=1; wav <= numlam; wav++)
         fprintf(outfp,"\nTargets for class 2 is %f", Rtarget2[wav]);
    fprintf(outfp, "\n");

    fprintf(outfp, "average for plant 1: ");
    for(wav=1; wav <= numlam; wav++)
    fprintf(outfp, "{%lf, %lf}", aver1[wav].Re, aver1[wav].Im);
    fprintf(outfp, "\n");

    fprintf(outfp, "average for plant 2: ");
    for(wav=1; wav <= numlam; wav++)
    fprintf(outfp, "{%lf, %lf}", aver2[wav].Re, aver2[wav].Im);
    fprintf(outfp, "\n");

    fprintf(outfp, "average for plant 3: ");
    for(wav=1; wav <= numlam; wav++)
    fprintf(outfp, "{%lf, %lf}", aver3[wav].Re, aver3[wav].Im);
    fprintf(outfp, "\n");

    /* Test the model with novel iris examples.*/
    TestData();
}


app_stats(pop)
/* Application-dependent statistics calculations called by statistic() */
/* This an example used for the iris classification.
struct individual *pop;
{
    int i, wav, start, stop;
    for(i = 0; i < vec_size; i++)
    {
        /* section of chromosome containing current integer field */
        start = (i * field_size) + 1;
        stop  = ((i + 1) * field_size);
```

```
        /* check if enough bits remain, if not, interpret as a short field */
        if(stop > lchrom) stop = lchrom;

        /* convert bit field in chromosome to an integer    */
        /* and store it in utility array.  Then compute     */
        /* chromosome fitness as sum of squares of integers */
    d_orig[i+1] = (double)ithruj2int(start, stop, (&bestfit)->chrom)/(256*40);
    }

 for(current = first, current = current->next, i=1; current != NULL; current = current->next, i++)
  {
      reflectance(lambdas, current->Ndx, d_orig, r_u, t_u, alfa, f);
      for(wav=1; wav <= numlam; wav++)
     {
               if (i <=40)
               {
               aver1[wav].Re = aver1[wav].Re + (r_u[wav].Re)/40;
               aver1[wav].Im = aver1[wav].Im + (r_u[wav].Im)/40;
               }
               if ((i > 40) && (i <= 80))
               {
               aver2[wav].Re = aver2[wav].Re + (r_u[wav].Re)/40;
               aver2[wav].Im = aver2[wav].Im + (r_u[wav].Im)/40;
               }
               if ((i > 80) && (i <= 120))
               {
               aver3[wav].Re = aver3[wav].Re + (r_u[wav].Re)/40;
               aver3[wav].Im = aver3[wav].Im + (r_u[wav].Im)/40;
               }
     }
  }
}


objfunc(critter)
/* Application dependent objective function */
struct individual *critter;
{
    int i, wav, start, stop;
    /* Interpret each chromosome as a vector of concatenanted integers */
    critter->fitness = 0;

    for(i = 0; i < vec_size; i++)
    {
        /* section of chromosome containing current integer field */
        start = (i * field_size) + 1;
        stop  = ((i + 1) * field_size);

        /* check if enough bits remain, if not, interpret as a short field */
        if(stop > lchrom) stop = lchrom;

        /* convert bit field in chromosome to an integer    */
        /* and store it in utility array.  Then compute     */
        /* chromosome fitness as sum of squares of integers */
        critter->utility[i] = ithruj2int(start, stop, critter->chrom);

    /* Note that the scale should be changed simutaniously with the one in app_report().*/
    /* it depents on how many bits we use for a chromosome */
    d_orig[i+1] = (double)critter->utility[i]/(256*40);
    }

 for(current = first, current = current->next, i=1; current != NULL; current = current->next, i++)
  {
      reflectance(lambdas, current->Ndx, d_orig, r_u, t_u, alfa, f);
      for(wav=1; wav <= numlam; wav++)
      {
               R_u[wav] = Cmagsq(r_u[wav]);
               if (i <= 40)
               critter->fitness += (Rtarget1[wav].Re  - R_u[wav].Re)
                                *(Rtarget1[wav].Re - R_u[wav].Re);
               if ((i > 40) && (i <= 80))
               critter->fitness += (Rtarget2[wav].Re - R_u[wav].Re)
                                *(Rtarget2[wav].Re - R_u[wav].Re);
               if ((i > 80) && (i <= 120))
               critter->fitness += (Rtarget3[wav].Re - R_u[wav].Re)
                                *(Rtarget3[wav].Re - R_u[wav].Re);
      }
  }
    critter->fitness = 10.0 - critter->fitness/(numlam*NumPat);
}
```

Bibliography

Anderson, E. (1935). "The Iris of the Gaspe Peninsula." *Bulletin of the American Iris Society*, vol **59**, pp.2-5.

Andrews, R., Diederich J. & Tickle, A.B. (1995). "A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks." Neurocomputing Research Centre, Queensland University of Technology, Queensland, Australia. pp.1-37.

Apfel, J.H. (1972). "Graphics in Optical Coating Design." *Applied Optics*, **11**: 6, pp.1303-1312.

Baldwin, G.C. (1969). *An Introduction to Nonlinear Optics*. New York: Plenum Press.

Bechtel, W. & Abrahamsen, A. (1991). *Connectionism and the Mind - An Introduction to Parallel Processing in Networks.* Cambridge, Massachusetts: Basil Blackwell Ltd.

Beightler, C.S., Phillips, D.T. & Wilde, D.J. (1979). *Foundations of Optimization.* Englewood Cliffs, N. J.: Prentice-Hall, Inc.

Borland International (1991). *OBJECTWINDOWS for C++ – User's Guide*. Scotts Valley, CA: Borland International.

Born, M. & Wolf, E. (1980). *Principles of Optics.* New York: Pergamon Press.

Box, G.E.P. & Jenkins, G.M. (1970), *Time Series Analysis, Forecasting and Control*. San Francisco: Holden Day.

Case, W. E. (1983). "New Synthesis Method for Optical Thin-Film Coatings." *Applied Optics*, **22**: 4, pp.4111-4117.

Cherkassky, V., Friedman, J.H. & Wechsler, H. (1994). *From Statistics to Neural Networks – Theory and Pattern Recognition Applications.* NATO ASI Series F: Computer and System

Sciences, vol.136. Berlin Heidelberg: Springer-Verlag.

Chopra, K.L. (1969). "Optical Properties of Thin-Film." In *Thin Film Phenomena*, pp.721-787. McGraw-Hill Book Company.

Chopra, K.L. & Kaur, I. (1983). *Thin Film Device Applications*. New York: Plenum Press.

Clark, A. & Lutz, R. (1992). "Introduction." In *Connectionism in Context*, pp.1-15. Edited by Clark, A. & Lutz, R. Berlin: Springer-Verlag.

Cohen, P.R. (1995). *Empirical Methods for Artificial Intelligence*. Cambridge, Mass.: MIT Press.

Coveney, P. & Highfield, R. (1995). *Frontiers of Complexity - The Search for Order in a Chaotic World*. New York: Ballantine Books, Inc.

Das, P.K. (1991). *Optical Signal Processing – Fundamentals*. Berlin Heidelberg: Springer-Verlag.

Davis, L. & Steenstrup, M. (1987). "Genetic Algorithms and Simulated Annealing: An Overview." In *Genetic Algorithms and Simulated Annealing*, pp.1-11. Edited by Davis, L. London: Pitman.

Davis, L. (1987). *Genectic Algorithms and Sinulated Annealing*. Pitman, London: Morgan Kaufmann Publishers, Inc., Los Altos, California.

De Jong, K., Spears, W.M. & Gordon, D.F. (1993). "A Knowledge-Intensive Genetic Algorithm for Supervised Learning." In *Genetic Algorithms for Machine Learning*, pp.5-29. Edited by J.J. Grefenstette. Massachusetts: Kluwer Academic Publishers.

Dietterich, T.G. (1996). "Statistical Tests for Comparing Supervised Classification Learning Algorithms." *Technical Report*. Department of Computer Science, Oregon State University. pp.1-20.

Dobrowolski, J.A., Ho, F.C., Belkind, A. & Koss, V.A. (1989). "Merit Functions for More Effective Thin Film Calculations." *Applied Optics*, **28**: 14, pp.2824-2831.

Dobrowolski, J.A. & Kemp, R.A. (July 1990). "Refinement of Optical Multilayer Systems with Different Optimization Procedures," *Applied Optics*, **29**: 19, pp.2876-2893.

Dobrowolski, J.A. & Piotrowski, S.H.C. (1982). "Refractive Index as a Variable in the Numerical Design off Optical Thin Film Systems." *Applied Optics*, **21**: 8, pp.1502-1511.

Dorffner, G. (1993). "How Connectionism Can Change AI and the Way We Think About Ourselves." *Applied Artificial Intelligence*, **7**: pp.59-85.

Duvillier, J., Killinger, M., Heggarty, K., Yao, K.& de Bougrenet de la Tocnaye, J. (1994). "All-Optical Implementation of a Self-Organizing Map: a Preliminary Approach." *Applied Optics*, **33**: 2, pp.258-266.

Farhat, N.H., Psaltis, D., Prata, A. & Paek, E. (1985). "Optical Implementation of the Hopfield Model." *Applied Optics*, **24**:10, pp.1469-1475.

Farmer, J.D. (1990). "A Rosetta Stone For Connectionism." *Physica D*, **42**: pp.153-187.

Feitelson, D. G. (1988). *Optical Computing, A Survey for Computer Scientists.* London, England: The MIT Press.

Feldman, J.A. & Ballard, D.H. (1982). "Connectionist Models and Their Properties." *Cognitive Science, **6***: pp.205-254.

Fisher, R.A. (1936). "The Use of Multiple Measurements in Taxonomic Problems." *Annual Eugenics,* **7**, Part II, pp.179-188.

Fisher, D.H. & McKusick, K.B. (1989). "An Empirical Comparison of ID3 and Back-propagation." *Proc. 11th International Joint Conference on Artificial Intelligence, Detroit.* pp.788-793.

Flexer, A.(1996). "Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice." In Trappl R., *Cybernetics and Systems'96, Proceedings of the 13th European Meeting on Cybernetics and Systems Research*. Austrian Society for Cybernetic Studies, Vienna, 2 vols. pp.1005-1008.

Fodor, J.A. & Pylyshyn, Z.W. (1988). "Connectionism and Cognitive Architecture: A Critical Analysis." *Cognition*, **28**: pp.3-71.

Foulds, L.R. (1981). *Optimization Techniques - An Introduction*. New York: Springer-Verlag New York Inc.

Furman, Sh.A. & Tikhonravov, A.V.(1992). *Basics of Optics of Multilayer Systems*. Editions Frontieres, B.P.33, 91192 Gif-sur-Yvette Cedex - France.

Gallant, S.I. (1988). "Connectionist Expert Systems". *Communication of the ACM*, **31**:2, pp.152-169.

Garzon, M.(1995). *Models of Massive Parallelism - Analysis of Cellular Automata and Neural Networks*. Berlin, New York: Springer-Verlag.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.

Grefenstette, J. J.(1987). "Incorporating Problem Specific Knowledge into Genetic Algorithms." In *Genetic Algorithms and Simulated Annealing*,pp.42-60. Edited by Davis, L. London: Pitman.

Greiner, H. (1996). "Robust Filter Design with Evolutionary Stratigies." *Applied Optics*, Sep.-Oct., pp. 1-20.

Harnad, S. (1992). "Connecting Object to Symbol in Modelling Cognition." In *Connectionism in Context*, pp.75-90. Edited by Clark, A. & Lutz, R. Berlin: Springer-Verlag.

Hart, A. (1992). "Using Neural Networks for Classification Tasks - Some Experiments on Datasets and Practical Advice." *Journal of the Operational Research Society* **43**:3, pp.215-226.

Hertz, J., Krogh, A. & Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation*, Redwood City, CA: Addison-Wesley Publishing Company.

Hillis, W.D. (1985). *The Connectionist Machine*. Cambridge: MIT Press.

Hinton, G.E. (1989). "Connectionist Learning Procedures." *Artificial Intelligence*, **40**: pp.185-189.

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

Janikow, C.Z. (1993). "A Knowledge-Intensive Genetic Algorithm for Supervised Learning." In *Genetic Algorithms for Machine Learning*, pp.33-72. Edited by Grefenstette, J.J. Massachusetts: Kluwer Academic Publishers.

Khanna, T. (1990). *Foundations of Neural Networks.* Reading Mass: Addison-Wesley.

Kock, G. & Serbedzija, N.B. (1993). "Object-Oriented and Functional Concepts in Artificial Neural Network Modelling." *Proceeding of 1993 International Joint Conference on Neural Networks*. pp.923-927.

Kosko, B. (1992). *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Englewood Ciffs, NJ: Prentice Hall.

Langton, C.G.(1990). "Computation at the Edge of Chaos: Phase Transition and Emergent Computation." *Physica D*, **42**, pp.12-37.

Levi, L. (1980). *Applied Optics.* Vol II. New York: John Wiley.

Li, L. & Dobrowolski, J.A. (1992). "Computation Speeds of Different Optical Thin-Film Synthesis Methods." *Applied Optics,* **31**: pp.3790-3799.

Li, X. & Purvis, M.K. (1996). "Using Genetic Algorithms for an Optical Thin-Film Learning Model." *1996 International Symposium on Multi-Technology Information Processing*. Hsin-Chu, Taiwan, Republic of China.

Li, X. & Purvis, M.K. "Connectionist Learning Architecture Based on an Optical Thin-Film Multilayer Model", submitted to *Applied Optics*, August 1997.

Liddell, H.M. (1981). *Computer-aided Techniques for the Design of Multilayer Filters.* Adam Hilger, Bristol.

Massaro, D.W. (1988). "Some Criticisms of Connectionist Models of Human Performance." *Journal of Memory and Language,* **27**, pp.213-234.

Masters, T. (1995). *Advanced Algorithms for Neural Networks: A C++ Sourcebook*. NY:John Wiley and Sons.

McAulay, A.D.(1989). "Optical Interconnections for Real-Time Symbolic and Numeric Processing." In *Optical Computing - Digital and Symbolic,* pp.363-403. Edited by Arrathoon, R. New York: Marcel Dekker Inc.

McClelland, J.L. (1988). "Connectionist Models and Psychological Evidence." *Journal of Memory and Language,* **27**: pp.107-123.

McClelland, J.L., Rumelhart, D.E., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Vol. II). Cambridge, MA: Bradford Books.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structure = Evolution Programs - Third, Revised and Extended Edition*. Berlin, New York: Springer-Verlag.

Michalski, R.S., Mozetic, I., Hong, J. & Lavrac, N. (1986). "The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains." In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*, Philadelphia, Pa., pp.1041-1045.

Minsky, M.L. & Papert, S.A. (1969). *Perceptrons*. Cambridge: MIT Press.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. Cambridge, Massachusetts: The MIT Press.

Mooney, R., Shavik, J., Towell G. & Gove, A. (1989). "An Experimental Comparison of Symbolic and Connectionist Learning Algorithms." *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, pp.775-780.

Neff, J.A. & Kushner, B.G. (1989). "Symbolic Computing and Artificial Intelligence." In *Optical Computing - Digital and Symbolic*, pp.279-354. Edited by Arrathoon, R. New York: Marcel Dekker Inc.

Pao, Y.H. (1989). *Adaptive Pattern Recognition and Neural Networks.* Reading Mass: Addison-Wesley.

Pedrycz, W., Lam, P.C. & Rocha, A.F. (1995). "Distributed Fuzzy System Modeling." *IEEE Transactions on Systems, Man, and Cybernetics*, vol.25, No. 5, May, pp.769-780.

Prechelt, L. (1995). "Some Notes on Neural Learning Algorithm Benchmarking." *Neurocomputing*. pp.1-4.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P. (1992). *Numerical Recipes in C - The Art of Scientific Conmputing (Second Edition)*. New York: Cambridge University Press.

Purvis, M. (1982). "Thin-Film Calculation." *Technical Report*, University of Columbia, pp.1-5. (Unpublished).

Purvis, M. & Li, X. (1993). "Connectionist Computation Based on an Optical Thin-Film Model." In *Proc. of the First New Zealand International Two Stream Conference on Artificial Neural Networks and Expert System*,pp.130-33. Edited by Kasabov, N.K., Los Alamitos, California: IEEE Computer Society Press.

Purvis, M. & Li, X. (1995). "Connectionist Learning Using an Optical Thin-Film Model." *Proceedings of the 2nd New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pp.63-66, *IEEE Computer Society Press*, Los Alamitos, California.

Purvis, M.K. & Li, X. (1997). "Connectionist Learning Using an Optical Thin-Film Model." *Proceeding of the 15th World Congress on Scientific Computation, Modelling and Applied Mathematics - Artificial Intelligence and Computer Science*, **4**:pp.239-244. Edited by Achim Sydow, Berlin: Wissenschaft & Technik Verlag.

Raghavan, R. (1993). "Cellular Automata in Pattern Recognition." *Information Sciences*, **70**: pp.145-177.

Rigler, A.K. & Pegis, R.J.(1980). "Optimization Methods in Optics." In *The Computer in Optical Research*. Edited by Frieden, B.R. Springer-Verlag.

Ripley, B.D. (1993). "Statistical Aspects of Neural Networks." In *Networks and Chaos - Statistical and Probabilistic Aspects*, pp.41-123. Edited by Barndorff-Nielsen, O.E., Jensen, J.L. and Kendall, W.S. London: Chapman & Hall.

Rumelhart, D.E., McClelland, J.L. & the PDP research group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*: *Foundations* (Vol. I). Cambridge, Massachusetts, London, England: MIT Press.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1986). "Learning Internal Representations by Error Propagation." In *Parallel Distributed Processing* (Vol. I), chapter 8. See Rumelhart, D.E., McClelland, J.L., *et al*.

Serra, R. & Zanarini, G. (1990). *Complex Systems and Cognitive Processes*. Berlin: Springer-Verlag.

Simpson, P.K. (1990). *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations.* New York: Pergamon Press.

Smith, R.E., Goldberg, D.E. & Earickson, J.A.(1994). "SGA-C: A C-language Implementation of a Simple Genetic Algorithm." *TCGA Report*, No. 91002. The University of Alabama.

Sugeno, M. & Yasukawa, T. (1993). "A Fuzzy-Logic-Based Approach to Qualitative Modeling." *IEEE Transactions on Fuzzy Systems*, **1**: 1, pp.7-31.

van der Smagt, P.P. (1994). "Minimisation Methods for Training Feed-forward Neural Networks." *Neural Networks*, **7**: 1, pp.1-11.

Wagner, K. & Psaltis, D. (1987). "Multilayer Optical Learning Networks." *Applied Optics*, **26**: 23, pp.5061-5077.

Weiss, S.M. & Kapouleas, I. (1989). "An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods." In *Proc. 11th International Joint Conference on Artificial Intelligence*, pp.781-787, Detroit.

Weiss, S.M. & Kulikowski, C.A. (1991). *Computer Systems That Learn*. California:Morgan Kaufmann.

Whitley, D. (1993). "A Genetic Algorithm Tutorial." *Technical Report* cs-93-103, Nov. 10, Department of Computer Science, Colorado State University.

Shang, Y. & Wah, B.W. (1996). “Global Optimization for Neural Network Training.” *Computer*: March, pp.45-54.