Design Methods for Cellular Neural Networks
with Minimum Number of Cloning Template Coefficients

Rahim Akbari-Dilmaghani

Thesis submitted to the University of London
for the Degree of Doctor of Philosophy
April 1998
Abstract

Over the past few years there has been an intense interest in the development of Neural Networks as a new computational paradigm. The first chapter of this thesis provides an explanation of the concepts and motivations behind such a research effort, and speculation on the future of neural networks.

The effectiveness of any neural network model in the execution of a cognitive task in real-time fashion is strongly dependent on the technology used to implement it. From the wide spectrum of technologies proposed for the implementation of neural networks reviewed in Chapter 1, analogue VLSI architectures appear attractive. However the high degree of interconnectivity required by neural networks to emulate neurobiological systems is ultimately constrained by VLSI hardware. Thus the nearest neighbour interactive property of Cellular Neural Networks (CNNs) make them ideal candidates for VLSI implementation.

However, there are some CNN applications which require a neighbourhood size of greater than one nearest neighbour interconnection (r > 1). This thesis describes two powerful methods for the implementation of large-neighbourhood CNNs. The first method uses spatially varying bias terms to achieve the virtual expansion of cloning templates in VLSI implementations of large-neighbourhood CNNs. The second method employs non-linear template coefficients to achieve the virtual expansion of cloning templates in VLSI implementation of large-neighbourhood CNNs. An efficient circuit for the VLSI implementation of the proposed non-linear template coefficients is described.

In general, the cloning templates which define the desired performance of CNNs can be found by solving a set of system design inequalities. This thesis describes a new learning algorithm for CNNs which offers the following advantages: (i) ease of design for the set of design inequalities, (ii) fast convergence rate (i.e., in most cases the solution of the system design inequalities are obtained in a single iteration), (iii) it acts as test for the functionality of the design. Also, a redundancy method is presented to reduce the number template coefficients to a level that can be matched to requirements of current microelectronic technology.

To confirm the viability of proposed methods theoretical analysis and computer simulation results are presented.
Acknowledgement

A PhD similar to other goal-oriented real life tasks is an journey to an unknown territory. You can find a lot of enjoyable periods through this journey, but there is no an period which could be characterised as an easy period.

I would like to thank my wife Ladan Monfered Niaki for her support and encouragement, without her my task would have been impossible. I am most grateful to my supervisor Dr John Taylor for his encouragement and guidance’s. Many thanks to Dr David Haigh and Dr David Selviah for their support and guidance’s during my PhD.

I wish to thank University College London for providing an excellent research environment. I would also like to thank The Graduate School of UCL for providing me travel grants, allowing me to attend international conferences where some of my new ideas were formulated. Many thanks to the computer support staff, namely Leon Chernack and Lee Heagney, for sorting out the workstations for me on the frequent occasions.

In addition, I wish to thank all my colleagues at room 621 for their support and friendship which is an important part of a successful PhD.

Finally, I would like to express my appreciation to the Engineering and Physical Sciences Research Council for making this work possible by funding the project over the past three years.
To my Mum, Dad and Wife
Chapter 1

Neural Networks

1.1) Introduction
1.2) What are Neural Networks?
1.3) Attractive Features of Neural Networks
1.4) Applications of Neural Networks
1.5) The Future of Neural Networks
1.6) Taxonomy of Neural Networks
1.7) Implementation of Neural Networks
1.8) Summary and Conclusions
1.9) Research Topics
1.10) References of Chapter 1

Chapter 2

Cellular Neural Networks (CNNs)

2.1) Introduction

(2.1) Introduction
(1.10) References of Chapter 1
(1.9) Research Topics
(1.8) Summary and Conclusions
(1.7) Implementation of Neural Networks
(1.6) Taxonomy of Neural Networks
(1.5) The Future of Neural Networks
(1.4) Applications of Neural Networks
(1.3) Attractive Features of Neural Networks
(1.2) What are Neural Networks?
(2.2) Basis of Cellular Neural Networks  41
(2.3) CNNs with Non-Linear and Delay-Type Template Elements  47
(2.4) The Full signal Range (FSR) Model of CNNs  47
(2.5) Discrete-Time Cellular Neural Networks (DT-CNNs)  48
(2.6) Effect of Boundary Cells on the Behaviour of CNNs  49
(2.7) Multi-Layer Cellular Neural Networks (ML-CNNs)  49
(2.8) Information Processing and Operating Modes of CNNs  50
(2.9) Stability of Cellular Neural Networks  51
(2.10) Applications of Cellular Neural Networks  53
(2.11) Design Methods for Cellular Neural Networks  53
(2.12) Summary and Conclusions  57
(2.13) References of Chapter 2  58

Chapter 3
Implementation of Large-Neighbourhood Cellular Neural Networks

(3.1) Introduction  62
(3.2) Linear Threshold CNNs  62
(3.3) Implementation of Large-Neighbourhood CNNs with Spatially Varying Bias Terms  66
(3.4) Implementation of Large-neighbourhood CNNs using Non-Linear Cloning Templates  69
(3.5) The proposed Non-Linear Coefficient Circuit  71
(3.6) Simulation Results  72
Chapter 4

Efficient Design Methods for Cellular Neural Networks

(4.1) Introduction

(4.2) The Learning Problem in CNNs

(4.3) Zou’s Learning Algorithm for CNNs

(4.4) The Redundancy Method for Template Coefficients

(4.5) Simulation Results 4.1

(4.6) A New Learning Algorithm for CNNs

(4.7) Feature selection of binary patterns

(4.8) Simulation Results 4.2

(4.9) Summary and Conclusions

(4.10) References of Chapter 4

Chapter 5

Conclusions

(5.1) Introduction

(5.2) Summary

(5.3) Original Contributions

(5.4) Recommendation for further research
<table>
<thead>
<tr>
<th>Symbols &amp; Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADALINE</td>
</tr>
<tr>
<td>AI</td>
</tr>
<tr>
<td>ANN</td>
</tr>
<tr>
<td>APR</td>
</tr>
<tr>
<td>A(i,j; k,l)</td>
</tr>
<tr>
<td>BP</td>
</tr>
<tr>
<td>BPTT</td>
</tr>
<tr>
<td>BNN</td>
</tr>
<tr>
<td>BPR</td>
</tr>
<tr>
<td>BI</td>
</tr>
<tr>
<td>B(i,j; k,l)</td>
</tr>
<tr>
<td>CNN</td>
</tr>
<tr>
<td>CI</td>
</tr>
<tr>
<td>CPS</td>
</tr>
<tr>
<td>CUPS</td>
</tr>
<tr>
<td>CA</td>
</tr>
<tr>
<td>CCD</td>
</tr>
<tr>
<td>CT-CNNs</td>
</tr>
<tr>
<td>CMOS</td>
</tr>
<tr>
<td>DT-CNNs</td>
</tr>
<tr>
<td>DTI</td>
</tr>
<tr>
<td>e_j</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>FLOPS</td>
</tr>
<tr>
<td>HD</td>
</tr>
<tr>
<td>IC</td>
</tr>
<tr>
<td>IPS</td>
</tr>
<tr>
<td>I_j</td>
</tr>
<tr>
<td>LMS</td>
</tr>
<tr>
<td>ML-CNNs</td>
</tr>
<tr>
<td>MNOS</td>
</tr>
<tr>
<td>MOS</td>
</tr>
<tr>
<td>MADALIN</td>
</tr>
<tr>
<td>MIMD</td>
</tr>
<tr>
<td>NN</td>
</tr>
<tr>
<td>NCTT</td>
</tr>
<tr>
<td>Abbreviation</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>NTL</td>
</tr>
<tr>
<td>Net</td>
</tr>
<tr>
<td>PR</td>
</tr>
<tr>
<td>RISC</td>
</tr>
<tr>
<td>SIMD</td>
</tr>
<tr>
<td>V\textsubscript{xij}</td>
</tr>
<tr>
<td>V\textsubscript{uij}</td>
</tr>
<tr>
<td>VLSI</td>
</tr>
</tbody>
</table>
Chapter 1

Neural Networks

1.1 Introduction

Over the past few years researchers from many diverse fields have joined forces to develop a new computational paradigm to deal effectively with the massive volume of information available to us today. This paradigm is variously called *Neural Networks*, *Neural Computing* [1-1], *Artificial Neural Networks* (ANN) [1-2], *Computational Neural-like Network* (CNLN) [1-3], *Neurocomputers* [1-4], *Connectionist Model* [1-5, 1-6], *Parallel Distributed Processing* [1-7], or *Neuromorphic System* [1-8]. It represents a core of techniques which draws its inspiration from the biological brain that has outperformed existing software and human experts in applications requiring pattern recognition and prediction. Considering the highly complex nature of the human brain, to understand completely the way it functions and consequently to build a machine with computational power in close approximation to the efficiency of the human brain, lies a long way in future. However, this latter statement does not imply that there are no partial solutions at each evolutionary stage of this challenge, and it must not be regarded as pessimism about the application of neural networks at each of these stages.

In this chapter, first some of the important questions such as: what are neural networks?, where are they effective? and what is their present situation and future from the technology transfer point of view?, will be answered. Next, a review of the technologies that are used to implementation ANNs models are presented. These will form a framework for developments undertaken by the author in the design of a class of neural network called Cellular Neural Networks (CNNs).

1.2 What are Neural Networks?

In part, because of rapid growth in the field of neural networks, there are differences of opinion and even confusion about the best way to categorize and describe neural networks. Robert Marks [1-9] has clustered neural networks into an area called *Computational Intelligence* which consists of a number of disciplines, including neural networks, certain fuzzy systems and all sub-areas of evolutionary computation. The detailed description of this term by Bezdek [1-3] can be a very useful example. Table
1.1 shows the definition of ABC’s (i.e. A = Artificial, B = Biological, C = Computational) that Bezdek uses to categorize different levels of intelligence. Figure 1.1 shows Bezdek’s view on the relationship between these ABCs and Neural Networks (NN), Pattern Recognition (PR), and Intelligence (I), where the symbol ( —► ) in the figure means a “proper subset of”. In this view, Computationally Neural-Like Network (CNLN) represents any computational model that draws its inspiration from biology and includes (among others): feed forward classifier networks, self-organizing feature maps, learning vector quantization, neocognitrons, adaptive resonance theories, genetic algorithms, Hebbian, Hopfield and counter propagation networks, evolutionary computing that are used for pattern recognition (feature analysis, clustering and classifier design). Bezdek considers Computational Pattern Recognition (CPR) as the “search for structure in numerical data” which includes deterministic, fuzzy and statistical PR models.

<table>
<thead>
<tr>
<th>Table 1.1. The definition of Bezdek’s ABCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNN</td>
</tr>
<tr>
<td>ANN</td>
</tr>
<tr>
<td>CNN</td>
</tr>
<tr>
<td>BPR</td>
</tr>
<tr>
<td>APR</td>
</tr>
<tr>
<td>CPR</td>
</tr>
<tr>
<td>BI</td>
</tr>
<tr>
<td>AI</td>
</tr>
<tr>
<td>CI</td>
</tr>
</tbody>
</table>

The core of Fig. 1.1 is the distinction between Artificial and Computational to prevent “confusion”, “misunderstanding”, “misrepresentation” and “misuse” of NN models in pattern recognition. At the highest level of Fig. 1.1, in the pattern recognition (PR)
sense, are Biological Neural Networks (BNNs). Bezdek has accepted the following definition from Defence Advanced Research Projects Agency (DARPA) as an adequate definition for the CNLN (but not for his idea of the ANN!): “a computational neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths and the processing performed at computing elements or nodes”.

![Complexity Level Diagram](image)

Fig. 1.1 Commuting through the ABCs

To classify a CNLN as a ANN, the following condition is proposed by Bezdek: “enough knowledge tidbits must be somehow added to the CNLN to endow it with rudimentary associative memory (an ability to link low-level data processing outputs with non-numerical rules, facts and constraints in order to increase the system’s understanding of its environment)”. To justify the hierarchical structure of his classification, Bezdek uses the following four “properties” of PR algorithms: P1) adaptivity, P2) fault tolerance, P3) speed during operation and P4) error rate optimality.

Property (P1)- adaptivity. It is assumed that the human brain adapts to changes in its environment by two “on the fly” mechanisms: 1) the synaptic weights vary over time, 2) activation and deactivation of (sets of) nodes (network reconfiguration). Bezdek offers the following definition for the word adaptive, from the standpoint of BNN: “the ability to adjust local parameters and global configurations to accommodate changes in environment without interruption of current services”. Based on the above definition, Bezdek makes the following statement: “if a CNLN (or other) algorithm is able to alter its parameters, and reconfigure its structure “on the fly”- that is, without interruption of on-line service, and can also assign itself new tasks when the demand exists, I would be happy to call that algorithm computationally adaptive. CNLNs are not more computationally adaptive than, say, Newton’s method. I don’t think any CPR algorithm, CNLN or otherwise, is adaptive in the BNN sense”.

Property (P2)- Fault Tolerance. The openness of the BNN to noisy, fuzzy, or even spurious data, information, rules and constraints is well known. Bezdek accepts that parallel and distributed computing in the style of the CNLN affords an architecture
that may provide tolerance to some or all of these data anomalies. Bezdek believes that there is lack of a vital item to specify precisely what computational fault tolerance means. He wants a tool to measure this property, and use it to make comparisons with other methods.

Property (P3)- *Computational Speed.* Bezdek accepts that CNLNs will eventually be faster than most conventional algorithms in terms of raw computational speed (like their BNN role models)- because they offer the possibility for distributed and parallel processing.

Property (P4)- *Error Rate Optimality.* Bezdek believes that there is need to standardize treatment of error rates (similar to Bayes error rate) as performance index of choice in classifier design and test.

Finally, Bezdek concluded his discussion by offering the following two definitions:

i) “A system is computationally intelligent when it deals only with numerical (low-level) data, has a pattern recognition component and does not use knowledge in AI sense, and additionally, when it (begins to) exhibits (1) computational adaptivity, (2) computational fault tolerance, (3) speed approaching human-like turnaround, and (4) error rates that approximate human performance”.

ii) “An artificially intelligent (AI) system is a computational system whose added value comes from incorporating knowledge (tidbits) in a non-numerical way”.

Although I agree with much of the Bezdek’s argument, in general, I find his remark about adaptivity very rigid. He considers an algorithm to be adaptive if it “alters its parameters, and re-configure its structure on the fly - that is, without interruption of online service, and can also assign itself new tasks when the demand exists”. Recently, there was a TV-documentary about human behaviour in situations that they have never experienced, such as a plane crash and a ship sinking at sea. Some of the cases in the program were evidence that the BNN also lacks such rigid “on-the fly” reaction in such untrained situations. For example, in the case of the ferry disaster a group of passengers were observed who were unable to react to save their lives. Therefore, I believe that we should consider an adaptive system in the evolutionary sense. I can still call an algorithm adaptive despite its limited capability, in comparison to the BNN, in terms of “on-the fly” alteration of its parameters (i.e., on-line learning capability). To distinguish the level of adaptivity of a computational network and of the BNN we can use a ranking methodology, say, an adaptive algorithm- generation 1, or 2 and higher order as we move toward adaptivity in BNN sense.
However, from now on I will use the term neural networks with computationally neural-like networks in mind for which the following definition is provided: *The neural networks are new information processing systems which are designed based on our present knowledge of the biological brain at its architectural and functional levels. A neural network is composed of a large number of simple processing elements (PEs), called synthetic neurons, nodes, or simply cells which are connected to each other in different configurations through variable coefficients called weights or synaptic weights. All the nodes perform local processing. The desired computational task is carried out by co-operation of all units which results in what is called parallel distributed processing (PDP).*

1.3 Attractive Features of Neural Networks

It is known that biological neural events happen in the millisecond range, whereas events in silicon chips happen (generally) in the nanosecond range. The immediate questions are: i) what makes the brain of a small child much smarter than super computers in performing tasks such as speech, robotics, pattern recognition and vision?, ii) How does the biological brain achieve the energetic efficiency of approximately $10^{-16}$ joules (J) per operation per second, whereas the corresponding value for the best computers in use today is about $10^{-6}$ joules (J) per operation per second [1-10]? It is believed that a biological neural network derives such computing power through: i) massively parallel distributed structure, ii) their ability to *learn*, and therefore to generalize. In other words, it is able to analyse data and extract knowledge from individual data and finally to generalize from single observations to the underlying principles and the structure of the information.

Some of the important benefits and properties of neural networks are given in [1-11]-[1-13] and are discussed briefly below.

- **Adaptivity.**

Neural networks have a built-in capability to adapt their synaptic weights to changes in their surrounding environment.
• Generalization

Neural networks are able to abstract, or to respond appropriately to input patterns not encountered during their training (learning) phase.

1.4 Applications of Neural Networks

The number of tasks in which neural networks can be used are quite varied. However, we can categorize these tasks into four general classes as follows:

• Classification. Conventional multivariate data analysis procedures could be modelled as a mapping

\[ f: A \rightarrow B \]

where A and B are finite sets. If B is a set of discrete categories, and A is a set of descriptive m-dimensional vector characterizing n points, then the mapping \( f \) can considered as a clustering and choosing an appropriate \( f \) is the domain of the cluster analysis. Cases where precise knowledge of B is available are called “supervised classification”. This has two distinguishing phases: i) learning, where the mapping \( f \) is determined using the training set, \( B' \in B \), and ii) the generalization phase. Discriminant analysis methods fall in this category.

Neural networks can be presented as nonlinear systems as shown in Figure 1.2. Mathematically, a network can be defined as a transformation that uniquely maps an input pattern into an output pattern. Then a neural network as a classifier determines a nonlinear \( f \), where \( f: A \rightarrow B \), A is a set of n m-dimensional input vectors, B is a set of n p-dimensional output vectors that some of its members are known a priori. B could be a label set, associated with a set of discrete classes. In this case, the training sets are the associated pairs of members from A and B which are used to determine \( f \).

• Association. In auto-association a neural network is able to reconstruct the original stored pattern when it is presented with a corrupted or incomplete version of the stored
pattern. Hetero-association is different from association in that the network makes a one-to-one association between members of two sets of patterns.

- **Transformation or mapping.** Neural networks are often used to map a multivariate space into another multivariate space of the same or lower dimension. In the dimensionality reduction technique, \( B \) is a space containing \( n \) points that has lower dimensionality than \( A \). For example, in the Kohonen network, \( B \) is a regular 2-dimensional grid of points, and \( f \) is an unsupervised mapping.

- **Modelling.** Typically the task of modelling requires an approximation to the continuous input-output mapping performed by the system in a robust manner. This is illustrated in Fig. 1.3. This task can be treated as a learning task where the model is taught such that the output error, \( e(t) \), is sufficiently close to zero. The internal representation of these models is generally fixed, except for a series of adjustable parameters, called weights, which adapt the continuous function mapping of the model. Weight identification, known as learning, is achieved by optimizing the weights with respect to some error criteria across a set of training data (i.e. experimental (input) and calculated (output) data). The usefulness of neural networks for modelling stems from the fact that neural networks: i) are simple mathematical structures that gather knowledge by learning from example, ii) can handle highly non-linear problems, making them vastly superior to classical linear approaches, and iii) are able to adapt on line.

![Fig. 1.3 Illustration of the modelling task.](image)

### 1.5 The Future of Neural Networks

The pioneering work of McCulloch and Pitts [1-14] started the modern era of neural networks. However, interest in neural networks continued until 1970 when a “dark age” of neural networks began and many researchers deserted the field. Rightly or wrongly, many factors have taken responsible for the creation of this period. Cowan [1-15] gives three reasons: i) lack of personal computers and work stations for experimentation, ii)
financial and psychological, and iii) the analogy between neural networks and lattice spins was premature.

The modern renaissance in neural network technology started in the 1980s. The 1982 paper by Hopfield and the book *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, by Rumelhart and McCleland (1986) [1-7] are widely credited with leading the resurgence of interest in neural networks in the 1980s. Having said that, it is important to answer the following questions: Has this modern renaissance in neural network research been able to establish it as a viable technology to overcome the inherent fear of a new technology? Is neural network technology now taken much more seriously than before? And consequently, is neural network research peaked?

To answer above questions we start with professor Mark's [1-9] remarks in response to the last question:

“Goodness no! on the contrary neural networks have spilled into a number of fields, including fuzzy systems and evolutionary computation. Lee Giles, a fellow “neural smith” did a data base research and found there were over four times as many neural network papers published in non-neural networks IEEE Transactions than are published in the IEEE Transactions on Neural Networks. This shows neural networks are flourishing in application and implementation...In 1994, the last year where nearly complete data is available, there were over 8,000 publications on neural networks and over 11,000 in computational intelligence. There were 250 patents in 1994 involving either neural networks or fuzzy systems. This compares to about 60 for AI. Patents reflect the application and implementation activity of a technology...Neural networks, in particular, continue to have a significant impact on engineering. In the 1994 Journal of Citation Reports, of the 138 electrical engineering journals ranked, the IEEE Transaction on Neural Networks scored number five in terms of impact.”

It has been accepted that neural technology is a developing technology and reached a sufficient maturity to offer realisable commercial benefits. In the UK, there are around 30 commercial suppliers of neural network services, and about 50 products covering software development tools, neural network hardware and application packages [1-16]. The neural computing database of the UK government’s Department of Trade and Industry (DTI) includes details of 9,000 individuals, representing about 6,000 organisations, who have shown an interest in neural networks [1-16]. Based on such factors, the DTI launched a 3 year, £5.75 million Neural Computing Technology Transfer Programme (NCTT) to promote awareness and exploitation of neural
computing, in February 1993. This nation-wide campaign is called “Neural Computing-Learning Solutions”.

Ryman-Tubb [1-17], founder and chief executive of neural technologies limited (NTL) pointed out that : “According to US market research company Frost & Sullivan, the world neural market is expected to expand at a compound rate of 46 percent a year up to 1998 [1-18]”.

The DTI conducted a survey into users of neural computing technology. It confirms the high level of interest in the technology with nearly 50 per cent of the users surveyed having more than on neural computing application [1-1]. The survey also shows that the role of neural networks has changed. Rather than being treated as a “be all and end all” it is used as part of a hybrid system. A total of 59 per cent of users said that this was the case and that neural systems are combined with other complementary technologies [1-1].

1.6 Taxonomy of Neural Networks

The three principle entities that characterise a neural network are as follows:

i) The characteristics of the individual synthetic neuron.

ii) The network topology, or interconnection of the nodes.

iii) The strategy for pattern learning or training

There are two main methods in classifying neural networks that are adopted by many authors in the fields. One such method to categorize NNs is based on the manner that the recall of information is performed, i.e., the Feedforward mode and the Recurrent mode. The second approach for classification of neural networks is based on their learning mode, i.e., Supervised learning and Unsupervised learning.

However, there is also a third approach in classifying neural networks which uses the biological evidence as the categorization criterion [1-19]:

i) neural networks that are based on mimicking biological neural systems (e.g., the silicon retina [1-20].

ii) neural networks operating on a somewhat higher level, such as edge detection. These approaches also use the present understanding of biological system (e.g., cellular neural networks [1-21]).
iii) neural networks inspired by biological evidence to a lesser degree (e.g., Backpropagation Learning algorithm [1-22]).

1.6.1 Synthetic Neuron Model

A specific artificial neural structure is shown in Fig. 1.4. It consist of : i) a set of variables called weights, ii) a summation unit which sums the weighted inputs where the resulting sum represents the state of activation whose value is shown by the notation net, and iii) an activation function fi that maps neuron activation into an output signal.

There are variety of activation functions. The simplest example is that of a linear unit, also called identity function. In this case the output of the neuron can be defined by

\[ y_i = f(\text{net}_i) \]  

(1.1)

where

\[ \text{net}_i = \sum_{j=1}^{n} w_{ij} x_j \]  

(1.2)

Typical functional structures that are often used are continues symmetrical bipolar and binary symmetrical functions:

\[ y_i = f(\text{net}_i) = \frac{1 - e^{-2\lambda \text{net}_i}}{1 + e^{-2\lambda \text{net}_i}} \]  

(1.3.a)

\[ y_i = \text{Sgn}(\text{net}_i) = \begin{cases} +1 & \text{net}_i \geq 0 \\ -1 & \text{net}_i \leq 0 \end{cases} \]  

(1.3.b)

respectively. Notice that if \( \lambda \to \infty \) the function becomes the Sgn (\text{net}_i). The unipolar (unsymmetrical) version of these functions can be represented by

\[ y_i = f(\text{net}_i) = \begin{cases} 1 & \text{net}_i \geq 0 \\ 0 & \text{net}_i \leq 0 \end{cases} \]  

(1.4)
1.6.2 Topology of Neural Networks

Neural networks can be divided into three general classes based on their architecture as follows:

*Feedforward Networks.* In feedforward neural networks neurons are organised in a feedforward manner in the form of layers such as the multi-layer feedforward network.

*Recurrent Networks (feedback networks).* The recurrent (also called feedback) neural networks can be obtained by connecting the neuron outputs to their inputs (e.g., Hopfield Network [1-23]).

*Cellular Structures.* In this class of networks the output neurons are arranged as one dimensional, two dimensional, or n-dimensional array with a set of source nodes in which each node is connected to every neuron in the array (e.g., Cellular Neural Networks and Kohonen network [1-24]).

1.7 Implementation of Neural Networks

The driving force behind the development of neural network models is the use of biologically inspired elements in their design to tackle problems such as optimization and adaptive learning to solve cognition tasks (e.g. image and speech recognition) in a real-time fashion. The degree of capturing such attribute is dependent on the technology used to execute these models. Some of the main conditions which must be met by a chosen technology, in addition to the constraints of power and space, include learning capability, adaptibility, massive interconnections and storage medium. The
design can be based on analog, digital or hybrid circuitry and these will now be discussed.

### 1.7.1 Hardware Neural Networks Design Concepts

For appropriate research into neural networks architectures that implement NN models much faster than is currently available are needed. Figure 1.5 summarizes some of the important applications and their range of computational requirements. The performance obtainable with some of the commercially available simulators in terms of weights and interconnections per second are also provided [1-25]. For acceptable performance of neural networks large numbers of neuron ($>10^3$ neurons, and $>10^6$ synapses) are required. Simulating such large networks thus require specialised machines that executes neural network models in real time (>1G CPs).

Neural networks are currently realised in a number of ways. Conventional digital computers are used to simulate neural systems. Such solutions are regarded as possessing a number of distinct disadvantages, a) low speed (e.g. computational power between 25,000 to 250,000 Connection Per Second (CPS) expected [1-26]), and b) high cost. For best results in real time applications, speed becomes the driving force to consider other solutions which are titled ‘Neurocomputers’ [1-27]. A neurocomputer is defined as: “essentially a parallel array of interconnected processors that operate concurrently. Each processing unit is a primitive (that is, it consists of an analog neuron or simple reduced instruction-set computer known as a RISC) and can contain a small amount of local memory”.

The realisation of these neurocomputers, which are optimized for the computation of neural network systems, follows one of the two following approaches:

**General-purpose neurocomputers.** These are generalized and programmable neurocomputers that are used for emulating a wide spectrum of neural network models.

**Special-purpose neurocomputers.** These are specialized neural network hardware implementations which are dedicated to a specific neural network model. Therefore, they potentially have higher computational power than generalized neurocomputers.

Although the first approach offers flexibility in the modelling (these software controlled networks support several neural network models), and shortening the design time due to availability of devices, they remain at least an order of magnitude slower than special-purpose neurocomputers in the emulation of neural networks.
1.7.2 Specification of Neurocomputers

It is important that to make clear those measures which decide the most cost-effective medium for the implementation of a neural network. According to Hecht-Nielsen these measures are [1-28]: 1) Speed, 2) Cost, 3) Accuracy, 4) Generality, which is often called flexibility, 5) Software Interface Provision, 7) Configuration Provisions.

Speed. In [1-28] the criteria for rating the computational power of neural networks was given as:

"speed is rated by how quickly a specified network of a specified size can be moved forward in time by one discrete time step. In other words, speed is determined by how quickly the network processing elements can all be updated by one time step, or by the time it takes to allow the network to undergo one complete cycle (e.g., as in backpropagation, where each cycle consists of a forward pass and a backward pass). Commercial neurocomputers are typically benchmarked using a large backpropagation network of a stated number of rows and processing elements. To normalize the speed measurement, the total number of connections that can be updated in a second is often used as the speed measurement".

The above measure of speed is often called Connection Update Per Second (CUPS). For many architecture which do not support learning BP algorithm another benchmark called Connection Per Second (CPS) is used. Although "CPS" is used by many authors
in the field as a performance measure, it is subject to wide interpretation (e.g., similar to the Instruction Per Second (MIPS), when the meaning of the metric is not defined [1-29].

*Information transfer.* While neural networks operate at very high speed, there is a bottleneck for the input and output signals that have to be brought to their destination points. In some cases, the input signal bottleneck can be circumvented by using optical inputs through opto-electronic devices such as photo-transistors.

However, comparison of the required computational power for each specific application indicates that a trade-off may be necessary between three main features: processing time, network size (in particular memory size) and the input-output bandwidth.

### 1.7.3 General-Purpose Neurocomputers for NNs Simulation

General-purpose neurocomputers can be categorized into the following groups: i) conventional sequential microprocessor, ii) coprocessor based hardware accelerators, and iii) *single instruction multiple data* (SIMD) and *multiple instruction multiple data* (MIMD) style neurocomputers. An overview of some of the well known parallel processor arrays and coprocessor-style neurocomputers is presented for the comparison purpose.

Neurocomputing co-processors are available from Hecht-Nielsen neurocomputers called ANZA plus and ANZA plus/VME. They are compatible with the PC-AT and the VME bus respectively [1-30]. The boards are based on the MC68020 microprocessor plus the Motorola MC68881 floating point processor. Both have 1,800,000 CPS during the learning and 10,000,000 CPS in the recall phase.

Another acceleration board approach uses the Intel i860 processor and was introduced by Hecht-Nielsen computers [1-31]. It can process about 7000,000 CUPS in the case of BP learning.

However, the computational power of these accelerator based systems is limited (e.g. clock frequency of up to 60 MHz with 32 bit buses) due to factors such as chip size and physical parameters. To overcome this speed bottleneck it becomes necessary to insert parallel components into the neural system's hardware design. Some examples of such solutions are given below.

A parallel processor based on Digital Parallel Processor (DSP) introduced by Fujitsu Laboratories [1-32]. Their system is based on the Texas Instruments floating point
digital signal processor TMS320C30. It is expected that a system with up to 256 processing elements (PEs) can operate with up to 567 million CUPS in a large network.

A parallel processor based on Transputers was reported in [1-33]. Speeds up to 250,000 CUPS for the BP algorithm with 74,996 connections reported, using four Inmos T800 transputers. One advantage of transputers over standard DSPs, from the neural system implementation point of view, is that the former has more on-chip memory, an external memory interface and communication channels.

A system based on the RISC processor was introduced in [1-34]. It is a 2-D array of mesh connected RISC processors (e.g., the Intel 80860 with 12 Kbyte of on-chip memory, a 32/64 bit floating point multiplier, an adder and a pipelined processing architecture). It was expected that a neural network with 368 PEs and 5120 training vectors would have computational power >1000 MCUPS based on the RISC array of size 128 that runs at a clock frequency of 50 MHz.

Mas Par MP-1s are massive data-parallel computers whose architecture is based on MP-1 SIMD array chips. Computational power up to 306,000 CUPS for the BP algorithm (e.g., 900 input units, 29 hidden units and 17 output units, applied to a speech recognition problem) was reported [1-35].

The computation rate of the biological brain has been estimated to be $10^{18}$-$10^{20}$ floating-point operations per second (FLOPS). Therefore, for certain complex applications such as speech recognition and optical character recognition, a level of performance is required that is not attainable with existing parallel processors, certainly within the cost limitations of the proposed application.

1.7.4 Special-purpose Neurocomputers

According to Boser [1-36] special-purpose VLSI (very large-scale integration) processors enable us to overcome neural network implementation problems. Neural networks are well matched to integrated circuit technology because: i) they have regular structures, and ii) they involve a number of well-defined arithmetic operations. The high density of modern VLSI technologies enable us implement a large number of identical, concurrently operating processors on one chip, thus exploiting the inherent parallelism of neural networks.

However there are a number of trade-offs in deciding the electronic technology with which to implement these systems.
1.7.5 Analog NN Design Vs. Digital NN Design Concepts

We start with the argument of Vittoz [1-37] against the commonly accepted view of the future role of analog VLSI (this states that the role of analog in future VLSI circuits and systems will be confined to that of an interface, a very thin “analog shell” between the fully digital substance of the growing signal processing “egg”):

According to this view, the task of analog designers will be essentially concentrated on developing converters to translate, as early as possible, all information into numbers, with an increasing demand on precision, and to bring the results of the all-digital computation back to the real world. This view is certainly correct for the implementation of all systems aiming at the precise restitution of information, like audio or video storage, communication and reproduction. Together with very fast but plain computation on numbers, these are the tasks at which the whole electronics industry has been most successful.

However, Vittoz pointed out that “the very precise computation sequences of numbers is certainly not what is needed to build systems intended for the quite different category of tasks corresponding to the perception of a continuously changing environment”.

To provide a clearer picture of the advantages and disadvantages of these two technologies we look at their capabilities in different circuit perspectives.

• Signal level. Analog circuit techniques provide a better understanding of the true analog nature of biological neural networks. As pointed out by Mead [1-8]: “our struggles with the digital computer have taught us much about how neural computation is not done, unfortunately, they have taught us relatively little about how it is done”. Part of the reason for this failure is that a large proportion of neural computation is done in an analog rather than in a digital manner. Again in favour of analog signal processing from different perspective we adopt a long statement from [1-38]:

“Another advantage of analog systems is that time can be easily represented as a continuous variable through a time to potential or current transformation, therefore avoiding the clocks and counters required in digital systems to take track of time. As a result, analog neural networks can provide not only a mechanism for summation and scaling of incoming signals, but for temporal integration as well, in a fashion similar to biological networks”.

One of the practical limitations on the size of networks which can be built into a single chip is the provision of communication channels for signals from within the chip to the
outside world and vice versa. Circuits employing analog I/O signals are at an advantage over digital implementations because each input and output may be allocated a single pin. In digital implementations in which there may be 8 lines required to define an input or output level, it is frequently necessary to multiplex the data on pins with corresponding impact on the overall throughput of the system [1-39].

- **Precision.** One of the central issues raised by supporters of digital implementation of neural networks is that analog circuits require high precision resistors and capacitors which are easily affected by noise’ [1-40].

As pointed out by Vittoz [1-37], the experimental evidence shows that for perception what we need are: i) a massively parallel collective processing of a large number of signals that are continuous in time and in amplitude, and ii) nonlinear processing is the rule rather than the exception, not the precision.

- **Activation and Transfer Function.** Implementation of the transfer function using analog circuit techniques is a very easy task using the natural non-linearities of analog components. A large class of non-linear functions can be easily implemented.

In digital techniques, the sigmoid function can be approximated by linear segments, with its derivative stored in a digital memory as lookup tables. But, digital implementation of sigmoid nonlinearities are handicapped by the large memory size, slow memory access time and degraded performance [1-41].

- **Implementation of Synaptic Weights.** Forming the weighted sum of input signals is the heaviest task from both computational load and silicon area point of view. Therefore the hardware to do this task must be selected very carefully. In digital VLSI techniques the multiplication and summation is done by digital multipliers and adders. For example, the cell body of the digital neuro-chip of Yasunaga [1-42], based on gate-array technology, is composed of two adders, a few registers and shift registers.

In the analog design style the weighted sum of inputs can be achieved by summing analog currents or charge packets. Therefore, the analog implementation of these elements using analog circuit techniques is superior over their digital counterparts in terms of silicon area, speed and power consumption.

- **Weight Storage Media.** While the technology of digital memories is very mature and well understood, the storage analog weights is one of the most important problem in the analog implementation of neural networks that has yet to be solved. Much work on analog NN implementations has focused on the concept of storing analog weights as a charge package on a capacitor [1-43, 1-44] as shown in Fig.1.6a.
Vittoz [1-45] pointed out that “since the synaptic weighting may be carried out in various ways through the gates of MOSFET transistors, the most natural method is to store the weight as voltage across a capacitor”. The most natural way to do this is to store the weight across the gate capacitance [1-46] which can be an intrinsic or an intentionally added one.

The storage time limitation of this method is due to the leakage current which may discharge such a capacitor within a few milliseconds. For appropriate analog storage this loss of charge should not be larger than 1%. In [1-47] a dynamic capacitive storage was presented which is compact, programmable and simple. However, it has a shortcoming due to an increase in system complexity which stem from the off-chip refresh circuitry that is employed.

In [1-48, 1-49] Metal Nitride Oxide Semiconductor (MNOS) device is used as an analog memory. A typical MNOS structure is shown in Fig.1.6b. To store electrons at the nitride-oxide layer, a large positive voltage pulse is applied to the gate. This will cause electrons to be driven into the gate structure from the silicon substrate by some tunneling mechanism such as Fowler-Nordheim tunneling. Thus, the threshold voltage of the device increases as a result. In order to reduce the threshold voltage, a negative high voltage pulse can be applied that causes electrons to be driven out from nitride to the substrate. Such a technique offers programmability, at the expense of the requirement for a non-standard process.

In [1-50, 1-51] Hydrogenated amorphous (a-Si:H) was used as an analog storage media which is compatible with conventional silicon device technology. The structure of a-Si:H is shown in Fig. 1.6c. The device consists of a thin (typically 100nm) layer of p-type amorphous silicon sandwiched between two metal contacting electrodes. The device has a resistance of $\sim10^9$ ohms after fabrication. For programming purpose a pulse (up to 12V for approximately 300 ns) is then used. It is understood that this pulse causes local heating which results in diffusion of the top metal electrode into the a-Si layer. The type of the top electrode metal determines predominantly electrical properties of the device. The main limitation of the approach is the requirement to download the synaptic resistance values.

Finally, the most common and promising technique for the future of analog storage media is the use of floating gate transistors [1-52]. The top view of such a device is shown in Fig. 1.6d. The operation of this device relies on electron tunneling through a thin oxide. If a sufficiently high positive voltage is applied to the poly-2 (also called the control gate) and the drain is grounded, electrons will tunnel from the floating gate
to the drain and transistor threshold voltage is decreased. On the other hand, if a negative pulse is applied to the control gate while the source and drain terminals are grounded, electrons will be added to the floating gate so that the threshold voltage is increased. Floating gate memories offer very long retention times of up one year, but they suffer from the writing limitation which limits the on-line reprogramming. To solve this problem many authors use dynamic memories such capacitors as short term memory which allows fast updating of synaptic weights, while non-volatile memory such as floating gate memory as long term memory that allows reliable long-term storage and low power dissipation once learning is complete.

**Interconnection** It is a well known fact that analog signals are very sensitive to noise and cross talk. The collective processing of data in massively parallel systems requires a dense network of communication between neurons, that is not naturally available in intrinsically two dimensional VLSI implementations. Furthermore, very large perception systems will have to be split into several VLSI chips, on which successive layers of processing may be implemented, with the result of a massive flow of signals from chip to chip [1-53]. These problems of inter-chip and intra-chip communication can be solved by a number of complementary approaches, i) exchanging information on a single node corresponding to a single wire across the
whole chip, ii) to use digital signals to carry information, and iii) to employ architectures such as cellular networks to avoid long distance communication altogether.

1.7.6 Hybrid Analog/Digital NN Design Concepts

It is known that analog techniques offer unquestionable advantages over their digital counterpart in terms of silicon area, speed and power consumption. However, analog signals are susceptible to interference when intra-chip communication is required. In contrast digital signals are known to be robust in this sense. This is why some neural network chips use a hybrid analog-digital design approach to combine the merits of both techniques. Another reason for the use of hybrid analog/digital realization of NNs is the lack of readily available, non-volatile, and flexible analog weight storage as previously discussed.

Pulse stream arithmetic represents a novel hybrid approach that employs the attributes of digital and analog electronic to realize large scale NNs. Devised by Murray et al [1-49], the pulse stream encoding technique performs analog computation under digital control, where digital signals are used to carry information encoded along the time axis, with no analog voltage present in the signal.

Hybrid neural network circuits often use analog processing elements while synaptic weights are stored in digital form. Therefore the use of various D/A converters is necessary. Such circuits are generally costly in terms of additional chip area and more importantly due to an increase in the number of required pins.

1.7.7 Optoelectronic Implementations of Neural Networks

The potential of optical computing has been well recognised since early days of Fourier optics. The most important advantage of optics lies in the optical system’s ability to provide the massive interconnections between neurons required in most neural network models. In electrical interconnection schemes, as the wiring density increases, wires become thinner and spacing smaller. This arrangement results in slower transmission, large signal skew, and increasing noise and crosstalk problems [1-54]. In electrical systems, signals flow by means of the charging and discharging of nodes. These delays, plus the device and interconnect transit time limit the clock speed of the circuit. Other problems include constraints on design options due to the need for impedance matching
and grounding due to high fan-out requirements. To compensate for these problems power consumption must be increased to drive heavier interconnection loads, producing further complications of size and heat removal [1-54].

Optical signal propagation, on the other hand, is free from capacitance. Light beam can propagate through free space and pass through one another without interacting. Optical sources have large bandwidths which can approach one giga hertz for laser diodes. Therefore, the large bandwidth of optical connections and their immunity to crosstalk makes them very good candidates for the implementation of extremely high 2-D spatial parallelism and 3-D free-interaction interconnections.

Therefore, optical computing systems provide the features that are essential for the efficient implementation of neural networks: massive parallelism, high computational power, high interconnectivity, and hence large fan-in and fan-out.

In general there are two optical approaches to implement optical computing systems [1-55]. These are bulky free space optics and integrated optics. Free-space optics (e.g., dynamic holographic media such as photorefractive crystals) are more mature, more accessible, and provide many functions. In this approach coherent laser lights is often used as information carrier. Mirrors and masks and complex lens systems are used to direct the light beams.

A more promising approach in the near future, is the use of hybrid optoelectronic processing element. The advantage of this combined optoelectronic approach is that it exploits the strengths of both optics and electronics with optics providing the connections and electronics providing the nonlinear processing [1-56, 1-57]. Gallium Arsenide (GaAS) is an excellent material for this purpose, since it can be used to fabricate both fast electronic circuits and optical sources and detectors.

However, the exploitation of optoelectronic and optical technologies may lead to systems of extremely high performance, but compared with digital and analog VLSI technology, optoelectronic computing is much less mature and still a topic of basic research.

1.8 Summary and Conclusions

Neural networks are new efficient computational paradigms which draw their inspiration from the biological brain, and have outperformed existing software and human experts in applications requiring pattern recognition and prediction.
Evidence shows that commercial pressures are hastening their adaptation in applications from health care to the military. The applications profile of DTI directory contains details on about 40 applications of neural computing, covering a range of business sectors including banking, finance and insurance, telecommunications, manufacturing, health care, property and construction, energy, retail, tourism, agriculture, transport, and the media, plus applications in central and local government [1-16]. The major international companies such as IBM (neural network is part of their current Anti Virus software), Siemens (Currently has more than 20 neural network applications running in a dozen of their production sites). I would like to share with readers this remark from Ryman-Tubb [1-17]: “The neural networks are no longer considered uniquely with eccentric innovators and research departments but have moved to mainstream development. Corporate have started to realize that they need to invest not just financially but also in people, time and organisational systems in order to get immediate benefits”.

I would like to share with readers of this thesis the remark of Haykin [1-58] about future of the neural networks: “Needless to say, they are here to stay, and will continue to grow in theory, design, and application”.

The effectiveness of any neural network model and other computationally intelligent systems is strongly dependent on the technology that is chosen to realise these architectures. Hence a number of different technologies for realisation of neural network models were discussed, in terms of their limitations and advantages. The designs can be based on analog, digital or hybrid electronic circuitry.

Neural network implementations based on customised digital electronic circuitry all have a sequential component in their computation. Consequently, neural networks simulated by these technologies tend to be slower than special purpose built analog neural network implementations (since analogue circuits are asynchronous). To implement simple functions a large number of transistors is needed which suggest that moderate sized neural networks would occupy considerable silicon area (i.e., high cost). However, digital electronic circuitry offers: i) a high degree of precision and a wealth of established technology, ii) ease of programmability which allows the flexible evaluation of different types of neural networks and learning algorithms.

In contrast, special purpose analog electronic designs offer optimised performance in terms of speed, power consumption and silicon area at the expense of flexibility. This is confirmed by Ismail’s remark [1-59], as follows:
"Mead [1-8] has built a family of silicon retinas. Each silicon retina is a VLSI chip that is a square centimetre in area, weights about a gram, and consumes about a milliwatt of power. Between array of photo-transistors etched in silicon, dedicated circuits execute smoothing, contrast, and motion processing. The transistors within the chip operate in subthreshold (analog mode) region. Compared with a typical charge-coupled device (CCD) and standard digital image processor, the Mead chip is paragon of efficiency in performance, power consumption, and compactness. A special-purpose digital equivalent would be about the size of a standard washing machine. Unlike cameras that must time sample, typically at 60 frames per second, the analog retina works continuously without needing to sample until the information leaves the chip already pre-processed’’.

However, analog systems are difficult to programme since there are no non-volatile analog storage mediums that can be easily written and read.

The problem of realising high connectivity in neural network models, suggest that an optical approach may have advantages. However, the current realisations of neural networks based on the optical technology are limited by the fact that this technology is very much in its infancy.

1.9 Research Topics

It has been shown that for acceptable performance of neural networks a large number of neurons and synaptic weights are required (>10^3 neurons, and 10^6 synapses). For instance (in the case of a three layer network), to map a square of 256 pixels requires 65,536 input neurons. For a square of 512 pixels, over a quarter of million neurons are required. The number of neurons in the first hidden layer should be equal to two times the number of input neurons pulse one according to Kolmogorov’s theorem. For a fully connected 256 pixel square image, this would require about 131,000 neurons in the hidden layer. This would require approximately 34 trillion connections [1-60].

It has been proposed that Rent’s rule is a statement of the high dimensionality of VLSI circuits [1-61], which suggest that one can define a fractal dimension of information flow D for a circuit by

\[ P = 1 - \frac{1}{D} \]
which implies that it would be very difficult to build networks with $P$ at or near unity, since as the system increases in the size the resulting high dimensionality of information flow necessitates compressing the interconnections into the constant three dimensional volume of the system. However, there is no VLSI system that has yet been constructed with a Rent exponent greater than 0.75. Therefore the dimensionality of the neural network system must be compatible with the VLSI technology used.

One possible solution would be to increase the number of metalisation layers to facilitate the high dimensionality of information flow. This approach has a number of shortcomings as follows: i) parasitic capacitance, ii) alignment difficulties, iii) noise resulting from the overlap of several metalisation layers.

The original Cellular Neural Network (CNN) paradigm was first proposed by Chua and Yang in 1988 [1-21]. One of the most fundamental ingredients of the CNN paradigm is that the direct interaction between the signals on various grid points limited to a finite local neighbourhood, which is sometimes called the receptive field. Thus the nearest neighbour interactive property of CNNs make them more amenable to VLSI implementation.

The aim of this thesis is to investigate the efficient design of Cellular Neural Networks (CNNs), in terms of minimum number of template coefficients (i.e. synaptic weights) and interconnections.

- **General philosophy.**

The general design philosophy in this thesis is that an absolute minimum number of template coefficients and thereby interconnections are required throughout the whole CNN. This makes the design of programmable and general purpose CNN chips, that match the requirements of the existing VLSI technology, possible.

One of the major issues that has to be tackled when designing a CNN circuit is the realisation of the complete set of programmable template coefficients in an efficient way, in terms of silicon area. The most efficient way is to make some of the coefficients redundant when it is possible.
Outline of this thesis.

In Chapter 1, the opinions on the way to categorise and describe neural networks, fundamental properties, applications, the basic motivation of research and future of neural networks are described.

In order to capture the fine grained computational parallelism of neural networks, a wide range of neural networks implementations have been proposed which utilise either digital/and or analogue semiconductor or optical technologies. It is important to assess advantages and disadvantages of each technology, thus a comparison between various technologies which used to implement neural networks is made in Chapter 1.

In Chapter 2 a description of different aspects of a class of non-conventional neural network model, known as a Cellular Neural Network, is given. The investigation into the design of CNN was prompted by its architecture. The local interconnectivity makes a CNN a good candidate for implementation in a widely available microelectronic technology and silicon compilation. Also their analogue nature enables them to process information in real-time fashion.

In Chapter 3 two methods for virtual template expansion in VLSI implementation of large-neighbourhood CNNs are presented [1-62, 1-63]. The proposed methods are matched by the requirements of current microelectronic technologies.

A practical circuit for the implementation of the proposed non-linear template coefficient is given in Chapter 3.

In Chapter 4 the importance of applying the redundancy and the feature selection techniques for design of CNNs is described [1-64]. A new design method for CNNs using the Ho-Kashyap algorithm is described [1-65].

In chapter 5 finally a summary, conclusion and recommendations for future works are given.
1.10 References of the Chapter 1


Chapter 2

Cellular Neural Networks

2.1 Introduction

One of the main features of neural computational architectures is that the structure of the problem can be reflected in the structure of the network. For example, prior information and invariance can be built into the design of a neural network. Such a specialised structure is desirable for the following reasons [2-1]:

• biological visual and auditory networks are known to be very specialised.

• a neural network with a specialised structure has a much smaller number of free parameters available for adjustment than a fully connected network. Consequently, the specialised network requires a smaller training data set and learns much faster.

• the network throughput is accelerated.

• the cost of building of such network is reduced by virtue of its smaller size, compared to fully connected networks.

One example of such a specialised architecture is the Cellular Neural Network [2-2]. They are large scale analog systems whose architectures are inspired from Cellular Automata (CA) [2-3] and the Hopfield neural network model [2-4]. Their unique features are [2-5]:

• their local connectivity makes them an ideal candidate for analog VLSI implementation.

• their continuous-time operation allows the computational speed to depend only on the time-constant of the underlying dynamic system.

The basic model of the cellular neural network (CNN) was initially introduced in [2-2, 2-6] and belongs to the category of analog recurrent networks. Their basic building block is called a “cell” or neuron which is an analog dynamical circuit that includes a linear resistor, a linear capacitor, linear and nonlinear dependent sources and independent sources. Intercell connections of a CNN are constrained to a pre-defined neighbourhood cells, similar to the connectivity scheme in Cellular Automata. The synaptic weights (also called cloning templates) of a CNN are uniform over the entire array.
Since then a number of generalizations of this model have been reported. CNNs with delay and non-linear templates were introduced in [2-7]. In contrast to a spatial operation approach (which involves an important constraint in the uniformity of weight values and results in spatial convolution with an operator defined by the cloning template) the fixed-point programming approach was adapted for design of the CNN by Tan [2-5]. According to the latter approach the CNN is considered more as a memory than an operator and the space invariant property of the basic CNN is removed. Cellular neural networks can be categorized based on the types of grid, cell, cloning templates and modes of operation employed.

The goal of this chapter is to summarize the theoretical results concerning the basic theory, stability, biological relevance of CNNs, their capabilities and current design approaches of cellular neural networks and their extension.

### 2.2 Basics of Cellular Neural Networks

The following definition of the Cellular Neural Network (or Cellular Nonlinear Network in a more general context) has been given by Chua [2-8]:

**Definition 1.** Cellular Neural Network are:

i) 2-, 3-, or n-dimensional array of

ii) mainly identical dynamical systems, called cells, which satisfies two properties:

iii) most interactions are local within a finite radius r, and

iv) all state variables are continuous valued signals.

#### 2.2.1 Topology of Cellular Neural Networks

In the simplest case, cellular neural networks are composed of an array of analog processors placed on a 2-dimensional geometric grid. Duplicating the grid in 3-dimensions, a multi-layer CNN can be constructed [2-8]. Fig. 2.1 shows three different geometric grid types that are often employed. Each node corresponds to an analog processor called a “cell” and the cells are usually identical. However, motivated partly by neuro-biological structure, the non-uniform cellular neural network architecture having more than one type of cell processor and/or more than one size of
neighbourhood has been introduced [2-7]. These are called *non-uniform processor CNN* (NUP-CNN) or *multiple-neighbourhood-size CNNs* (MNS-CNN) respectively.

![Fig. 2.1 Three different grid types: a) Square grid, b) Hexagonal grid, c) Triangle grid.](image)

**Definition 2. Neighbourhood.** The intercell connections of a CNN are rather different from other conventional neural networks. The interaction between the cell is restricted to the nearest neighbours. This restriction is imposed to limit the number of interconnections and is defined as the distance between \(C(i,j)\) and \(C(m,n)\) [2-3]

\[
d(i, j; k, l) = \max(|k - i|, |l - j|)
\]

(2.1)

The \(r\)-neighbourhood of a cell \(C(i,j)\), in a cellular neural network is defined by

\[
N_r(i,j) = \{C(k,l) | \max(|k - i|, |l - j|) \leq r, 1 \leq k \leq M, 1 \leq l \leq N\}
\]

(2.2)

where \(r\) is a positive integer. Fig. 2.2 shows two different types of neighbourhood of the cell \(C_{ij}\) with \(r=1\) and \(r=2\), respectively, in a 2-D grid [2-2].

**2.2.2 The Basic Model of CNN's Cell**

A model of a CNN cell is shown in Fig. 2.3a. Note that the signals injected into the cell are:
i) External signals which are weighted inputs with suffix \( u \), and the bias term (also called offset term) \( I \).

ii) Internal signals, which include weighted outputs with suffix \( y \) including the cell’s own output.

The connection coefficients \( A(i,j;k,l) \) and \( B(i,j;k,l) \) are called the feedback operator and control operator respectively.

![Fig. 2.2 Two different neighbourhood size of the cell C(i,j).](image)

The basic cell circuit model is shown in Fig. 2.3b [2-2]. The suffixes \( u, x, y \) represent the input, state and output of the cell respectively. There are three node voltages:

i) the input voltage \( v_{uij} \) which is provided by independent voltage source \( E_{ij} \)

ii) the state voltage \( v_{xij} \) as represented by the voltage across the only memory element of the cell, state capacitor \( C \). This capacitor and the resistor \( R_x \) constitute the cell kernel [2-9].

iii) The output voltage \( v_{yij} \) is a nonlinear source which is controlled by \( v_{xij} \).

![Fig. 2.3 a) Model of CNN cell C(i,j), b) Circuit model of CNN cell.](image)
There are a number of nonlinear output functions that can be employed in the CNN’s cell (Piece-wise linear, Gaussian and inverse Gaussian [2-9]). However, it is often appropriate that sigmoid type piece-wise linear transfer characteristics $f(\cdot)$ are used. The function that decides the current through the capacitor of a cell is unique for all the cells. This is because in image processing applications the operation should be invariant under translation of the images. In other words, the interaction between each cell and its nearest neighbour must be uniform over the entire array. However, this is true as for the inner cells. The cells on the boundary of the array must be treated separately (see section 2.6).

2.2.3 The Main Types of System Equations

Analysis of a CNN’s cell circuit results in a complete set of equations describing the dynamics of a CNN. For the original CNN [2-2] these equations are:

**State equation:**

$$C \frac{dv_{ij}}{dt} = -\frac{1}{R_s}v_{ij}(t) + \sum_{C(k,l) \in N_r(i,j)} A(i,j; k,l)v_{ij}(t) + \sum_{C(k,l) \in N_r(i,j)} B(i,j; k,l)v_{ij} + I \quad (2.3)$$

**Output equation:**

$$v_{ij}(t) = f(v_{ij}(t)) \quad (2.4)$$

where

$$f(v_{ij}(t)) = \frac{1}{2}v_{ij}(t) + V_{sat} - |v_{ij}(t) - V_{sat}| \quad 1 \leq i \leq M; 1 \leq j \leq N \quad (2.5)$$

or

$$f(v_{ij}) = \begin{cases} -V_{sat} & \text{if } v_{ij} \leq -V_{sat} \\ v_{ij} & \text{if } |v_{ij}| < V_{sat} \\ +V_{sat} & \text{if } v_{ij} \geq +V_{sat} \end{cases} \quad (2.6)$$

This means that the piecewise-linear function $f(\cdot)$ is bounded as $f(\cdot) \leq V_{sat}$ then $|v_{ij}| \leq V_{sat}$.
**Input Equation:**

\[ v_{uij} = E_{ij}, \quad 1 \leq i \leq M; \quad 1 \leq j \leq N. \]  

\( (2.7) \)

**Constraint Conditions:**

The CNN is subjected to the following constraint:

\[ |v_{uij}(0)| \leq V_{sat}, \quad 1 \leq i \leq M, \quad 1 \leq j \leq N. \]

\[ |v_{uij}(t)| \leq V_{sat}, \quad 1 \leq i \leq M, \quad 1 \leq j \leq N. \]  

\( (2.8) \)

**Parameter Assumption:**

\( i) \ A(i, j; k, l) = A(k, l; i, j) \quad 1 \leq i, k \leq M, \quad 1 \leq j, l \leq N \) (symmetry condition).

\( ii) \ C > 0, \quad R_x > 0 \) (Component condition).  

\( (2.9) \)

The symmetry condition means that the feedback coefficients between two cells are reciprocal in the sense that the corresponding values are the same. However, it has been shown that symmetric cellular neural networks are not reciprocal in the rigorous sense of circuit theory [2-4].

In the CNN literature, these equations are often expressed in other format using the Einstein summation convention as follows [2-10]:

**State Equation:**

\[ \frac{dx^c(t)}{dt} = -x^c(t) + \sum_{d \in N_{c}(c)} a^c_{jd} y^d(t) + \sum_{d \in N_{c}(c)} b^c_{jd} u^d + i^c \]  

\( (2.10) \)

where the variable \( x^c \) denotes the state of the cell \( c \), \( y^c \) its output and \( u^c \) its input. The feedback and forward template coefficients are \( a^c_{jd} \) and \( b^c_{jd} \), respectively. When the Einstein summation convention is applied, a summation is performed automatically for identical indices in the lower and upper positions of variables in a product term. A summation is not done if the indices are parenthesized. With this convention, \( (2.10) \) is written as

\[ \frac{dx^c(t)}{dt} = -x^c(t) + a^c_{jd} y^d(t) + b^c_{jd} u^d + i^c \]  

\( (2.11) \)
Output equation:

\[ y^c(t) = f(x^c(t)) = \begin{cases} 
-1 & \text{if } x^c \leq -1 \\
 x^c & \text{if } |x^c| < 1 \\
+1 & \text{if } x^c > +1 
\end{cases} \quad (2.12) \]

2.2.4 Dynamic Range of Cellular Neural Networks

In order to implement the CNN as a physical system its dynamical range must be known and bounded. This is also called boundedness of states and was expressed as a theorem in [2] as follow:

Theorem 1. All states \( v_{xij} \) in a cellular neural network are bounded for all time \( t>0 \) and the bound \( v_{\text{max}} \) can be computed by the following formula for any cellular neural network:

\[
v_{\text{max}} = 1 + R_x|l| + R_e \max_{1s \leq M, 1s \leq N} \left[ \sum_{C(k,l) \in N_j(i,j)} |A(i,j;k,l)| + |B(i,j;k,l)| \right]. \quad (2.13)\]

Since \( v_{\text{max}} \) is independent of the time and the cell \( C(i,j) \) for all \( i \) and \( j \), we have

\[
\max_i |v_{xij}| \leq v_{\text{max}} \quad \text{for all } 1 \leq i \leq M, 1 \leq j \leq N. \quad (2.14)\]

2.2.5 Binary Output Property

It has been shown [2-2], [2-10] that if the following condition

\[
A(i,j;i,j) > R^{-1} \quad (2.15)
\]

holds, the output of all cells, \( y_{ij}(\infty) \), of a convergent CNN is binary. In other words, for any asymptotically stable system, a bipolar output exists if the following condition holds [2-11]:

\[
\sum_{k \in N_r(i,j)} A(ij;kl)v_{xkl} + \sum_{k \in N_r(i,j)} B(ij;kl)v_{xkl} + I_{ij} \geq 1 \quad \text{if } y_{ij}(\infty) = 1, \quad (2.16)
\]

\[
\sum_{k \in N_r(i,j)} A(ij;kl)v_{xkl} + \sum_{k \in N_r(i,j)} B(ij;kl)v_{xkl} + I_{ij} \leq -1 \quad \text{if } y_{ij}(\infty) = -1, \quad (2.17)
\]
The above conditions can be used to test the solution of the network. It can also be used as a design tool to determine the network templates.

### 2.3 CNNs with Non-Linear and Delay-Type Template Elements

A generalization of the basic CNN is achieved by incorporation of new types of templates called delay type or non-linear type templates [2-7]. Introducing the CNN with non-linear and delay type template elements has made the CNN a powerful framework for general array dynamic. The canonical equation describing the CNN with non-linear and delay type templates is expressed by

\[
C(v_{ij}) = -(1/R_c)v_{ij}(t) + I_j + \sum_{C(k,l) \in N,(ij)} \hat{A}(ij;kl)(v_{kl}(t),v_{ij}(t)) + \sum_{C(k,l) \in N,(ij)} \hat{B}(ij;kl)(v_{kl}(t),v_{ij}(t)) \\
+ \sum_{C(k,l) \in N,(ij)} A^*(ij;kl)v_{kl}(t-\tau) + \sum_{C(k,l) \in N,(ij)} B^*(ij;kl)v_{kl}(t-\tau) 
\]  

(2.18)

In this case instead of two linear controlled sources (A, B), there are nonlinear and delayed controlled sources, \((\hat{A}, \hat{B})\) and \((A^*, B^*)\), respectively.

The structure of the nonlinearity is important, and it is at most a function of two variables, namely the output voltage of the cell \(C(ij)\) and that of a neighbour \(C(kl)\).

In particular, \(\hat{A}(ij;kl)\) and \(\hat{B}(ij;kl)\) are continuous functions of at most two variables, and \(A^*(ij;kl)\) and \(B^*(ij;kl)\) are real constants.

As a further generalization, the output function can be allowed to have its own dynamics too. For example, the output equation (2.4) can be replaced by a first order state equation

\[
\frac{dv_{ij}}{dt} = -av_{ij} + f(v_{ij}(t)) 
\]  

(2.19)

### 2.4 The Full Signal Range (FSR) Model of CNNs

This is a modified version of the original model [2-2] and was introduced in [2-12]. It has been defined by the following equation:

\[
\tau \frac{dx^c}{dt} = g\left[(x^c(t))\right] + I^c + \sum_{d \in N_x(c)} \left\{ A^c_{ij}y^d(t) + B^c_{ij}u^d \right\} 
\]  

(2.20)
where $g(.)$ is defined as follows:

$$
g(x^c) = \lim_{n \to \infty} \begin{cases} 
-m(x^c + 1) + 1, & x^c < -1 \\
-x^c, & \text{otherwise} \\
m(x^c - 1) - 1, & x^c > 1 
\end{cases}
$$

and $m > 1$ is a parameter of the model. This model has properties similar to the original one in many respects. But, it differs in the range interval for the state variables. In the original model the range of the state variables was larger and their bound is given by Eq. (2.14). In contrast, the modified model exhibits identical signal ranges for state and output variables (e.g., $[-V_{sat}, +V_{sat}]$). This results in a more simple design process for current mode CNNs, with reduced area and power consumption [2-12].

### 2.5 Discrete-Time Cellular Neural Networks

Discrete-time cellular neural networks (DTCNNs) are a special type of feedback threshold network where the local interconnections and the shift-invariant weights are transferred from continuous-time CNNs. They are completely described by a recursive algorithm. The dynamic behaviour is based on the feedback of clocked, binary outputs and a single cell is influenced by the inputs and outputs of neighboring cells.

The DTCNN is the discrete-time version of (2.3) and defined by the state equation

$$x^c(k) = a^c_i y^d(k) + b^c_i u^d + i^c$$ (2.21)

and the output equation

$$y^c(k) = f(x^c(k-1)) = \begin{cases} 
+1 & \text{if } x^c(k-1) > 0 \\
-1 & \text{if } x^c(k-1) < 0 
\end{cases}$$ (2.22)

According to Harrer [2-13] DTCNNs offer the following advantages over CTCNNs:

- because of the binary nature of the outputs, the interconnection of several chips is very simple. For CNNs, continuous output signals have to be propagated during the transient.

- the network is robust against fabrication tolerances if the templates are designed appropriately.
• the propagation speed can be controlled within a large range only by changing the clock rate. This also simplifies the testability of a chip.

2.6 Effect of Boundary Cells on the Behaviour of CNNs

Due to the finite neighborhood size of the CNNs, there is a problem with cells close to the boundary of the array. To compensate for the absence of a full complement of nearest neighbors in the boundary of the array it was suggested in [2-3] to surround the rectangular array with imaginary boundary cells, whose output lies between -1 and +1 and remains constant with time. In [2-14] these boundary conditions were referred to as Neumann boundary. Another way of implementing such boundary condition is to modify the state equations of edge cells by adding 1 to the self feedback term for each connection that would be made to a non-existence cell [2-14]. In [2-15] three different groups of CNNs were established based on the influence of these boundary conditions on the dynamical behaviour of CNNs. The first group are those that are always stable, independently of the boundary conditions. For example CNNs where the condition

\[ A(i,j;k,l) > 1 + \sum_{i,j} A(i,j;k,l) \]  

(2.23)

is satisfied. The second group are those ones that are always completely unstable (except for a set of initial conditions of measure zero), no matter what boundary conditions are, since they have no stable equilibrium points at all. Their instability is “intrinsic” to the template that defines them. The third category includes those CNNs that have stable equilibria with some boundary conditions but are completely unstable with other boundary conditions.

2.7 Multi-layer CNNs (MLCNNs)

A complete equivalent way to view a multiple layer CNN is as an L x M x N array of cells with scalar state variables. Often each layer will perform different task. In this interpretation, imaginary boundary cells with scalar valued output are associated with each layer. Using the convolution operator for simplicity of expression, the state equation governing the state of \( C_m(i,j) \) is:

\[
C \frac{d\nu_{ij}}{dt} = -\frac{\nu_{ij}(t)}{R} + A\nu_{ui}(t) + B\nu_{uj} + I
\]  

(2.24)
where

\[ C = \begin{bmatrix} C_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & C_n \end{bmatrix}, \quad R = \begin{bmatrix} R_{11} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & R_{nx} \end{bmatrix}, \quad A = \begin{bmatrix} A_{11} & 0 & 0 \\ 0 & 0 & 0 \\ A_{m1} & A_{mn} \end{bmatrix}, \quad B = \begin{bmatrix} B_{t1} & 0 & 0 \\ 0 & 0 & 0 \\ B_{m1} & B_{mn} \end{bmatrix}. \]

To ensure stability for multiple layer CNNs, in [2-2] it was assumed that layer \( m \) was connected only to the output of lower layers and that the coefficients were symmetric within each layer.

### 2.8. Information Processing and Operating Modes of CNNs

The CNN array is an information-processing device and performs a mapping of [2-8]

\[ \begin{bmatrix} v_{xij}(0) \\ v_{uij}(t) \end{bmatrix} \xrightarrow{f} v_{yij}(t) \quad (2.25) \]

where \( v_{xij}(0) \) and \( v_{uij}(t) \) convey the input information and \( v_{yij}(t) \) conveys the output information. The function \( f \) and hence the processing task performed by the array is a function of the cloning templates and actual time instant at which the output \( v_{yij}(t) \) is sampled. The two modes of the output \( v_{yij}(t) \) are [2-7]:

i) the **DC steady state output** \( (v_{yij}^{\infty}) \), also called fixed point mode operation, in which the output \( v_{yij}(t) \) is sampled once the network has converged to a stable equilibrium state.

ii) the **snapshot** output in which the output is defined at a given time instant \( t = T \), i.e., \( v_{yij}(T) \). This is also called the transient mode of operation.

Besides these two operation modes, there are other modes of operation such as oscillating [2-13], chaotic [2-4].
2.9 Stability of Cellular Neural Networks

Since a cellular neural network (CNN) is a nonlinear analog circuit, its solutions may have a complex dynamical behaviour, from complete stability [2-2] to Chaos [2-4].

Consequently, it is important to provide answers to the following important questions: How can we guarantee the convergence of CNNs? What are the conditions and restrictions for such convergence to be possible?

In [2-2] and [2-3] the symmetry condition (2.9) was given as a sufficient condition for complete stability (e.g., globally asymptotically stable) of a reciprocal CNN where complete stability for the CNNs is specified by following properties [2-16]:

• The state variables (almost) always converge towards an equilibrium which is stable and whose corresponding output values are either +1 or -1.

This result has been obtained using a Liapunov function $E(t)$ in [2-2] where $E(t)$ represents the generalized energy of a CNN, and is defined by the scalar function

$$E(t) = \sum_{(i,j)} v_{ij}(t) \left( -\frac{1}{2} \sum_{(k,l)} A(i,j;k,l)v_{kl}(t) + \frac{1}{2R_j} v_{ij} - \sum_{(k,l)} B(i,j;k,l)v_{kl} - I \right)$$ (2.26)

In [2-17] the stability of a class of nonreciprocal CNNs was reported where the positive cell-linking condition has been introduced as a sufficient condition. This condition does not require the symmetry of lateral feedback coefficients, but, they should be positive. The stability of opposite-sign templates were also reported in the same literature. This result has been obtained using a general theorem given in [2-18]. These two classes of templates are defined as follows [2-17]:

• The template $A$ is called a positive cell linking template for a cellular neural network if the following two conditions are satisfied:

i) \( \sum_{(i,j)} A(i,j;k,l)v_{kl}(t) \geq 0 \), for all \( C(k,l) \in N_i(i,j) \); \( i,j \neq k,l \) (2-27)

ii) the non-zero (hence positive) values of $A(i,j;k,l)$ are strategically located so as to give rise to cell-linking property where any two cells in the entire cell array can be connected by a sequence of cells with positive template values. Similarly, a negative cell linking template can be defined by reversing the inequality sign in condition (i).

• The class of opposite-sign templates is defined by templates values having one of the following structure and sign patterns
The pattern $[r, p, -s]$ with $s, r > 0$ exhibits similar stability properties as symmetric one $[s, p, -s]$.

The stability of opposite-sign templates are covered completely in [2-19], and an example of a CNN that allows a stable limit cycle were reported.

In [2-20], a sufficient condition for the stronger stability of general cellular neural network with nonsymmetric template values was given, which requires that the comparison matrix of $A - I$ be an M-matrix, i.e., matrix with positive real-valued eigenvalues.

In [2-21]-[2-22] the stability of delayed cellular neural networks and nonmonotonic output functions has been analysed by means of a Lyapunov function. According to their investigation:

A system defined by

\[ \dot{x}(t) = -x(t) + A_0 y(t) + A_1 (t - \tau) + Bu(t) + I \]  

(2.28)

is completely stable if the following two conditions hold:

i) there exists a positive diagonal matrix $D$, such that the product of $DA_0$ and $DA_1$ are symmetric matrices;

ii) $\|A_1\| < \frac{2}{3\tau}$

where $A_0, A_1$ and $B \in R^{MxN, MxN}$ are matrices that depend on the way that the cells are ordered.

In [2-23] sufficient conditions for the uniqueness and global asymptotic stability (GAS) of the equilibrium point, and the existance of a stable equilibrium point in the complete saturation region were given as follow :

i) If \( S = \{s_{ij}\} \) is a nonsingular M-matrix, then the origin of the shifted state equation is the unique equilibrium point and is globally asymptotically stable, where $s_{ij}$ is defined by
ii) There exists a stable equilibrium point in the complete saturation for a DCNN with \( u=0 \) if the comparison matrix of \( A^+ - I \) is a nonsingular M-matrix.

### 2.10 Applications

It has been already demonstrated that cellular neural networks are a powerful tool for computational study of vision [2-24]-[2-26]. In [2-24] some of the simple phenomena of the visual pathway (e.g., triadic synapse function, directional selectivity, and length tuning) were realized using CNN. In [2-25] CNN templates for realization of the horizontal cells, the biopolar cells, and functional modules of the inner retina were reported.

Cellular neural networks with their important features such as 2-D structure and local interactions are specially suited for image processing [2-27]-[2-29]. A summary of the main areas of application is as follows [2-30]:

- Enhancement of images (e.g. medical imaging, colour fax, laser printers, radar).
- Reconstruction and compression of images.
- Visual aids and multi-modal sensing.
- Motion control and estimation.
- Modelling in neurobiology.
- Solving partial differential equations in real time.

### 2.11 Design Methods for Cellular Neural Networks

In the simplest case, the cloning template can be considered as the program of the network. Although they can be considered as the synaptic weights of the network, there are two important characteristics which make them different from synaptic weights in conventional neural networks: i) their geometric meaning due to the one-to-one

\[
s_{ij} = \begin{cases} 
1 - a_{ii} - |a_{ij}| & \text{if } i = j \\
-\left(|a_{ij}| + |a_{ji}|\right) & \text{if } i \neq j
\end{cases} \tag{2-29}
\]
topographic correspondence between the cell and the processed signal-array elements (e.g., pixels), and ii) since image processing operations should often be invariant under translation of the image, the interaction between each cell and its nearest neighbours is often uniform over the entire array. For a “3 x 3” case the cloning template can be described by a two $(2r x 1) \times (2r+1)$ real matrix $A$ and $B$, and a constant term $I$

$$A = \begin{bmatrix} a_{i-1,j-1} & a_{i,j-1} & a_{i,j} & a_{i,j+1} & a_{i+1,j-1} \\ a_{i-1,j} & a_{i,j} & a_{i,j+1} \\ a_{i-1,j+1} & a_{i,j+1} & a_{i,j+2} \end{bmatrix} \quad B = \begin{bmatrix} b_{i-1,j-1} & b_{i,j-1} & b_{i-1,j+1} \\ b_{i-1,j} & b_{i,j} & b_{i+1,j} \\ b_{i-1,j+1} & b_{i,j+1} & b_{i+1,j+1} \end{bmatrix} \quad I = c \quad (2.30)$$

The template $A$ and $B$ is often represented as follows:

$$T_a = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \quad T_b = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix}$$

which provides a pictorial view of the synaptic weights.

In some cases the constant current term $I$ (also called bias term) is space varying ($I_{ij}$) and plays a role in “spatial programming”. In this case we have three independent input signal arrays in the array processing applications:

$$S_1(i, j) = v_{uij}(t), \quad S_2(i, j) = v_{xij}(0), \quad S_3(i, j) = I_{ij}$$

The third input array can possibly be time varying as well which could play the role of a spatial geometric program, and this may be a reference map, or a prescribed route, etc., [2-31]. Note that the generic input $v_{uij}(t)$ can be time varying (continuous and discrete-time), but the initial state changed in sample mode only, after the transient has settled.

The template coefficients (synaptic weights) of a CNN can be found by two methods in general: i) design, and ii) learning. These are defined in [2-32] as follows: “by design” means, that the desired function to be performed could be translated into a set of local dynamic rules, while “by learning” is based exclusively on pairs of input and corresponding output signals, the relationship of which maybe far too complicated for explicit formulation of local rule”.

A summary of design and learning methods for CNNs is given here. The most simple, but cumbersome approach for the design of the cloning templates is based on the cut-and-try method. These methods are often based on known image processing algorithms.
In [2-33] an analytical method to design CNNs was reported. This method is based on defining a set of inequalities that describe the task to be accomplished by the network. These inequalities are obtained using the so called comparison principles, which will provide bounds on the state and output waveforms of an analog processing cell circuit. Although solution of this set of inequalities guarantees correct operation of the network, it may be difficult to construct such a set of inequalities for a given task. For some cases, this method is so restrictive and it result in an empty solution space [2-34].

The design method published in [2-11] was the first systematic approach for the design of CNNs which was aimed at programming desired fixed points. For instance, consider the state equation (2.4), and assume that the equilibrium points of the network are stable, i.e., for $t \to \infty \frac{dv_{ij}}{dt} = 0$. Thus

$$v_{ij}(\infty) = \sum_{k,l \in \mathbb{N}} A(ij;kl)v_{pl} + \sum_{k,l \in \mathbb{N}} B(ij;kl)v_{uk} + I_{ij} \quad (2.31)$$

Assuming further that a desired output is binary and in the saturation region, i.e., $v_{ij}(t) = \pm 1$ for the $t \to \infty$ and a fixed input $u_{ij}$, we have:

$$\sum_{k,l \in \mathbb{N}} A(ij;kl)v_{pl} + \sum_{k,l \in \mathbb{N}} B(ij;kl)v_{uk} + I_{ij} \geq 1 \quad \text{if } y_{ij}(\infty) = 1, \quad (2.32)$$

or

$$\sum_{k,l \in \mathbb{N}} A(ij;kl)v_{pl} + \sum_{k,l \in \mathbb{N}} B(ij;kl)v_{uk} + I_{ij} \leq -1 \quad \text{if } y_{ij}(\infty) = -1, \quad (2.33)$$

The system inequalities is then solved using the relaxation method. However, this approach does not ensure control over the basin of attraction of the fixed points because of the nonlinear nature of network. To solve this problem Zou [2-4] suggested the following approach

$$\sum_{k,l \in \mathbb{N}} A(ij;kl)v_{pl}(0) + \sum_{k,l \in \mathbb{N}} B(ij;kl)v_{uk}(0) + I_{ij} - v_{ij}(0) \geq 0 \quad (2.34)$$

if $y_{ij}(\infty) = 1$ and $v_{xij}(0) < 1$

or

$$\sum_{k,l \in \mathbb{N}} A(ij;kl)v_{pl}(0) + \sum_{k,l \in \mathbb{N}} B(ij;kl)v_{uk}(0) + I_{ij} - v_{ij}(0) \leq 0 \quad (2.35)$$
if \( y_{ij}(\infty) = -1 \) and \( v_{xij}(0) > 1 \)

but the reliability of this approach depends on the assumption that the transients of the states of \( v_{xij}(t) \) are monotone.

The design of the fixed points, however, does not guarantee the correct behaviour of the dynamical system, since the initial states do not necessarily lie in the basins of attraction of the correct fixed points. Therefore in [2-35, 2-36] learning algorithms based on minimisation of some function of the system parameters using the gradient descent method. Recurrent Backpropagation (RBP) was applied to CNNs by Balsi [2-35] where piece-wise linear output functions are replaced with sigmoid type output function. But, the change of output function does not guarantee the correct behaviour of the system based on the corresponding original CNNs model [2-37].

Schuler [2-38] applied the modified Backpropagation-through-Time algorithm (BPTT), where the error function was defined as the product of a function of the state at a given time and the integral with respect to the time function of the state variables over the trajectory prior to this time. One of the drawbacks of this algorithm is that only local minima of the error surface are found. Therefore, the result depends on the selected initial parameters [2-32]. Another drawback of this algorithm is its computational complexity. Because, the computation of the gradient of the error surface involves not only forward integration of the differential equation of the CNN in time, but also the Euler-Lagrange equation, which is of the same dimensionality and has to be integrated backwards in time.

A number of different global learning algorithms were proposed which are based on the idea that an objective function is defined, which measures how well the network maps a set of input patterns onto the desired output patterns [2-39, 2-40]. In these approaches the input patterns are inputs for the whole network as opposed to local cell input patterns in local learning algorithms. Therefore, these algorithms themselves are able to design the trajectory. However, global learning algorithms are computationally expensive. As pointed out in [2-32] the global learning algorithm for DTCNNS belongs to the class of NP-complete problems.

In [2-41] the learning was formulated as a genetic optimization algorithm. The cost function is defined as follows:

\[
g(p) = \sum_{i=1}^{k} (y_i^d - y_i(\infty))^2
\]  

(2-36)
where $p$ denotes the parameter vector, i.e., the template, $k$ is the size of the network (the number of cells), $y^i$ is the value of the $i$th pixel of the desired output and $y^{(\infty)}$ stands for the corresponding pixel of the settled output. The $g(p) = 0$ if the result of template $p$ is identical to the desired output and gives a quadratically increasing distance elsewhere. By using $g(\cdot)$ as a cost function the problem of learning can be formulated as an optimisation problem. Applying genetic algorithm $g(\cdot)$ is minimised indirectly: its value is mapped into a fitness value $f(\cdot)$ which is to be maximized. However, it was pointed out that the coding of the coefficients for these algorithms is an open problem, which is important for its success.

There are also some other design methods that are based on a) similarity of the CNN to cellular automata (CA) [2-42], and b) use of the different types of partial differential equations (in their spatially discrete forms) [2-43].

### 2.12 Summary and Conclusions

The high degree of the interconnectivity required by neural networks to emulate neurobiological systems is ultimately constrained by VLSI hardware. Thus the nearest neighbour interactive property of CNNs make them ideal candidates for VLSI implementation. However, there are some CNN applications (e.g., modelling centre-surround receptive field organisation [2-44], texture analysis [2-45], length tuning [2-45]) where a neighborhood size of $r>1$ is required. Such a network requires larger interconnectivity than a $r=1$ CNN and hence much of the benefit inherent in the CNN architecture will be lost.

To exploit the complete capability of CNNs there is a need for a systematic and efficient design methods for CNNs.

Our main aim and contribution to this work is to develop and invent efficient design methods for cellular neural networks which can facilitate the large-scale VLSI implementation of CNNs.
2.13 References of Chapter 2


Chapter 3

Implementation of Large Neighbourhood Cellular Neural Networks

3.1 Introduction

A basic constraint imposed on the VLSI implementation of CNNs is that the neighbourhood size of the CNN is restricted to 1 (r = 1). However, there are some CNN applications (e.g., modelling centre-surround receptive field organization [3-1], texture analysis [3-2], length tuning [3-2] etc.) where a neighbourhood size of r > 1 is required. Such a network requires a large number of interconnections and hence much of the benefit inherent in the CNN architecture (i.e., the local connectivity which makes them an ideal candidate for the analog VLSI implementation) is lost. In [3-3] a method for the implementation of large neighbourhood Discrete-Time CNNs (DTCNNs) has been proposed. However, this method is very expensive in terms of time and circuit complexity because: a) it requires a large number of operations, and b) a framework of the DTCNNs Universal Machine must be provided to deal with the required large signal processing load.

In this chapter two methods for the implementation of large neighborhood CNNs are presented in which the number of interconnections and circuit complexity are preserved at a level which is practical for existing VLSI technology. The proposed methods employ spatially varying bias terms and nonlinear template coefficients to implement r>1 CNNs. The theoretical design, computer simulation, and HSPICE simulation of analogue building blocks for the VLSI implementation of the proposed methods are presented.

3.2 Linear Threshold CNNs

A single layer CNN which operates with binary images can be considered as a binary-input classifier. If such a classifier separates its input patterns into two decision regions, we can refer to it as a two-class binary-input classifier, and the mapping performed by such a classifier can be called a linear threshold function with respect to its binary input
variables. A linear threshold CNN is defined as a CNN with $A(ij; kl) = 0$ for all $ij \neq kl$ [3-4]. For this class of CNN the mapping performed is defined by:

$$\lim_{t \to \infty} y_{ij}(t) = \text{sgn} \left[ (A(ij; ij) - 1)y_{ij}(0) + \sum_{C(k,l) \in N_r(i,j)} B(i,j; k,l)u_{kl} + I \right]$$  \hspace{1cm} (3.1)

The map (3.1) shows that any linearly separable input pattern can be classified using a single layer linear threshold CNN. This can be extended to include all linearly separable input patterns and the initial state variables provided that the condition $A(ij; ij) > 1$ holds. As pointed out in [3-5] the map (1) can be considered as the input/output map of the single layer perceptron. However, the continuous time dynamics and multiple layer capabilities of CNNs extend the possible applications of CNNs beyond those of simple perceptrons [3-5].

To investigate the properties of threshold functions similar to the map (3.1), some technical definitions of decision surfaces and discriminant functions are introduced, and then the discussion is directed to their properties and classifiers based on these types of discriminant functions.

### 3.2.1 Linear Discriminant Functions and Decision Surfaces

A pattern can be considered as a point in the pattern space. Patterns pertaining to different classes will fall into different regions in the pattern space (i.e., different classes of patterns will cluster in different regions and can easily be separated by separating surfaces). Separating surfaces, called decision surfaces, can formally be defined as surfaces in $n$ dimensions which are used to separate known patterns into their respective categories and are used to classify unknown patterns. Such decision surfaces are called hyperplanes and are $(n-1)$-dimensional.

A linear discriminant function that is a linear combination of the components of $x$ can be written as

$$g(x) = w^T x + w_0$$  \hspace{1cm} (3.2)

where $w$ is called the weight vector and $w_0$ the threshold weight. A two class linear classifier implements the following decision rule: categorize $x$ to class $c_{+1}$ if $g(x) > 0$ and categorize to the class $c_{-1}$ if $g(x) < 0$. 

The equation $g(x) = 0$ defines the decision surface that separates the points assigned to class $c_{+1}$ from points assigned to the class $c_{-1}$. When $g(x)$ is a linear function, this decision surface is a \textit{hyperplane}. If $x_1$ and $x_2$ are both on the decision surface, then

$$w^T x_1 + w_0 = w^T x_2 + w_0 \quad (3.3)$$

or

$$w^T (x_1 - x_2) = 0 \quad (3.4)$$

so that the weight vector $w$ is normal to any vector lying in the \textit{hyperplane} which is shown in Fig. 3.1. The separating \textit{hyperplane} $H$ divides the n-dimensional input space into two half-spaces, the decision region $R_1$ for the class $c_{+1}$ (i.e., $g(x) > 0$) and the decision region $R_2$ for the class $c_{-1}$ (i.e., $g(x) < 0$). Therefore, if $x$ is in the $R_1$, then the vector $w$ points into the region $R_1$, and it is said that $x$ is on the positive side of the hyperplane $H$. Similarly, any $x$ in $R_2$ is on the negative side of $H$.

![Fig. 3.1 A separating hyperplane in a 2-dimensional input space.](image)

The discriminant function $g(x)$ can be considered as an algebraic measure of the distance from $x$ to the hyperplane [3-6]. The easiest way to see this is to decompose the vector $x$ into two components (See Fig. 3.2):

$$x = x_q + \rho \frac{w}{|w|} \quad (3.5)$$

where $x_q$ is the orthogonal projection of $x$ onto hyperplane, and $\rho$ is the length of the component $x$ along $w$. Since $g(x_q) = 0$,

$$g(x) = w^T x + w_0 = \rho |w| \quad (3.6)$$
then

\[ \rho = \frac{w^T x}{|w|} \]  

(3.7)

and it is positive if \( x \) is on the positive side and negative if \( x \) is on the negative side. The distance from hyperplane to \( x \) can be obtained by subtracting the distance \( d = \frac{w_0}{|w|} \) from the origin to the hyperplane,

\[ \lambda = \frac{w^T x + w_0}{|w|} \]  

(3.8)

**Definition 1. Bipolar linear threshold function:** a function \( y = f\{x_1, x_2, ..., x_n\} \) is called a bipolar linear threshold function with respect to the bipolar-valued variables \( x_1, x_2, ..., x_n \) if there exists a set of numbers \( \{w_0, w_1, w_2, ..., w_n\} \) such that \( y = 1 \) if and only if \( g(x) > 0 \).

Therefore, a function can be realised by a single layer CNN if and only if it can be represented by a linear threshold function (i.e., both sets of input patterns are linearly separable and the condition \( A(ij;ij) \) holds.)

![Fig. 3.2 Decomposition of the input vector \( x \).](image-url)
3.3 Implementation of Large-Neighborhood CNN with Spatially Varying Bias Terms.

A threshold machine with variable threshold value (i.e., variable bias term, \(w_0\)) and fixed weights is called a variable threshold machine.

The advantage of using the variable threshold value is that it can be easily implemented, since it corresponds simply to a voltage or a current level [3-7]. From the linear discriminant function point of view, two hyperplanes with the same weights except for the threshold value are parallel hyperplanes which are separated by a distance

\[
\frac{w_0^{(1)} - w_0^{(2)}}{|w|}
\]  

(3.9)

Therefore, varying the threshold value corresponds to moving the hyperplane while maintaining its direction. In pattern classification problems, this is interpreted as adjusting the a priori probabilities, modifying the expected relative frequency of occurrence of the two classes and favouring the most likely one [3-7].

As was pointed out in Chapter 2, to exploit the full capability of the CNN, the possibility of applying a spatially variant bias term \(I_{ij}\) as another independent input array has been proposed [3-8]. In this case, the input signal arrays for a CNN are

\[
S_1(ij) = u_{ij}, \quad S_2(ij) = x_{ij}(0) \text{ and } S_3(ij) = I_{ij}
\]

The additional input array \(S_3\) is considered to be an effective way of programming the CNN in array processing applications because the dynamics of a CNN are very sensitive to this term [3-8]. This input array is used here to implement large-neighborhood \(B\) templates.

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} \\
\end{array}
\]

Fig. 3.3 A row of five cells.

Consider the row of five cells (a, b, c, d, e) in Fig.3.3 and an arbitrary (1 x 5) control template \(B\) (i.e. \(r = 2\)):

\[
B = \begin{bmatrix} b_{-2} & b_{-1} & b_0 & b_1 & b_2 \end{bmatrix}
\]  

(3.10)
We further define a convolution operator * as follows [3-9]:

\[ B*V_{uij} = \sum_{C(k,l) \in N_p(i,j)} B(k-i, l-j)V_{ukl} \]  

(3.11)

The result of this convolution for cell c is:

\[ V_{xc} = B*V_{uij} \]

\[ = b_{-2}V_{ua} + b_{-1}V_{ub} + b_0V_{uc} + b_1V_{ud} + b_2V_{ue} \]  

(3.12)

Assuming that the input image values of \( V_{ua} = V_{ue} = V_{uc} \), then (3.12) can be written as:

\[ V_{xc} = B*V_{uij} \]

\[ = (b_{-1})V_{ub} + (b_0 + b_{-2} + b_2)V_{uc} + (b_1)V_{ud} \]  

(3.13)

which is the same as the convolution of the input image with a smaller \( (r = 1) \) kernel:

\[ B_{eq} = \begin{bmatrix} b_{-1} & (b_0 + b_{-2} + b_2) & (b_1) \end{bmatrix} \]  

(3.14)

In other words, based on the assumption of 100% correlation between the inputs of the centre cell and its neighbouring cells at \( r = 2 \), the \( r=2 \) template \( B \) transformed to its equivalent \( r=1 \) template \( B_{eq} \).

However, in general, the assumption that \( V_{ua} = V_{ue} = V_{uc} \) is unlikely to be valid, resulting in a large error in the magnitude of the state variable of the cell c. For example, if the input values of cells a and e are equal and not correlated with the input value of cell c then there is going to be an error of \(-2b_{-2} + 2b_2\) or \(2b_{-2} + 2b_2\) when the inputs of cells a and e are both black (+1) or white (-1), respectively. This error can be cancel out by adding an offset current term \( I_{os} \):

\[ I_{os} = (2b_{-2} + 2b_2) \]  

(3.15)

Therefore (3.9) can be restated as follows:

\[ V_{xc} = (b_{-1})V_{ub} + (b_0 + b_{-2} + b_2)V_{uc} + (b_1)V_{ud} + I_{os} \]  

(3.16)

where the value of \( I_{os} \) depends on the correlation between input values of cells a and e with the input value of the cell a.

To generalize the above approach, consider an \( r = 2 \) CNN. We define the structure of cell indices as shown in Fig. 3.4. Each cell is identified by \( m(r, n) \) where \( n \) represents...
the index number of a cell and \( r \) is the cell’s distance with respect to central cell \( m(0,0) \).

We also define an \( r = 2 \) cloning template as follow:

\[
B = \begin{bmatrix}
w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\
w_{2,16} & w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\
w_{2,15} & w_{1,8} & w_{0,0} & w_{1,4} & w_{2,7} \\
w_{2,14} & w_{1,7} & w_{1,6} & w_{1,5} & w_{2,8} \\
w_{2,13} & w_{2,12} & w_{2,11} & w_{2,10} & w_{2,9}
\end{bmatrix}
\]

where for every template element \((w_{r,n})\), \( r \) and \( n \) have the same meaning as above.

If we assume 100% correlation between the input value (we assume binary input values, i.e. ±1) of the cell \( m(0,0) \) and its neighbouring cells at \( r=2 \) (i.e. input values of cells \( m(2,1) - m(2,16) \)), we can transform the above \( r=2 \) template to a new 3 x 3 template given by

\[
\tilde{B}_{eq} = \begin{bmatrix}
w_{1,1} & w_{1,2} & w_{1,3} \\
w_{1,8} & (w_{0,0} + w_{2,1} + w_{2,3} + \cdots + w_{2,16}) & w_{1,4} \\
w_{1,7} & w_{1,6} & w_{1,5}
\end{bmatrix}, \quad I_{(i,j)eq} = I_0 + I_{(i,j)offset}(3.17)
\]

where \( I_{(i,j)offset} \) is given by

\[
I_{(i,j)offset} = -\left[ (m(0,0) + m(2,1))w_{2,1} + (m(0,0) + m(2,2))w_{2,2} + \cdots + (m(0,0) + m(2,16))w_{2,16} \right]
\]

and \( m(r,n) \) means that the relevant input value is inverted. This approach is valid for all \( r > 1 \) \( B \) templates as long as we are able to compute the \( I_{(i,j)offset} \) term.
3.4 Implementation of Large-Neighborhood CNN Using Non-linear Cloning Templates

The proposed method [3-10], described in section 3.3, enables us to transform a large-neighborhood cloning template into a new cloning template of a lower dimension in which the bias term is spatially varying. This means that a p x p CNN system requires (p x p) distinct bias terms. From the implementation point of view, the introduction of such a p x p matrix makes the following requirements a necessity:

1) pre-processing of the input data in order to define \( I_{(i,j)}^{\text{new}} \). This makes a signal processing arrangement such as a framework of Universal CNNs Machine [3-11] a necessity.

2) An increase in the pin count (unless we assume that a general purpose CNN machine with spatial programming capability is already in place). However, this problem can be solved by implementing Modular CNNs [3-12] where a number of CNN chips, each with a modest number of cells, are interconnected. The signals are robust, i.e. they are not susceptible to intra-chip interference during the interchip communication, since we consider only the design of large-neighborhood CNNs which operate with bipolar images.

In this section a new method to the VLSI implementation of large-neighborhood CNNs is presented which does not need any extra requirements (i.e., a framework of Universal CNNs Machine and modular CNNs implementation). This is achieved by introducing very simple non-linear cloning templates which can be implemented at a minimum cost in terms of the circuit complexity and silicon.

Consider the row of cells in Fig. 3.3, and a (1 x 5) control template \( B=[w_1 \ w_2 \ w_3 \ w_4 \ w_5] \). The output current for the coefficients \( w_1 \) and \( w_2 \) can be defined by

\[
I_{12} = v_{ua}w_1 + v_{ub}w_2
\]  

(3.18)

For a CNN which operates with bipolar images, the magnitude of \( I_{12} \) for different input image values are shown in Table 3.1.

Therefore, we can define another (1 x 3) equivalent non-linear control template \( B_{eq} \) by

\[
B_{eq} = [\hat{w}_{12} \ w_3 \ \hat{w}_{54}]
\]  

(3.19)
where the nonlinear coefficients $\tilde{w}_{12}$ and $\tilde{w}_{54}$ are functions of the $v_{ua} \times v_{ab}$ and $v_{ud} \times v_{ae}$ respectively. For instance, the values of non-linear coefficient $\tilde{w}_{12}$ are summarised in Table 3.2.

Table 3.1 The magnitude of $I_{12}$ for different input image values.

<table>
<thead>
<tr>
<th>$V_{ua}$</th>
<th>$V_{ub}$</th>
<th>$I_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1$</td>
<td>$+1$</td>
<td>$w_1 + w_2$</td>
</tr>
<tr>
<td>$+1$</td>
<td>$-1$</td>
<td>$w_1 - w_2$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$+1$</td>
<td>$-w_1 + w_2$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$-1$</td>
<td>$-w_1 - w_2$</td>
</tr>
</tbody>
</table>

Table 3.2 The values of non-linear coefficient $\tilde{w}_{12}$

<table>
<thead>
<tr>
<th>$v_{ua} \oplus v_{ub}$</th>
<th>$\tilde{w}_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1$</td>
<td>$-w_1 + w_2$</td>
</tr>
<tr>
<td>$+1$</td>
<td>$-w_1 + w_2$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$w_1 + w_2$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$w_1 + w_2$</td>
</tr>
</tbody>
</table>

For example, the characteristic of the non-linear coefficients $\tilde{w}_{12}$ for two different cases:

i) $w_1$ and $w_2$ are both $>0$, and $w_1 > w_2$,

ii) $w_1$ and $w_2$ are both $>0$, and $w_1 < w_2$,

are shown in Fig. 3.5a and Fig. 3.5b respectively.
This approach offers two important advantages as follows:

1) It can be easily extended to the implementation of large-neighborhood feedback templates (A) where a very high gain non-linearity is employed at the output of a CNN cell which operates with bipolar images [2-13].

2) The ease of the VLSI implementation of such non-linear coefficients. In the next section (3.5), we will present a circuit to implement the proposed non-linear coefficients.

3.5 The Proposed Non-linear Coefficient Circuit

In this section, a circuit which can be used to implement the proposed non-linear template coefficients is presented. Since only multiplication by ±1 are performed, there is no need for complex multipliers (e.g. Gilbert multiplier) and the polarity of the coefficient currents can be set by simple analog switches.

Fig. 3.5 Examples of the nonlinear coefficients a) \( w_1 > 0, w_2 > 0 \) and \( w_1 > w_2 \), b) \( w_1 > 0, w_2 > 0 \) and \( w_1 < w_2 \).

The basic functional schematic for the proposed non-linear template coefficients is shown in Fig. 3.6. It consists of two separate XOR sections that are used to determine the magnitude and polarity of the coefficient currents based on the input image values and the variable “Sign” which defines the initial polarity of the coefficient currents and supplied externally.
An electronic circuit for the proposed non-linear coefficient is shown in Fig. 3.7, which is a modified version of the linear coefficient circuitry proposed in [2-13]. However, the XOR circuit used in [2-13] has the following drawbacks: i) it has poor output signal and ii) it requires complementary input signals. Therefore, the XORing circuit proposed in [2-14] is adapted where it requires non-complementary inputs and has very good output signal. It employs two NMOS transistors and a weak pull-up transistor which remains permanently on. To improve the driving capability of the XOR an inverter is added to the output of XOR. However, care should be taken in W/L ratio of NMOS and PMOS transistors of inverter as the XOR circuit is a kind of ratioed logic and the circuit characteristic is greatly affected by the weak transistor geometries.

The proposed circuit consists of: i) two current sources (M1 and M3) and their complements, using a current mirror (M2 and M4). The output signals of the XOR circuits (M7-M9 and M10-M12) are used to set the polarity and magnitude of the coefficient currents via the two sets of analog switches (M5-M6 and M13-M14).

![Diagram](image)

**Fig. 3.6** The basic functional schematic of the non-linear template coefficients.

### 3.6 Simulation Results

To demonstrate the viability of the proposed methods for the implementation of large-neighbourhood cloning templates, a number of simulations were performed as follows:

i) First, an arbitrary (1 x 5) \( r = 2 \) cloning template, \( A = [0 \ 0 \ 2 \ 0 \ 0] \) and \( B = [0.75 \ 0.5 \ 1 \ 0.5 \ 0.75] \), were considered. Simulations were carried out on an arbitrary input image pattern (see Fig. 3.8), using two sets of cloning templates: i) the \( r = 2 \) cloning template given above, and ii) the equivalent \( r = 1 \) templates that are obtained using the proposed...
methods and shown in Fig. 3.9a-3.9c. In all three cases, the resultant state variables were found to be identical as shown in Fig. 3.10.

Fig. 3.7 The electronic circuit for the non-linear template coefficient implementation.

Fig. 3.8 An arbitrary image pattern.

\[
A = \begin{bmatrix} 0 & 2 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1.25 & 1 & 1.25 \end{bmatrix},
\]

\[
I = \begin{bmatrix} -0.25 & 2 & 1.25 & 2.75 & -0.25 & -0.25 & 1.25 & 2.75 & 2 & 1.25 \end{bmatrix}
\]

(a)

(b)

\[
A = \begin{bmatrix} 0 & 2 & 0 \end{bmatrix}, \quad \hat{B} = \begin{bmatrix} b & 1 & b \end{bmatrix}
\]

Fig. 3.9 a) An arbitrary (1 x 5) cloning template, b) the r = 1 equivalent templates obtained using the proposed methods, c) the non-linear coefficient b.
ii) Next, the proposed methods were tested on implementation of the “cut7v” cloning template [3-2] which has neighborhood size of r = 2 and shown in Fig. 3.11a. The equivalent r = 1 version of this template which were obtained using the proposed methods are shown in Fig 3.11b, Fig. 3.12 and Fig. 3.13. Simulations were performed on the image pattern shown in Fig. 3.14a. The resulting output image patterns for both cases are shown in Figs. 3.14b and 3.14c respectively.

\[
A = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0.5 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 0.5 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix}, \quad I = -5.5
\]

(a)

\[
A = \begin{bmatrix}
0 & 1.5 & 0 \\
0 & 2 & 0 \\
0 & 1.5 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 2 & 0 \\
0 & 1 & 0 \\
0 & 2 & 0
\end{bmatrix}, \quad I = -5.5
\]

(b)

Fig. 3.11 a) The original 5x5 cut7v template, b) the 3x3 cut7v template obtained using the proposed method.

iii) Finally, we chose the cloning template for motion detection [3-15] (Fig.3.15a). The equivalent 3 x 3 template for motion detection based on the proposed method is shown in Fig. 3.15b. The computer simulation of these cloning templates for two cases:
case 1) a car movement with the correct speed (i.e. one pixel) and direction (Fig.3.16a-c), case 2) the same object moves in the correct direction but with greater speed (Fig. 3.16d-g).

![Image](image.png)

**Fig. 3.13** a) The nonlinear feedback coefficient b) The nonlinear control coefficient.

![Image](image.png)

**Fig. 3.14** a) The input image, b) the output image in the case of the original 5x5 template and c) the output image in the case of the 3x3 template obtained by the proposed method.

The proposed non-linear coefficient circuit described in section (3.5) has been designed for 1.2 μm AMS CMOS technology and 3.3 volt power supply. The circuit was then simulated on Cadence analog framework 2 SUNSPARC5 computer. The simulation results is shown in Figs. 3.17 where the polarity of the current is correctly function of XORing two input variables $V_{ua}$ and $V_{ub}$. 
\[
A = \begin{bmatrix}
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.3 & 0.3 & 0.3 & 0.0 \\
0.0 & 0.3 & 0.3 & 0.3 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
-0.3 & -0.3 & -0.3 & 0.0 & 0.0 \\
0.0 & 3.10 & -0.3 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}
\]

\[
I = -6.0
\]

Fig. 3.15 a) The original motion detection template, and b) its nearest neighbour equivalent.

Fig. 3.16 Motion detection: the moving car is erased if it moved with an undesired speed.

3.7 Conclusion

A VLSI realization feasibility of CNNs is a key issue which makes the concept very important from the practical standpoint. One of the basic constraints imposed by a
VLSI technology is a limitation of a number of direct cell interconnections and only circuits featuring the r=1 neighborhood size seem to be physically realizable. In this chapter, two new approaches for implementation of large neighborhood CNNs were presented. The proposed approaches offer a significant reduction in circuit complexity of the VLSI implementation of large-neighborhood CNNs. Therefore, we consider the proposed approaches as important steps toward the implementation of universal CNN machine.

Fig. 3.17 Simulation of the proposed non-linear coefficient circuit.
3.8 References of Chapter 3


Chapter 4

Efficient Design Methods For Cellular Neural Networks

4.1 Introduction

The learning capability of neural networks is one of their most favourable features and therefore is a major focus of neural networks research. Learning means here an adaptation process in which the network adjusts its parameters (i.e., template coefficients) according to a number of examples (i.e., training patterns). The originally introduced CNN does not possess any learning algorithm so that the cloning templates of the network in many applications are found either by inspection [4-1] or analytically [4-2], where the local rules defining the task to be performed by the CNN are directly mapped into a set of design inequalities that bound the solution space of the CNN’s parameters for the given task. In [4-3] the first learning algorithm for CNNs was presented in which the cloning template of the network determined from input image patterns and desired output image patterns. The basic method employed in [4-1, 4-2, 4-3] were to set up a system of inequalities which provided the desired output to be a stable equilibrium point. Then the solution of the system design inequalities were computed using an error-correction procedure, the relaxation algorithm [4-4]. However these approaches are complicated and required extensive numerical simulations (e.g. [4-2]) or do not always guarantee functionality of templates which are designed (e.g., the learning algorithm in [4-3]).

In this chapter, first an efficient redundancy method is presented which can be used to reduce the number coefficients of cloning templates which are obtained using learning algorithms in [4-3, 4-5].

Next, a new learning algorithm for CNNs is presented. The Ho-Kashyap algorithm [4-6] is introduced, which is the basic foundation of this learning algorithm. This algorithm offers following advantages: i) it indicates the functionality of the design, ii) a simple design procedure, and iii) fast convergence rate, i.e. the solution can be obtained in a single iteration.
4.2 The Learning Problem in CNNs

A cellular neural network consists of a 2-dimensional (or more) array of locally connected processing elements (i.e., cells) which have the following properties:

i) at each processing step, outputs of all cells perform an identical mapping, \( F \):

\[
\begin{align*}
    x_{ij}(0) & \quad F \\
    u_{ij}(t) & \quad \mapsto y_{ij}(t)
\end{align*}
\]  

(4.1)

where \( x_{ij}(0) \) and \( u_{ij}(t) \) convey the input information and \( y_{ij}(t) \) conveys the output information. In fixed point mode operation, the output \( y_{ij}(t) \) is sampled once the network has converged to a local stable equilibrium state, i.e. \( y_{ij}(t \rightarrow \infty) \).

ii) at each time step, the function \( F \) is determined by an instruction, i.e., the cloning template \( T(A,B,I) \), and each cloning template contains the following information:

a) addresses of the incoming signals and the location at which the result of each operation is to be stored. In fact, they convey a geometric meaning due to the one-to-one topographic correspondence between the cell and processed signal-array elements.

b) the dynamical rules (e.g., pixel classification rules) with respect to its input patterns.

In general, a CNN performs a desired task by appropriate selection of a set of cloning template \( T(A,B,I) \) and the initial data, i.e. the image to be processed which can be either \( u_{ij} \) and/or \( x_{ij}(0) \). A desired output is then obtained as a stable equilibrium of the network.

Therefore the learning problem of a CNN can be formulated as follows: given some training patterns (e.g., examples of the input patterns and the desired output patterns), find a cloning template such that a CNN can map an initial state \( x(0) \) to the desired output pattern for each of the given input patterns.

4.3 Zou's Learning Algorithm for CNNs

In [4-3] a learning algorithm was presented for extracting the cloning template of a CNN from the input image patterns and the desired output image patterns.
Consider a CNN of a r=1 neighborhood which can be described by the following state equation

$$\frac{dx_{ij}}{dt} + x_{ij} = a^T y^* + b^T u^* + i \tag{4.2}$$

where normalized variables will be redefined by using following notations:

- $a \in IR^g$ be the feedback vector,
- $b \in IR^g$ be the control vector,
- $y^* \in IR^g$ be the output pattern vector centered on cell C(ij), and
- $u^* \in IR^g$ be the input pattern vector centered on cell C(ij).

Let assume that the equilibrium points of the network are stable, i.e. for $t \to \infty$, $dx_{ij} / dt = 0$. It follows from (4.2) that

$$x_{ij} = a^T y^* + b^T u^* + i, \text{ (for } 1 \leq i \leq M, 1 \leq j \leq N) \tag{4.3}$$

where $y^* \in IR^g$, $u^* \in IR^g$ are the output and input patterns, respectively, centered on the cell C(ij), M and N are the size of the array containing MxN cells. Assuming further that the desired output is binary, +1 and -1, we get

$$a^T y^* + b^T u^* + i \geq 1, \text{ if } y_{ij} = 1, \text{ (for } 1 \leq i \leq M, 1 \leq j \leq N) \tag{4.4}$$

or

$$a^T y^* + b^T u^* + i \leq -1, \text{ if } y_{ij} = -1, \text{ (for } 1 \leq i \leq M, 1 \leq j \leq N) \tag{4.5}$$

From (4.5) and (4.6) the following general formulation can be obtained:

$$+ (a^T y^* + b^T u^* + i - 1) \geq 0 \tag{4.6}$$

To assure the stability of the network the symmetry of the feedback template A is provided by adding some equality constraints to (4.6). To obtain a more efficient templates (i.e., to have control on the basins of attraction of these fixed points) the following inequalities can be added to the above design inequalities providing a shorter and possibly monotone transients:

$$a^T y^*(0) + b^T u^* + i - x_{ij}(0) \geq 0, \text{ if } y_{ij} = 1 \text{ and } x_{ij}(0) < 1 \tag{4.7}$$
or

\[
\mathbf{a}^T \mathbf{y}^*(0) + \mathbf{b}^T \mathbf{u}^* + \mathbf{i} - x_i^*(0) \leq 0, \text{ if } y_{ij} = -1 \text{ and } x_i^*(0) > 1
\] (4.8)

In [4-3] the relaxation algorithm were then used to find a solution of these system design inequalities.

In [4-5] another learning algorithm was presented which was formulated as an optimization problem. The Simplex optimization algorithm was used to optimize the following cost function:

\[
\text{cost}(T) = \sum_i \delta(T, p_i)
\] (4.9)

where \( \delta(T, p_i) \) denotes the signed distance between the point in the parameter space (i.e., weight space) corresponding to the parameter vector (or template) \( T \) and the plane defined by the \( i \)th inequality.

A number of useful cloning templates had been found using these learning algorithms [4-3, 4-5]. These cloning templates are a) the edge detector, b) the convex corner detector c) the concave corner detector, d) the robust edge detector, e) the robust concave corner detector, f) the WireHor template that can be used to detect the minimal line width violation for horizontal wires in PCB layout, and g) the WireVer template that can be used to detect the minimal line width violation for vertical wires in PCB layout. These are shown in Fig. 4.1.

4.4 The Redundancy Method for Template Coefficients

It has been observed that the dynamics of almost all binary-valued CNNs are monotonic [4-7]. This property of the circuit transients of CNNs was defined by Roska [4-1] in the following form:

\[
\text{if } \frac{dv_{xij}(t_0)}{dt} \leq 0, \text{ then } \frac{dv_{xij}(t)}{dt} \leq 0 \quad (t > t_0)
\]

\[
\text{if } \frac{dv_{xij}(t_0)}{dt} \geq 0, \text{ then } \frac{dv_{xij}(t)}{dt} \geq 0 \quad (t > t_0)
\] (4.10)

This significant property of the circuit transients in classes of CNNs leads to the conclusion that if the system design inequalities hold at \( t = 0 \) then they are also fulfilled in \( 0 < t < t_s \), where \( t_s \) is the settling time of the CNN.
\[
\begin{bmatrix}
0.007863 & -0.074703 & 0.007863 \\
-0.074703 & 1.279297 & -0.074703 \\
0.007863 & -0.074703 & 0.007863
\end{bmatrix}
\begin{bmatrix}
-0.039762 & -0.130159 & -0.039762 \\
-0.130159 & 0.710423 & -0.130159 \\
-0.039762 & -0.130159 & -0.039762
\end{bmatrix}, I = -0.36495
\]

(a)

\[
\begin{bmatrix}
0.015938 & 0.287388 & 0.015938 \\
0.287388 & 0.190105 & 0.287388 \\
0.015938 & 0.287388 & 0.015938
\end{bmatrix}
\begin{bmatrix}
-0.188352 & 0.287739 & -0.188352 \\
0.287739 & 0.190105 & 0.287739 \\
-0.188352 & 0.287739 & -0.188352
\end{bmatrix}, I = -0.42348
\]

(b)

\[
\begin{bmatrix}
0.022 & -0.029 & 0.022 \\
-0.029 & 1.091 & -0.029 \\
0.022 & -0.029 & 0.022
\end{bmatrix}
\begin{bmatrix}
-0.030 & -0.081 & -0.030 \\
-0.081 & 0.437 & -0.081 \\
-0.030 & -0.081 & -0.030
\end{bmatrix}, I = -0.146
\]

(c)

\[
\begin{bmatrix}
0.016 & 1.29 & 0.016 \\
0.045 & 0.016 & 0.045
\end{bmatrix}
\begin{bmatrix}
0.288 & 0.190 & 0.288 \\
-0.188 & 0.288 & -0.188
\end{bmatrix}, I = -0.423
\]

(d)

\[
\begin{bmatrix}
0.40 & 0.00 & 0.40 \\
-0.1 & 0.40 & -0.1
\end{bmatrix}
\begin{bmatrix}
0.20 & -3.0 & 0.20 \\
-3.0 & 0.00 & 0.00
\end{bmatrix}, I = -5
\]

(e)

\[
\begin{bmatrix}
0.40 & 0.00 & 0.40 \\
-0.1 & 0.40 & -0.1
\end{bmatrix}
\begin{bmatrix}
0.20 & 0.00 & 0.20 \\
-3.0 & 2.5 & -3.0
\end{bmatrix}, I = -5
\]

(f)

\[
\begin{bmatrix}
0.022 & -0.029 & 0.022 \\
-0.029 & 1.091 & -0.029 \\
0.022 & -0.029 & 0.022
\end{bmatrix}
\begin{bmatrix}
-0.030 & -0.081 & -0.030 \\
-0.081 & 0.437 & -0.081 \\
-0.030 & -0.081 & -0.030
\end{bmatrix}, I = -0.146
\]

(c)

\[
\begin{bmatrix}
0.288 & 0.190 & 0.288 \\
0.045 & 0.016 & 0.045
\end{bmatrix}
\begin{bmatrix}
-0.188 & 0.288 & -0.188 \\
0.288 & 0.190 & 0.288 \\
-0.188 & 0.288 & -0.188
\end{bmatrix}, I = -0.423
\]

(d)

\[
\begin{bmatrix}
-0.1 & 0.40 & -0.1 \\
0.40 & 0.00 & 0.40 \\
-0.1 & 0.40 & -0.1
\end{bmatrix}
\begin{bmatrix}
0.20 & -3.0 & 0.20 \\
-3.0 & 0.00 & 0.00
\end{bmatrix}, I = -5
\]

(e)

\[
\begin{bmatrix}
0.40 & 0.00 & 0.40 \\
-0.1 & 0.40 & -0.1
\end{bmatrix}
\begin{bmatrix}
0.20 & 0.00 & 0.20 \\
-3.0 & 2.5 & -3.0
\end{bmatrix}, I = -5
\]

(f)

\[
\begin{bmatrix}
0.007863 & -0.074703 & 0.007863 \\
-0.074703 & 1.279297 & -0.074703 \\
0.007863 & -0.074703 & 0.007863
\end{bmatrix}
\begin{bmatrix}
-0.039762 & -0.130159 & -0.039762 \\
-0.130159 & 0.710423 & -0.130159 \\
-0.039762 & -0.130159 & -0.039762
\end{bmatrix}, I = -0.36495
\]

Therefore, it is possible to consider the system design inequalities at t=0 at which \( x_q(0) = y_q(0) \).

From above discussion it is clear that if

a) transients of the state variables \( x_q(t) \) are monotonic,

b) the input image and the initial state are the same,

c) there are no need for propagation of the output signals,

then
the CNN that is designed by learning algorithms in [4-3, 4-5] can be considered as a linear threshold CNN at time t=0 in which \( y_{ii}(0) \) can be defined as another constant input array where \( y_{ii}(0) = x_{yi}(0) \).

This is an crucial observation that enables us to set up an extremely simple method to remove some redundant template coefficients. This is achieved by decomposing the original feedback template, \( A_0 \), into another feedback template \( A_1 \) whose coefficients are all zero except the self-feedback coefficient, and a control template \( B_1 \).

For instance, consider a cloning template \( T(A_0, B_0, I_0) \) which is obtained using learning algorithms in [4-3, 4-5] and defined as follows:

\[
A_0 = \begin{bmatrix}
a_{-1,1} & a_{0,1} & a_{1,1} \\
a_{-1,0} & a_{0,0} & a_{1,0} \\
a_{-1,-1} & a_{0,-1} & a_{1,-1}
\end{bmatrix}, \quad B_0 = \begin{bmatrix}
b_{-1,1} & b_{0,1} & b_{1,1} \\
b_{-1,0} & b_{0,0} & b_{1,0} \\
b_{-1,-1} & b_{0,-1} & b_{1,-1}
\end{bmatrix}, \quad I_0 = I_y
\]

The feedback template \( A_0 \) can be decomposed into \( A_1 \) and \( B_1 \) as follows:

\[
A_0 = \begin{bmatrix}
a_{-1,1} & a_{0,1} & a_{1,1} \\
a_{-1,0} & a_{0,0} & a_{1,0} \\
a_{-1,-1} & a_{0,-1} & a_{1,-1}
\end{bmatrix} \Rightarrow A_1 = \begin{bmatrix}
0 & 0 & 0 \\
0 & a_{0,0} & 0 \\
0 & 0 & 0
\end{bmatrix}, \quad \text{and} \quad B_1 = \begin{bmatrix}
a_{-1,1} & a_{0,1} & a_{1,1} \\
a_{-1,0} & 0 & a_{1,0} \\
a_{-1,-1} & a_{0,-1} & a_{1,-1}
\end{bmatrix}
\]

Then equivalent cloning template \( T_{eq}(A_{eq}, B_{eq}, I_0) \) can then be constructed as follows:

\[
A_{eq} = \begin{bmatrix}
0 & 0 & 0 \\
0 & a_{0,0} & 0 \\
0 & 0 & 0
\end{bmatrix}, \quad B_{eq} = B_1 + B_0 = \begin{bmatrix}
(a_{-1,1} + b_{-1,1}) & (a_{0,1} + b_{0,1}) & (a_{1,1} + b_{1,1}) \\
(a_{-1,0} + b_{-1,0}) & b_{0,0} & (a_{1,0} + b_{1,0}) \\
(a_{-1,-1} + b_{-1,-1}) & (a_{0,-1} + b_{0,-1}) & (a_{1,-1} + b_{1,-1})
\end{bmatrix}, \quad I_0 = I_y
\]

Therefore the size of feedback template is reduced to a single element. This offers a significant reduction in the circuit complexity of the VLSI implementation of CNNs.

### 4.5 Simulation Results 4.1

To confirm the viability of the proposed redundancy method, it is applied to cloning templates in Fig. 4.1, that are obtained by learning algorithms [4-3, 4-5]. The equivalent form of these templates with reduced number of coefficients are shown in Fig. 4.2. Simulation results show that these templates work successfully for image processing.
tasks such as: the edge detection, the convex corner detection, the concave corner
detection, the robust edge detection, the robust concave corner detection and the
minimal line width violation detection. The input and output images for both sets of
cloning templates are shown in Fig. 4.3 and 4.4. These results were obtained using the
CNN simulator (version 3.6) [4-8].

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.279297 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.031899 & -0.204862 & -0.031899 \\ -0.204862 & 0.7104230 & -0.204862 \\ -0.031899 & -0.204862 & -0.031899 \end{bmatrix}, \quad I = -0.36495$$

(a)

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.236609 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.142633 & -0.157648 & -0.031899 \\ -0.157648 & 0.401407 & -0.031899 \\ -0.031899 & -0.031899 & -0.031899 \end{bmatrix}, \quad I = -0.608$$

(b)

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.3 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.0303677 & 0.303677 & -0.143315 \\ 0.0 & 0.303677 & 0.190105 \\ -0.143315 & -0.143315 & -0.143315 \end{bmatrix}, \quad I = -0.6$$

(c)

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.091 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.08 & -0.110 & -0.008 \\ -0.110 & 0.437 & -0.110 \\ -0.008 & -0.110 & -0.008 \end{bmatrix}, \quad I = -0.146$$

(d)

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.29 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.143 & 0.304 & -0.143 \\ 0.304 & 0.190 & 0.304 \\ -0.143 & 0.304 & -0.143 \end{bmatrix}, \quad I = -0.9$$

(e)

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.10 & -2.6 & 0.10 \\ 0.40 & 2.50 & 0.40 \\ 0.10 & -2.6 & 0.10 \end{bmatrix}, \quad I = -5$$

(f)

$$A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.10 & 0.40 & 0.10 \\ -2.6 & 2.50 & -2.6 \\ 0.10 & 0.40 & 0.10 \end{bmatrix}, \quad I = -5$$

(g)

Fig. 4.2 The equivalent linear thresholding version of the cloning templates
in Fig. 4.1: a) the edge detector, b) the convex corner detector c) the
concave corner detector, d) the robust edge detector, e) the robust concave
corner detector, f) the WireHor, and g) the WireVer.
Fig. 4.3 Input and output images for tasks: a) edge detection, b) convex corner detection, c) concave corner detection, d) robust edge detection.
Fig. 4.4 Input and output images for tasks: a) robust concave corner detection, b) horizontal line width checking, and c) vertical line width checking.

4.6 A New Learning Algorithm for CNNs

To obtain the solution of the system design inequalities, in general, two main types of algorithms can be adapted: i) a deterministic approach (i.e. it does not rely on any assumptions concerning the statistical properties of the pattern classes), and ii) a statistical approach.

In [4-3] the relaxation algorithm, a deterministic approach, was employed to solve system design inequalities for the unknown cloning template $T(A,B,I)$. However, this algorithm and its variants such as the perceptron algorithm can be used to obtain a solution vector (i.e. separating vector) when the given training patterns are linearly
separable. These are called error-correction procedures, since they call for a modification of the weight vector when and only when an error is encountered. Therefore, their success on separable problems is largely due to their intensive search for an error-free solution.

One major shortcoming of the error-correction procedure occurs when the training patterns are non-separable. Since no weight vector can correctly categorize every training pattern in a non-separable set, the corrections can continue for as long as they are allowed. It is also impossible to determine the number of steps required for convergence in a separable case, therefore, it is very difficult to be sure whether or not a long training sequence implies that the classes are not linearly separable. Therefore, these types of algorithms can produce a sequence of solution vectors, any member of which may or may not yield a useful solution.

In next section the Ho-Kashap algorithm is first introduced, which is the basic foundation of the proposed learning algorithm. Design methods using this learning algorithm and some design examples are given.

4.6.1 Design of CNNs as Two-class binary classifier

A single layer CNN which operates with binary images can be considered as a binary-input classifier. Such a classifier separates its input patterns into two decision regions. Therefore, it can be referred to as a two-class binary-input classifier, and the mapping performed by such a classifier can be called a linear threshold function with respect to its binary input variables.

The solution of a two-class problem can be restated as a solution of a set of linear inequalities (i.e. the system design inequalities). Let the two classes that are to be separated be \( c_{+1} \) and \( c_{-1} \) which are defined by:

\[
\text{Class } c_{+1} = \left\{ x_{c_{+1}}^{(1)}, x_{c_{+1}}^{(2)}, \ldots, x_{c_{+1}}^{(p)} \right\}
\]

\[
\text{Class } c_{-1} = \left\{ x_{c_{-1}}^{(1)}, x_{c_{-1}}^{(2)}, \ldots, x_{c_{-1}}^{(q)} \right\}
\]

where \( p, q \) are number of patterns belong to class \( C_{+1} \) and \( C_{-1} \), respectively.

Let

\[
x^{T} = (x_0, x_1, \ldots, x_n)
\]
The component \( x_0 = +1 \), and the components \( x_0, x_1, \ldots, x_n \) are the coordinates of the vertices assumed to be a point in \( n \)-dimensional space. It is required to find an \( (n + 1) \) dimensional solution vector \( w \) (i.e., the cloning template \( T(A, B, I) \)) with the property that

\[
\begin{align*}
\mathbf{w}^T \mathbf{x}_{c+1}^{(i)} &> 0 \quad i = 1, 2, \ldots, p \\
\mathbf{w}^T \mathbf{x}_{c-1}^{(i)} &< 0 \quad i = 1, 2, \ldots, q
\end{align*}
\]  

(4.13)

In the latter case, if the input patterns in the class \( c_1 \) are normalized (i.e., multiplied by -1), then the equivalent condition

\[
\begin{align*}
\mathbf{w}^T \mathbf{x}_{c+1}^{(i)} &> 0 \quad i = 1, 2, \ldots, p \\
\mathbf{w}^T - \mathbf{x}_{c-1}^{(i)} &> 0 \quad i = 1, 2, \ldots, q
\end{align*}
\]  

(4.14)

for all the patterns can be obtained. A 2-dimensional example illustrating the solution region for both the normalized and unnormalized case is shown in Fig. 4.5.

Eq (4.14) can be put in the following compact form

\[
\mathbf{X} \mathbf{w} > 0 
\]  

(4.15)

where \( \mathbf{X} \) is a \( (m \times n) \) training matrix of the form

\[
\mathbf{X} = \begin{bmatrix}
\mathbf{x}_{c+1}^{(1)} \\
\vdots \\
\mathbf{x}_{c+1}^{(p)} \\
\mathbf{x}_{c-1}^{(1)} \\
-\mathbf{x}_{c-1}^{(1)} \\
\vdots \\
-\mathbf{x}_{c-1}^{(q)}
\end{bmatrix}
\]  

(4.16)

and \( m = p+q \).
If there exists a solution vector $w$ which satisfies expression (4.15), the inequalities are said to be consistent, otherwise, they are inconsistent. In pattern recognition terminology, the classes are called linearly separable or inseparable, respectively. In switching circuit terminology, the logical function is called a linear threshold function with respect to the input variables, otherwise, it is not a linear threshold function with respect to the input variables, respectively.

### 4.6.2 Minimum-Square-Error Procedures

Consider the problem which is stated as that of finding a vector $w$ which satisfies the equalities

$$Xw = b$$

instead of finding a vector $w$ such that inequalities $Xw > 0$ is satisfied.

If $X$ is square and nonsingular, then Eq. (4.17) can be solved uniquely for $w$ by setting

$$w = X^{-1}b$$

In general, $X$ is a rectangular matrix (since, it is necessary that number of training patterns $(p + q)$ to be larger than the number of variables in input pattern vector, $n$. Thus there exist many solutions to $Xw = b$. 

But, this problem can be restated in the following equivalent form

\[
\begin{align*}
\text{find } w \text{ such that } & J = \frac{1}{2} |e|^2 = \frac{1}{2} |Xw - b|^2 \text{ is minimized } \\
\end{align*}
\]

(4.19)

The gradient of J with respect to w is

\[
\nabla_w J(w) = X^T (Xw - b)
\]

(4.20)

Setting \(\nabla_w J(w) = 0\) implies

\[
X^T Xw = X^T b
\]

(4.21)

or

\[
w = X^\# b
\]

(4.22)

where \(X^\# = (X^T X)^{-1} X^T\) is called the generalized inverse of X.

This procedure makes the training time very much shorter than in the error-correction procedure, since it computes w for all x together and only one solution is needed.

### 4.6.3 The Ho-Kashyap Procedure

When the criterion function J is minimized with respect to both w and b, the training algorithm is called the Ho-Kashyap algorithm [4-6]. The partial derivative of J with respect to w and b is given by

\[
\nabla_w J(w, b) = X^T (Xw - b)
\]

(4.23)

and

\[
\nabla_b J(w, b) = -(Xw - b)
\]

(4.24)

respectively.

Since w is not constrained, \(\nabla J(w) = 0\) yields

\[
w = X^\# b
\]

(4.25)
Since \( b > 0 \) (i.e. all components of \( b \) are constrained to be positive), adjustments on vector \( b \) can be made such that

\[
b(i + 1) = b(i) + \delta b(i)
\]  

(4.26)

where

\[
\delta b_j(i) = \begin{cases} 
2\rho|e(i)| & \text{ when } e(i) > 0 \\
0 & \text{ when } e(i) \leq 0 
\end{cases}
\]  

(4.27)

where \( i, j \) and \( \rho > 0 \) represent iteration number, the component index of the vector, and the scalar correction increment, respectively. From Eq. (4.26) we have

\[
w(i + 1) = X^\#b(i + 1)
\]  

(4.28)

Combining Eqs. (4.25), (4.26) and (4.28), we obtain

\[
w(i + 1) = w(i) + X^\#\delta b(i)
\]  

(4.29)

By letting

\[
e(i) = Xw(i) - b(i)
\]  

(4.30)

the algorithm can be put in the following form:

\[
w(0) = X^\#b(0), \quad b(0) > 0, \text{ but otherwise arbitrary}
\]

\[
e(i) = Xw(i) - b(i)
\]

\[
w(i + 1) = w(i) + \rho X^\#[e(i) + |e(i)|]
\]  

(4.31)

\[
b(i + 1) = b(i) + \rho[e(i) + |e(i)|]
\]

In [4-6], it was pointed out that

\[
X^TX = (2^m)I
\]  

(4.32)
Therefore the algorithm simplifies to

\[
\begin{align*}
    w(0) &= \frac{1}{2^m}X^Tb(0), \quad b(0) > 0, \text{ but otherwise arbitrary} \\
    e(i) &= Xw(i) - b(i) \\
    w(i + 1) &= w(i) + \rho X^T[e(i) + |e(i)|] \\
    b(i + 1) &= b(i) + \rho [e(i) + |e(i)|]
\end{align*}
\] (4.33)

When the inequalities \( Xw > 0 \) have a solution this algorithm converges for \( 0 < \rho \leq 1 \). Moreover, if all the components of \( e(i) \) is non-positive, at any iteration step, this indicates that the classes are non-separable by the specified decision boundary. This feature, test of separability, is an important feature of the algorithm that makes it superior to the error-correction procedures such as the relaxation algorithm as was mentioned earlier. The other advantage of this algorithm is its high convergence rate as mentioned earlier. In most cases, the solution vector is obtained in a single iteration.

### 4.7 Feature Selection of Binary Patterns

Ideally, the use of features reduces the complexity of the pattern recognition problem by extracting from a mass of raw data just that information needed for recognition. This general approach to pattern recognition, the extraction of significant and characterising features followed by classification on the basis of the values of features has been applied for many pattern recognition tasks.

It is very important to determine which input variables of a linear threshold function are most valuable to implement the required function. This data selection approach in design of CNNs plays a central role in area efficient implementation of CNNs. Because the reduction in the dimension of required input patterns leads to reduction in the dimension of required cloning template coefficients and thereby reduction in the complexity of the VLSI implementation of CNNs.

In [4-9] a sequential algorithm for feature selection of binary patterns was introduced. This is a suitable algorithm for selection of essential binary input variables to implement the required function in an efficient way.
The algorithm is introduced with the aid of the following example: Consider the three binary patterns $P_1$, $P_2$, and $P_3$, shown in Fig. 4.6a. The generation of essential features proceeds as follows. First an initial threshold value $\theta$ is arbitrarily selected. Let $f_1(0)$ represent the initial format of feature $f_1$, and let $\|f_1(0) \cap P_1\|$ represent the measure of similarity between $f_1(0)$ and $P_1$, which is defined as the number of 1's in the intersection of $f_1(0)$ and $P_1$. Then, if $\|f_1(0) \cap P_1\| \geq \theta$, let $f_1(1) = f_1(0) \cap P_1$, otherwise $f_1(1) = f_1(0)$. For instance, for $\theta = 3$ in this case the final value of feature $f_1$ is shown in Fig. 4.6b4. To determine the feature $f_2$ the procedure is repeated, with the exception that the threshold is recomputed before making each comparison. The first new value of threshold in this case, is given by $\theta' = \theta + \|f_1(3) \cap (f_2(0) \cap P_1)\|$.

In general, for N binary input patterns $P_1$, $P_2$, $\ldots$, $P_N$, this algorithm can be formalised as follows:

$$f_k(i) = \begin{cases} f_k(i+1) \cap P_i & \text{if } \|f_k(i-1) \cap P_i\| \geq \theta' \\ f_k(i-1) & \text{otherwise} \end{cases} \quad \text{for } i = 1,2,\ldots,N$$

(4.34)

where

$$\theta' = \theta + \|f_k(M) \cap [f_k(i-1) \cap P_i]\| + \|f_k(M) \cap [f_k(i-1) \cap P_i]\|$$

(4.35)

To select a suitable threshold value $\theta$, it is often required to repeat the procedure for several values and to choose the threshold which yields the best result.

### 4.8 Simulation Results 4.2

A number of important cloning templates are designed using the proposed learning algorithm in section 4.6. First, two arbitrary examples will be presented to demonstrate the computational efficiency of the algorithm and its ability to indicate non separable problems. In these examples the vertices of the cube will be given by the decimal equivalent of the binary number associated with it.
Example 1. This is a 4-variable non separable problem for which the non separability is indicated on the first iteration.

Class $C_{+1} = \{0,1,2,4,7,9,14\}$, \hspace{1cm} p = 7

Class $C_{-1} = \{3,5,6,8,10,11,12,13,15\}$, \hspace{1cm} q = 9

$\mathbf{b}^T(0) = [1,1,\ldots,1]$

$\mathbf{w}^T(0) = \frac{1}{8}[-1,-3,-1,-1,-1]$ 

$\mathbf{e}^T(0) = \frac{1}{8}[-3,-5,-5,-5,-9,-11,-13,-9,-9,-7,-5,-5,-3,-5,-3,-1]$
where \( e(0) \) is the nonpositive vector that indicates the non separability of the problem.

**Example 2.** This is a 4-variable separable problem for which the solution is obtained on the first iteration.

Class \( C_+ = \{7, 9, \text{ to } 15\} \), \( p = 8 \)

Class \( C_- = \{0 \text{ to } 6, 8\} \), \( q = 8 \)

\[
b^T(0) = [1, 1, \ldots, 1] \\
w^T(0) = [0, 0.75, 0.25, 0.25, 0.25] \\
e^T(0) = [-1, -0.5, -0.5, 0, -0.5, 0, 0, 0.5, 0, 0, -0.5, 0, -0.5, -1] \\
b^T(1) = b^T(0) + p(e^T(0) + |e^T(0)|) \\
\quad = [1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1] \\
w(1) = w^T(0) + pX^T\{e^T(0) + |e^T(0)|\} \\
w^T(1) = [0, 0.875, 0.375, 0.375, 0.375] \\
\]

where \( w(1) \) is the desired solution.

• The Shadowing Template

The operation of the *Shadowing* template is to produce the shadow of an object which is projected by a light source, in this case, from its right. Therefore required feature variables for this task are the inputs (in this case the initial state variables) of cell \( C(ij) \) and cell \( C(j+1) \). The desired operation can be described in a pictorial way, as shown in Fig. 4.7. The training matrix for the shadowing task can be defined as follows:

\[
x_{\text{shadow}} = \begin{bmatrix} +1 & +1 & -1 \\ +1 & +1 & +1 \\ +1 & -1 & +1 \\ -1 & +1 & +1 \end{bmatrix}
\]
Figure 4.7 The training patterns for the shadowing.

Figure 4.8 shows the extracted shadowing template and two different objects with their shadows.

\[
A = \begin{bmatrix}
0.0 & 0.0 & 0.0 \\
0.0 & 1.3 & 1.3 \\
0.0 & 0.0 & 0.0 \\
\end{bmatrix}, \quad
B = \begin{bmatrix}
0.0 & 0.0 & 0.0 \\
0.0 & 1.3 & 0.0 \\
0.0 & 0.0 & 0.0 \\
\end{bmatrix}, \quad I = 1.3
\]

Fig. 4.8 a) the shadowing template, b-c) two objects and their shadows.
• Connected Component Detector (CCD)

For pattern recognition and data compression, black objects with a single line should be compressed to the size of one pixel and moved to the right boundary, where black and white pixels are alternating depending on the number of objects. The operation of a Connected Component Detector (CCD) can be also considered as a counter which counts the number of connected black bodies in a specified direction. The result of this operation is a number of black cells that remain, each separated by a white cell. The required feature variables for this task are the inputs (the initial state variables) of cell C(j), cell C(j+1) and cell (j-1). The desired operation can be described in a pictorial way, as shown in Fig. 4.9. The training matrix for the CCD can be defined as follows:

\[
X_{\text{CCD}} = \begin{bmatrix}
  +1 & +1 & +1 & +1 \\
  +1 & +1 & +1 & -1 \\
  +1 & +1 & -1 & -1 \\
  +1 & -1 & +1 & -1 \\
  -1 & +1 & +1 & +1 \\
  -1 & +1 & +1 & -1 \\
  -1 & +1 & -1 & -1 \\
  -1 & -1 & +1 & -1 \\
\end{bmatrix}
\]

The training matrix for the CCD is a 9x3 matrix with the following categories and input patterns:

<table>
<thead>
<tr>
<th>Categories</th>
<th>Input Pattern</th>
<th>Desired Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>C+1</td>
<td>□ □ □</td>
<td>□</td>
</tr>
<tr>
<td>C+1</td>
<td>□ □ □</td>
<td>□</td>
</tr>
<tr>
<td>C-1</td>
<td>□ □ □</td>
<td>□</td>
</tr>
<tr>
<td>C+1</td>
<td>□ □ □</td>
<td>□</td>
</tr>
<tr>
<td>C-1</td>
<td>□ □ □</td>
<td>□</td>
</tr>
<tr>
<td>C+1</td>
<td>□ □ □</td>
<td>□</td>
</tr>
<tr>
<td>C-1</td>
<td>□ □ □</td>
<td>□</td>
</tr>
</tbody>
</table>

Fig. 4.9 The training patterns for the CCD.

Figure 4.10 shows the extracted template for CCD, inverted CCD, and their functionality on an arbitrary image object.
\[ A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 2.1 & 2.1 & -2.1 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad l = 0.0 \]

(a)

\[ A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 2.1 & 2.1 & -2.1 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad l = 1.0 \]

(b)

Fig. 4.10 a-b) CCD and INVCCD templates, c-d) input and output images.

- **Horizontal Line Detection**

The Horizontal Line Detection (HLD) can be used to detect a horizontal line which is considered to be a succession of at least two black cells. The required feature variables for this task are the inputs variables of cell C(j), cell C(j+1) and cell (j-1). The desired operation can be described in a pictorial way, as shown in Fig. 4.11. The training matrix for the HLD can be defined as follows:
Fig. 4.11 The training patterns for the HLD.

Figures 4.12-13 show two extracted HLD templates and their operation on two different image objects.

(a) $A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.05 & 1.55 & 1.05 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad I = -1.05$

(b) $A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.05 & 1.55 & 1.05 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.05 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad I = -1.05$

Fig. 4.12 a-b) HLD templates, c) the operation of the HLD.
• The Edge Detector

Edges are the most significant feature of an object. The training input pattern for edge detection can be obtained as follows:

A cell does not belong to the edge of an object (or equivalently class $C_i$), i.e. the given input pattern belongs to class $C_i$ if

i) the cell does not belong to object, i.e. it is white (-1).

ii) the cell is belongs to the inner part of an object, i.e. it is black (or equivalently its input value is one) and the input values of its four one-nearest neighbour off-diagonal are also black.

Figure 4.14 shows the extracted edge detector template and its operation.

\[
A = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0 & -1.0625 & 0.0 \\ -1.0625 & 2.3375 & -1.0625 \\ 0.0 & -1.0625 & 0.0 \end{bmatrix}, \quad I = -1.0625
\]

(a)

Fig. 4.14 a) the edge detector template, b) the operation of the edge detection
The Hole Filing Template

The hole filler problem can be considered as complementary of edge detection and described as follows: white pixels completely enclosed by a black object should be black in the final output state, and it can be achieved by a propagation starting at the boundaries of the cells. Figure 4.15 shows the extracted hole filing template and its operation on two different objects.

\[
A = \begin{bmatrix}
0.0 & 1.0625 & 0.0 \\
1.0625 & 1.0000 & 1.0625 \\
0.0 & 1.0625 & 0.0
\end{bmatrix}, \quad
B = \begin{bmatrix}
0.0 & 2.3375 & 0.0 \\
0.0 & 0.0 & 0.0
\end{bmatrix}, \quad
I = 1.0625
\]

Fig. 15 a) the hole filing template, b-c) the hole filing operation.

The Concentric Contour Template

A concentric contour of a black object is defined as a set of alternating black and white rings starting from the boundary to the interior. The template extracted for the concentric contour extraction is:

\[
A = \begin{bmatrix}
0.0 & -4.125 & 0.0 \\
-4.125 & 6.475 & -4.125 \\
0.0 & -4.125 & 0.0
\end{bmatrix}, \quad
B = \begin{bmatrix}
0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0
\end{bmatrix}, \quad
I = 0.0
\]
Figure 4.16 shows a black square and its concentric contour. This can be used for a special kind of size-independent object recognition in which objects of different size but with identical interior contour rings could be detected by linear thresholding.

![Input Output](image)

Fig. 4.16 a square and its concentric contour.

### 4.9 Summary and Conclusions

The design of programmable and general purpose CNN chips with dimensions that suit the requirements of practical applications is still a major challenge. One of the major issues that has to be tackled when designing a CNN circuit is the realisation of the complete set of required programmable synaptic weights (i.e. cloning template coefficients) in an efficient way in terms of silicon area. The most efficient way is to make some of the weights redundant.

In this chapter, first, an efficient redundancy method for template coefficients is presented. This method can be used to reduced the size of the cloning templates which are obtained by learning algorithms in [4-3, 4-5].

Next, a new learning algorithm for CNNs which uses Ho-kashyap method to solve system design inequalities was introduced. The proposed approach offers following advantages:

i) ease of design.

ii) very fast convergence rate.

iii) a test for realisability of linear threshold CNNs.
4.10 References of Chapter 4


Chapter 5

CONCLUSIONS

5.1 Introduction

In this thesis techniques have been presented to facilitate the efficient design and VLSI implementations of cellular neural networks.

The general design philosophy in this thesis is to facilitate the design of VLSI implementations of large scale practical CNN by allowing only the minimum number of template coefficients and, therefore, the minimum number of interconnections. This strategy starts at CNNs design level and is extended to the level of electronic circuitry.

Section 5.2 summarises the contents of the thesis. In section 5.3 the original contributions are summarised. In section 5.4 recommendations for further research are given. In section 5.5 a list of publications of the author during the course of his PhD is given.

5.2 Summary

* Chapter 1

Over the past few years researchers from many diverse fields have joined forces to develop a new computational paradigm to deal effectively with the massive volume of information available to us today. This paradigm is called neural networks.

In Chapter 1, first, the following aspects of neural networks are described:

i) A number of different opinions on the way to categorise and describe neural networks.

ii) The primary motivations for neural networks research.

iii) The possible applications for neural networks.

iv) Evidence for continuing renaissance in neural networks technology continue, i.e. neural networks are here to stay, and will continue to grow in theoretical understanding, design, and application.
The biological perceptive function is immensely complex and is not yet fully understood. In general, there are two school of thoughts on modelling neurobiological systems. One view involves replicating neurobiological systems as closely as possible. Another uses a more abstract representation of its functions to facilitate computing tasks that are difficult to perform by more conventional means.

However, there are no neural network models yet developed that represent a panacea, in that all have their own intrinsic limitations. Thus a review of some of the most popular neural network models is given in Chapter 1.

In order to capture the fine grained computational parallelism of neural networks a wide range of implementation techniques have been proposed which utilise either software or hardware (i.e. digital/ and or analogue electronics or optical technologies). It is important to assess each technology in terms of their advantages and limitations. Thus, in Chapter 1 a comprehensive comparison between the various technologies which are used to implement neural networks is made.

• Chapter 2

In Chapter 1, it was shown that specialised neural network VLSI implementations offer higher computational power than generalised neurocomputers. One of the practical limitations on the size of networks which can built into a chip is the provision of communication channels, i.e. interconnections.

Cellular Neural Networks are large-scale analogue systems with their origins in the Hopfield neural network model and the architecture of Cellular Automata. The investigation into the design of CNNs was prompted by their specialised architecture. The local interconnectivity makes them a good candidate for implementation in a widely available microelectronics technology. Also, their analog nature enables them to process information in real-time fashion.

In Chapter 2 the following aspects of cellular neural networks are described:

i) A system description of the original CNN and its variants is given.

ii) Stability is an important issue for CNNs. For this reason, some of the key works on the stability of CNNs are reviewed.
iii) The functionality of CNNs is mainly determined by their cloning templates (i.e. synaptic weights). A number of different strategies for obtaining cloning templates are reviewed.

• Chapter 3

A basic constraint imposed on the VLSI implementation of CNNs is that the neighbourhood size of the CNN is restricted to 1 \( (r = 1) \). However, there are some CNN applications (e.g. modelling centre-surround receptive field organisation, texture analysis length tuning, etc.) where a neighbourhood size of \( r > 1 \) is required. Such a system requires a large number of interconnections and hence much of the benefit inherent in the CNN architecture (i.e. the local connectivity which makes them an ideal candidate for the analog VLSI implementation) is lost.

In Chapter 3 two methods for the implementation of large neighbourhood CNNs are presented in which the number of interconnections and circuit complexity are preserved at a level which is practical for existing VLSI technology.

i) The first method employs spatially varying bias terms to achieve virtual template expansion in VLSI implementation of large-neighbourhood CNNs.

ii) In the second method, the virtual template expansion is achieved through the use of non-linear template coefficients.

A practical circuit for the implementation of the proposed non-linear templates coefficients is given in Chapter 3.

Simulation results are presented to confirm the viability of the proposed methods.

• Chapter 4

In general, the cloning templates which define the desired performance of CNNs can be found by solving a set of system design inequalities. These inequalities are formed in a number of ways (e.g., by inspection, analytically, learning algorithms, and etc.). The quality of cloning templates depend on the strategy used to define the system design inequalities and the method used to solve them for unknown template coefficients.

The solution of the system design inequalities is often obtained using an error correction procedure, the relaxation algorithm. However, this algorithm is expensive in
terms of computational time. In addition, the resultant solution vectors do not always guarantee the functionality of design.

In Chapter 4, a new design method for CNNs using the Ho-Kashyap algorithm is proposed. This method offers the following advantages:

i) ease of design for the set of design inequalities.

ii) fast convergence rate (i.e., in most cases the solution of the system design inequalities are obtained in single iteration).

iii) it acts as test for realisability of the design.

One of the major issues that has to be tackled when designing a CNN circuit is the realisation of the complete set of required programmable template coefficients in an efficient way in terms of silicon area. Of course, the most efficient way is to make some of the weights redundant. In Chapter 4, the feature selection method for binary patterns has been described that were found suitable for the design of CNNs with reduced number of template coefficients.

Simulation results are presented to confirm the viability of the proposed methods.

**Over-all Conclusion**

In the last few years, the amount of research on neural networks and the results obtained in the field has been growing at an exponential rate. Consequently, there are also significant progresses on their capabilities and of their operations.

One major motivation for these efforts is the computational power of neural networks in large scale information processing tasks such as cognitive tasks. Such potential is dependent on realising the inherent parallelism of neural network models, which can not truly be captured using conventional computers to implement them. Consequently, this challenge has developed considerable efforts in implementing neural network models in hardware using a wide range of technologies.

From the review presented in Chapter 1, it appears that analogue CMOS VLSI technology offers a high integration and low power dissipation which allows the high degree of parallelism of the neural network model to be captured.

However, analogue electronic implementation of neural networks have not reached the goal of implementing a large scale neural network model on a single chip. One of the
main reasons for this is the constraint imposed by VLSI technologies on the number of physically realisable interconnections.

Cellular neural networks are an example of specialised neural network models which exhibits several interesting features and properties: each neuron is locally connected only to its nearest neighbours, cloning templates (i.e. synaptic weights) are space-invariant and these properties makes CNNs very good candidates for the analogue VLSI implementation. Also their continuous-time operation allows the computational speed to depend only on the time constant of the underlying dynamical system, and therefore CNNs offer a framework for an efficient solution of large scale signal processing problems where a massive computational parallelism and a high processing speed is necessary.

However, the design of programmable and general purpose CNN chips with dimensions that suit the requirements of practical applications is still a great challenge for CNN hardware designers.

Therefore, the techniques that were developed in this thesis offer successful solutions for the implementation of CNNs with minimum number of template coefficients and interconnections, i.e. they facilitate the analogue VLSI implementation of large-scale practical CNNs.

5.3 Original Contributions

The original contributions to the thesis are summarised bellow.

• The use of spatially varying bias terms to achieve the virtual expansion of cloning templates in VLSI implementation of large-neighbourhood CNNs is presented, Chapter 3.

• The use of non-linear template coefficients to achieve the virtual expansion of cloning templates in VLSI implementation of large-neighbourhood CNNs, Chapter 3.

• A suitable circuit for implementation of the virtual large-neighbourhood template coefficients, Chapter 3.

• A new efficient design method for CNNs, Chapter 4.

• A new redundancy method to reduce the number of required template coefficients, Chapter 4.
• The significant effect of employing the feature selection method in the design of CNNs with minimum number of template coefficients, Chapter 4.

Publications.


5.4 Recommendations for Further Research.

- Cellular neural networks are used for solving optimisation problems in which the convergence of a system to a stable state is tracked via the energy function $E$:

$$E(t) = \sum_{(i,j)} v_{yij}(t) \left\{ -\frac{1}{2} \sum_{(k,l)} A(i, j; k, l) v_{ykl}(t) + \frac{1}{2R_F} v_{yij} - \sum_{(k,l)} B(i, j; k, l) v_{ukl} - I \right\}$$

where stable state exists at the global minimum of $E$.

In Chapter 2, it was shown that under the constraints $|x_{ij}(0)| \leq 1$ and $|u_{ij}| \leq 1$ the network with the symmetric feedback template $A$ always produces a stable steady-state output at which the energy function (5.1) is locally minimised.

In [5-1] the hardware annealing method was employed to help the network escape from the local-minimum points and quickly search for the global optimal solution. The annealing is performed by controlling the gain of the neuron.

However, in the presence of finite and random cellular disturbance/noise the energy function (1) can be defined by:

$$E(t) = \sum_{(i,j)} v_{yij}(t) \left\{ -\frac{1}{2} \sum_{(k,l)} A(i, j; k, l) v_{ykl}(t) + \frac{1}{2R_F} v_{yij} - \sum_{(k,l)} B(i, j; k, l) v_{ukl} - I - \theta_i \right\}$$

where $\theta_i$ represents the intracell disturbance/noise and could unstable the progress of the network towards the global minimum.

Thus future work may be focused on developing adaptive spatially varying bias terms to offset the effect of these intracellular disturbances.

- In order to increase possible applications of a single layer linear threshold CNN a multi-step method can be developed to emulate the MADALINE (Many Adaptive Linear Threshold Element) where each step represents an ADALINE (Adaptive Linear Threshold Element).