İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

DESIGN OF ALGEBRAIC AND DYNAMICAL PWL AND PWC NEURAL NETWORKS FOR CLASSIFICATION

Ph.D. Thesis by,

İbrahim GENÇ, MSE

Department : Electronics and Communication Engineering

Programme : Electronics and Communication Engineering

SEPTEMBER 2007

İSTANBUL TECHNICAL UNIVERSITY ⋆ INSTITUTE OF SCIENCE AND TECHNOLOGY

DESIGN OF ALGEBRAIC AND DYNAMICAL PWL AND PWC NEURAL NETWORKS FOR CLASSIFICATION

Ph.D. Thesis by,

İbrahim GENÇ, MSE

504962004

Date of submission : April 09, 2007

Date of defense examination : September 10, 2007

Supervisor (Chairman) : Prof. Dr. Cüneyt GÜZELİŞ

Members of the Examining Committee Prof.Dr. İnci ÇİLESİZ
Assoc.Prof.Dr. H. Özcan GÜLÇÜR (BU)
Prof.Dr. Fikret GÜRGEN (BU)
Asst.Prof.Dr. Neslihan S. ŞENGÖR

SEPTEMBER 2007

İSTANBUL TEKNİK ÜNİVERSİTESİ ⋆ FEN BİLİMLERİ ENSTİTÜSÜ

SINIFLANDIRMA PROBLEMLERİ İÇİN CEBRİK VE DİNAMİK PPD VE PPS YAPAY SİNİR AĞLARININ TASARIMI

DOKTORA TEZİ

Y. Müh. İbrahim GENÇ

504962004

Tezin Enstitüye Verildiği Tarih : 09 Nisan 2007

Tezin Savunulduğu Tarih : 10 Eylül 2007

Tez Danışmanı : Prof. Dr. Cüneyt GÜZELİŞ

Diğer Jüri Üyeleri Prof.Dr. İnci ÇİLESİZ
Doç.Dr. H. Özcan GÜLÇÜR (BÜ)
Prof.Dr. Fikret GÜRGEN (BÜ)
Yard.Doç.Dr. Neslihan S. ŞENGÖR

EYLÜL 2007

PREFACE

I would like to thank my advisor, Prof. Dr. Cüneyt Güzeliş. Without his help, this thesis would never be completed. His corrections, motivations, guidance and predictions have always been just in time. I also thank to my parents, they have ever been compassionate to me, and they have an important role in my academic career.

I acknowledge TÜBİTAK Münir Birsel Foundation for its partly support for my works toward a Ph.D. degree.

With the hope that this thesis improves the humanity, and goodness over the world —no matter it is even small.

SEPTEMBER 2007 İbrahim GENÇ

ABBREVIATIONS

ANN	: Artificial Neural Network
BSB	: Brain State in a Box
CNN	: Cellular Neural Network
CPWL	: Canonical Piecewise Linear
D-HN	: Discrete Hopfield Network
DMLP	: Discrete Multilayer Perceptron
DT-CNN	: Discrete Time Cellular Neural Network
GCNN	: Generalized Cellular Neural Network
FPGA	: Field Programmable Gate Array
MLP	: Multilayer Perceptron
NN	: Neural Network
PWC	: Piecewise Constant
PWL	: Piecewise Linear
PWL-NN	: Piecewise Linear Neural Network
RBFN	: Radial Basis Functions Network
RTL	: Register Transfer Level
SP	: Signal Processing
TSDP	: Two Stage Discrete Perceptron
VHDL	: VHSIC Hardware Description Language,
VHSIC	: Very-High-Speed Integrated Circuit

CONTENTS

LIST OF TABLES

LIST OF FIGURES

DESIGN OF ALGEBRAIC AND DYNAMICAL PWL AND PWC NEURAL NETWORKS FOR CLASSIFICATION

SUMMARY

Piecewise Linear (PWL) and Piecewise Constant (PWC) structures are commonly used in the field of Neural Networks (NNs), even it is not declared that they have such structures. In fact, very known models realize PWL or PWC functions. It is also possible that some complex models could be simplified considering their PWL or PWC counterparts. Despite the fact that PWL and PWC structures are simple, fast and able to approximate to any function, there is not enough emphasis on these subjects on the NNs literature.

This thesis is mainly concerned on PWL and PWC models of NNs to enable profiting the full advantages of PWL/PWC structures by proposing new NN models and developing learning algorithms for both new and existing NNs.

Perceptron with input dependent threshold value is introduced as a new NN structure. It is well known that discrete perceptron cannot classify linearly nonseparable sets. To overcome this problem, perceptron can be modified as its threshold depends on its input. The model brings a solution for a subset of linearly nonseparable classification problems. This is obtained by making the threshold value as a function of input in a PWC manner. What is more important with this model is that all connection weights can be learned using the perceptron learning rule. For the training of the model, an algorithm, three-step perceptron learning algorithm, is proposed. Demonstration of the method is done by edge detection task on black and white images which is indeed a linearly nonseparable task.

Another novelty in the thesis is a new discrete perceptron model forming a cascade structure and being capable of realizing an arbitrary classification task designed by a constructive learning algorithm proposed for this model. The main idea of the model is to copy of a discrete perceptron neuron's output to have a complementary dual output for the neuron, then to select, by using a multiplexer, the true output which might be 0 or 1 depending on the given input. Hence, the problem of realization of the desired output is transformed into the realization of selector signal of the multiplexer. In the next step, the selector signal is taken as the desired output signal of the remaining part of the network. The repeated applications of the procedure renders the problem into a linearly separable one, thus eliminates the necessity of using the selector signal in the last step of the

algorithm. The algorithm completes the construction of the network in a finite number of steps. The convergence theorem for the algorithm associated to the proposed model is introduced and proved in the thesis.

The proposed modification on discrete perceptron brings the universality with the expense of getting just a slight complication in hardware implementation. This is shown in the thesis with an FPGA (Field Programmable Gate Array) hardware implementation of a discrete-weight version of the model.

Cellular Neural Network (CNN) is a dynamical model with a PWL output function and has many applications especially in image processing. However, design or training of CNNs is a great deal and there is not a well defined and good-working learning algorithm for this type of NNs. A Recurrent Perceptron Learning Algorithm for CNNs is analysed and its convergence properties are thoroughly investigated.

In this work, it is also proposed a new class of CNN, in which the initial state of the network are input-dependent. Initial conditions of the network is computed by a piecewise constant function in terms of the external inputs and this function can be learned also by a proposed three-stage Perceptron learning rule.

SINIFLANDIRMA PROBLEMLERİ İÇİN CEBRİK VE DİNAMİK PPD VE PPS YAPAY SİNİR AĞLARININ TASARIMI

ÖZET

Parça Parça Doğrusal (PPD) ve Parça Parça Sabit (PPS) özellikte modeller, çoğunlukla bu türde yapılar olduğu açıkça belirtilmese de Yapay Sinir Ağları (YSA) alanında sıklıkla kullanılmaktadır. Gerçekten de gayet iyi bilinen bir çok model PPS veya PPD işlevler gerçeklerler. Ayrıca, bazı karmaşık modeller, PPS/PPD karşılıkları oluşturularak basitleştirilebilir. PPS ve PPD modeller basitlik, hızlılık ve verilen herhangi bir fonksiyona istenen duyarlılıkla yaklaşabilme özelliklerine rağmen YSA literatüründe yeterli ilgiyi görmemiş veya değerleri açıkça vurgulanmamıştır.

Bu tezde, temel olarak PPS ve PPD YSA modelleri ele alınmış olup, bu alanlarda PPS/PPD yapıların üstünlüklerinden yararlanılmaya çalışılmıştır. Bu amaçla yeni YSA modelleri önerilirken, halihazırda mevcut olan ve bu tezde önerilen PPS/PPD YSA modelleri için öğrenme algoritmaları da geliştirilmiştir.

Girişe bağlı eşikli algılayıcı yeni bir YSA yapısı olarak önerilmektedir. Ayrık algılayıcının doğrusal olarak ayrıştırılamaz kümeleri sınıflandıramadığı gayet iyi bilinmektedir. Bu sorunun aşılması için, algılayıcı eşik değeri girişin PPS bir fonksiyonu olarak değişecek şekilde yeniden düzenlenebilir. Böylece ortaya çıkan model, doğrusal ayrıştırılamayan problemlerin bir alt kümesi için çözüm üretebilir. Buradaki asıl önemli nokta tüm bağlantı ağırlıklarının algılayıcı öğrenme algoritması ile öğrenilebiliyor olmasıdır. Modelin eğitimi için algılayıcı öğrenme kuralının üç aşamalı olarak uygulanmasını içeren bir algoritma önerilmiştir. Modelin ve yöntemin işleyişi, doğrusal ayrıştırılamayan bir sınıflandırma problemi olan siyah beyaz görüntüler üzerinde kenar belirleme ile açıklanmıştır.

Tezdeki yeniliklerden birisi de, herhangi bir sınıflandırma problemini gerçekleştirebilen ve bu model için önerilen yapılandırmacı (constructive) bir öğrenme algoritması ile tasarlanabilen yeni bir kaskad ayrık algılayıcı modelidir. Modelin geliştirilmesindeki ana fikir, ayrık algılayıcının çıkışını diğeri birincisinin tümleyeni olacak şekilde çiftlemek, ardından da bir çoklayıcı ve bir seçici sinyal ile verilen giriş için istenen çıkışı sağlayan hangisi ise onu seçmek üzerine kuruludur. Böylece çözülecek problem istenen çıkışın gerçeklenmesinden seçici sinyalin gerçeklenmesine dönüştürülmüş olur. Bir sonraki adımda her giriş için belirlenen seçici sinyal, ağın geri kalan kısmı için istenen çıkış değerleri

olarak alınır. Yöntemin ardışıl uygulanması ile başlangıçta verilen sınıflandırma problemi doğrusal ayrıştırılabilir bir hale dönüştürülür ve son adımda seçici sinyale olan ihtiyaç ortadan kalkar. Ve bu yolla algoritma ağın kurulumunu sonlu bir adımda tamamlar. Algoritmanın yakınsaklık analizi tezde verilmiş ve ispatı yapılmıştır.

Ayrık algılayıcı üzerinde önerilen değişiklikler her türlü sınıflandırma problemi için genelliği getirirken ağın donanım gerçeklemesi açısından bakıldığında ise önemsenmeyecek düzeyde bir fazlalık getirmektedir. Bu da, önerilen modelin ayrık ağırlık uzayında çalışan bir türünün FPGA (Field Programmable Gate Array) ile gerçeklenmesiyle gösterilmiştir.

Hücresel Yapay Sinir Ağları (HYSA), PPD çıkış fonksiyonuna sahip dinamik bir model olup özellikle görüntü işleme başta olmak üzere bir çok uygulamada kullanılabilmektedir. Fakat HYSA'nın eğitim ya da tasarımı iyi tanımlanmış ve sonuç veren bir öğrenme algoritmasının bulunmaması neeniyle büyük bir sorun oluşturmaktadır. HYSA için önerilmiş olan Yinelenen Algılayıcı Öğrenme Algoritması (RPLA) bu çalışmada çözümlenmiş ve yakınsaklık özellikleri ayrıntılı olarak araştırılmıştır.

Bu çalışmada ayrıca, başlangıç değerlerinin girişe bağlı olarak değiştiği yeni bir HYSA sınıfı önerilmiştir. Ağın başlangıç değerleri HYSA'nın dış girişlerinin bir PPD fonksiyonu ile hesaplanır ve bu fonksiyon önerilen üç aşamalı algılayıcı öğrenme kuralı uygulamasıyla öğrenilebilir.

1. INTRODUCTION

In the society of circuit theory, as a thoroughly investigated branch of electrical engineering, Piecewise Linear (PWL) approximation methods and PWL structures are commonly used. In one of the bible books of circuit theory, it is stated that "PWL approximation is useful in dealing with both simple and general circuits made up of nonlinear resistors. Furthermore, like many graphics methods, the PWL method helps us in understanding qualitatively the nonlinear behaviour of circuits. In addition, its straightforward and effective." [6, p.81] Beside the statement of this book, many publications are released on PWL methods, by many researchers [7–9].

Moreover, PWL and Piecewise Constant (PWC), which is a special case of PWL, structures are commonly used in the field of neural networks, even it is not declared that they have such structures. At a first glance, discrete Multilayer Perceptron (MLP), Cellular Neural Networks (CNNs), Discrete Hopfield Network (D-HN), Discrete Time CNNs (DT-CNNs), Brain State in a Box (BSB) are in fact PWL or PWC models. It is also possible to simplify some other complex models, considering their PWL or PWC counterparts.

The so-called canonical representation plays an important role in the PWL representation theory. Canonical PWL (CPWL) functions are orginally introduced by Chua and Kang [10]. CPWL functions have a very simple structure and they approximate to continuous functions [11] with a relatively small number of parameters [12]. Another important feature of the CPWL functions that any continuous function defined on a compact domain $D \subset R^n$ can be approximated to an arbitrary precision through the CPWL function [11].

Despite the fact that PWL and PWC stuctures are simple, fast and able to approximate to any function, there is not enough emphasis on these subjects on the Artificial Neural Networks (ANN, or simply Neural Networks – NN henceforth) literature. One of the most important works in PWL literature is by Lin and Unbehauen [8]. The paper [8] presents some modifications and applications of classical CPWL functions to supply mathematical background for mapping networks. It also studies the CPWL feature of the popular Multilayer Perceptron-Like (MLPL) networks. It is shown that generalized CPWL is suitable for neural network applications.

On the PWC approximation, as a special case of PWL approximation, Blum and Li provided a view of PWC approximation for the MLPL networks with a hardlimiter output function in the hidden layer [13].

Another PWL-NN is proposed by Batruni [14] which is a cascaded structure with an absolute-value function as an output function of the computation units in the hidden layers.

The objectives of this thesis are, while investigating existing PWL-PWC stuructures in NN models, to propose new models and methods, and to develop new algorithms for both new and existing structures. This thesis is just mainly concerned on PWL and PWC models of NNs. The main motivations behind this idea is to make use of the PWL representation theory into the NN area and to get the full advantage of PWL/PWC structures, such as simplicity, effectiveness and approximation capability.

Before proposing some novelties, it is necessary to present existing models: in chapter 2, NN models are explained. General taxonomy of neural networks, fundamental NN models and learning algorithms to adapt them are all given in the sections of this chapter.

Chapter 3 proposes two new algebraic PWL/PWC models, namely; Two Stage Discrete Perceptron (TSDP) and a cascade multilayer discrete perceptron. Two

learning algorithms for both models are also proposed in this chapter. The one is 3-step perceptron learning algorithm for TSDP and the other is sequential learning algorithm for the cascade discrete multilayer perceptron. Convergence theorem of this sequential learning algorithm is given and proved here.

Beside Chapter 2 explaining existing dynamical PWL/PWC NN models and CNN that is one of them, Chapter 4 introduces Recurrent Perceptron Learning Algorithm (RPLA), an original learning algorithm for completely stable CNN, which is a major example of dynamical PWL NNs. In Chapter 4, furthermore, Threshold class CNNs with input-dependent initial state is proposed as an improvement over the plain CNN model.

2. NEURAL NETWORK MODELS

In this chapter, an insight on Artificial Neural Networks (ANNs) is given. Their history, classification, models, design and learning concepts are all considered.

Artificial Neural Networks (or shortly Neural Networks (NNs) henceforth) with a lot of models, their interdisciplinary nature, design and learning algorithms associated ro them and with their widespread applications, are among the most popular research subjects today. The first NN model is proposed by McCulloch and Pitts [15]. However serious improvements are realized after the proposal of a learning algorithm by Rosenblatt in 1962 [16]. The works of Hebb, Widrow and Hoff, Minsky and Papert, Amari, Kohonen, Grossberg, Hopfield, Rumelhart are milestones on the NN field.

The first examples of neural computers are capable of logical operations and are mainly inspired from biological counterparts. Moreover, with the book of Hebb [17], large scale theory of psychology is introduced into neural networks terminology. This is also the first proposal of a learning scheme for updating neurons' connections toward values that network realizes a desired function. This scheme is called as Hebbian Learning Rule.

In the early 50's, with the concept of cybernetics, an attempt to combine many disciplines such as biology, psychology, engineering and mathematics, the goal was to make machines that could learn. The first neurocomputer were built and tested in 1954 [18] and a couple of years later, in 1958, Perceptron, a trainable machine capable of learning to classify certain patterns by modifying connections to the threshold elements, was invented by Frank Rosenblatt [19]

In the 60's, great improvements on neural networks are realized. With the book of Rosenblatt [20], perceptron is defined and many theorems about it were proved, and also The Perceptron Learning Rule (PLR), one of the most powerful learning algorithms, is proposed. It ensures finding appropriate weights ensuring that perceptron perform a desired function correctly if there is a possibility to do that.

Other important works done in this decade are the introduction of ADALINE (ADAptive LINEar combiner) by Widrow and Hoff and Widrow-Hoff learning rule associated to it [21]. The idea behind the rule is the minimization of summed square error during training.

During the classical period of Perceptron, it seems as if NNs and Perceptrons could do anything. But in 1969 Minsky and Papert publishes the famous book "Perceptrons" in which limitations of perceptron are mathematically proved. Moreover, it is also stated in the book that there is no reason to assume that any of the limitations of single layer perceptrons could be overcome in the multilayer version [22, Sec.13.2].

Minsky and Papert, in the second edition of their books, state that a little of significance had changed since 1969. They reason why progress has been so slow in this field is that researchers unfamiliar with its history have continued to make many of the same mistakes that others have made before them. It may possibly be a connectionist revolution in the sense that there is a great deal of interest and discussion and discoveries have been made that may turn out to be fundamental importance, but certainly no inthat there has been little clear-cut change in the conceptual basis of the field [23, p.vii]. This sentences can be true for the perceptron case, during the 70's and afterwards important improvements occurs in the NNs field.

Self organizing networks and competitive learning algorithms are first proposed in the 70's [24] and Grossberg's work ripens toward 80's and Adaptive Resonance Theory is proposed [25]. Basically, the theory involves a bottom-up recognition

layer and a top-down generative layer. If the input pattern and learned feedback pattern match, a dynamical state called "adaptive resonance" takes places.

In 1982, Hopfield uses the idea of an energy function to formulate a new way of understanding the computation performed by recurrent networks with symmetric synaptic connections. This class of NNs with feedback attracted a great attention in the 80's, and it is known as Hopfield Networks (HNs) [26].

An other important development in 1982 is the publication on self-organizing maps by Kohonen [27] and receives great attention from NNs society.

In 1986, Rumelhart et. al. reports the famous back-propagation algorithm [28]. 2 years later, Radial Basis Functions Network(RBFN) is introduced [29]. This is a layered feedforward network utilizing radial basis functions and being a powerful alternative to MLPs.

Cellular Neural Networks (CNNs) are introduced by Lin and Chua in 1988 and attracts the attention of circuits and systems society together with NNs' [5]. CNNs consist of dynamical and locally connected circuits —or cells— and, are found useful especially for image processing because of their 2-dimensional structure. Güzeliş publishes an article [30] which proposes generalized CNNs as a general model not only a generalization of CNN but also some other models, such as HN and MLP.

In the early 90's, Vapnik and colleagues invents a computationally powerful class of supervised learning networks, called Support Vector Machines (SVMs), for solving patten recognition, regression and density estimation problems [31]. This new method is based on results in the theory of learning with finite sample sizes.

During 90's, efforts are mainly interested in learning algorithms and applications of NNs into very different disciplines. In fact, there are too many articles are published to mention them here.

2.1. General Taxonomy

NNs have a lot of different models. To understand what is a neural network, it is better to group the known models based on their structures and to list common properties of these models. First, some common properties of existing NNs are briefly itemized below and then a taxonomy of NNs is explained.

Common properties of different NN models are as follows.

1. Anatomy of the networks forms a directed graph.
2. Every node is a cell (see items 5, 6).
3. Directed branches of graph correspond to network connections.
4. Network consists of structural units so called cells (see items 5, 6), and these cells are connected within a given geometry (see items 14, 15).
5. Cells are usually multi-input single-output, high-order nonlinear dynamical sub-circuits.
6. Cells consist of three units; (i) —usually linear— algebraic summing unit calculates weighted summation of inputs. (ii) single-input, single-output n^{th}-order dynamical circuit, and (iii) nonlinear algebraic map.
7. Output of the cell can be duplicated and can feed several outputs and inputs of other cells.
8. A cell can be fed by external inputs, neighbouring cell outputs and threshold values.
9. Linear dynamical unit of the cell can be of any-order.
10. Nonlinear algebraic function at the cell output is single-input single-output and can be linear in particular.
11. Connection weights, network geometry and/or internal structures of cells can be changed by a defined learning rule, or they can be held fixed.
12. Cell parameters may be different from cell to cell.

13. Every layer in a network is a multidimensional array of cells.

14. All cells in a layer obey the same connection geometry. So intra-layer connection geometry is regular.

15. Inter-layer connection geometry is well-defined, regular and can be feeforward and/or with feedback.

16. Network is composed of layers which are hierarchically connected to each other.

Although almost all NN models share some common properties, NNs models can be classified into several classes based on structures, functions, learning systems etc.

A functional taxonomy of NNs are depicted in Figure 2.1, and examples for each class is given below.

Stochastic

Amari 1972 [32], Gelenbe 1989 [33]

Deterministic

Distributed parameter NN

Hodgkin and Huxley 1952 [34]

Lumped parameter NN

Algebraic

McCulloch-Pits 1943 [15], Adaline: Widrow 1960 [21], Cognitron: Fukushima 1975 [35]

Dynamical

Discrete Time

McCulloch-Pits 1943 [15], Hopfield 1982 [26]

Continuous Time

First order

Grossberg 1967 [36], Hopfield 1982 [26], CNN: Chua-Yang 1988 [5]

Higher order

Freeman 1987 [37], G-CNN: Güzeliş 1993 [30]

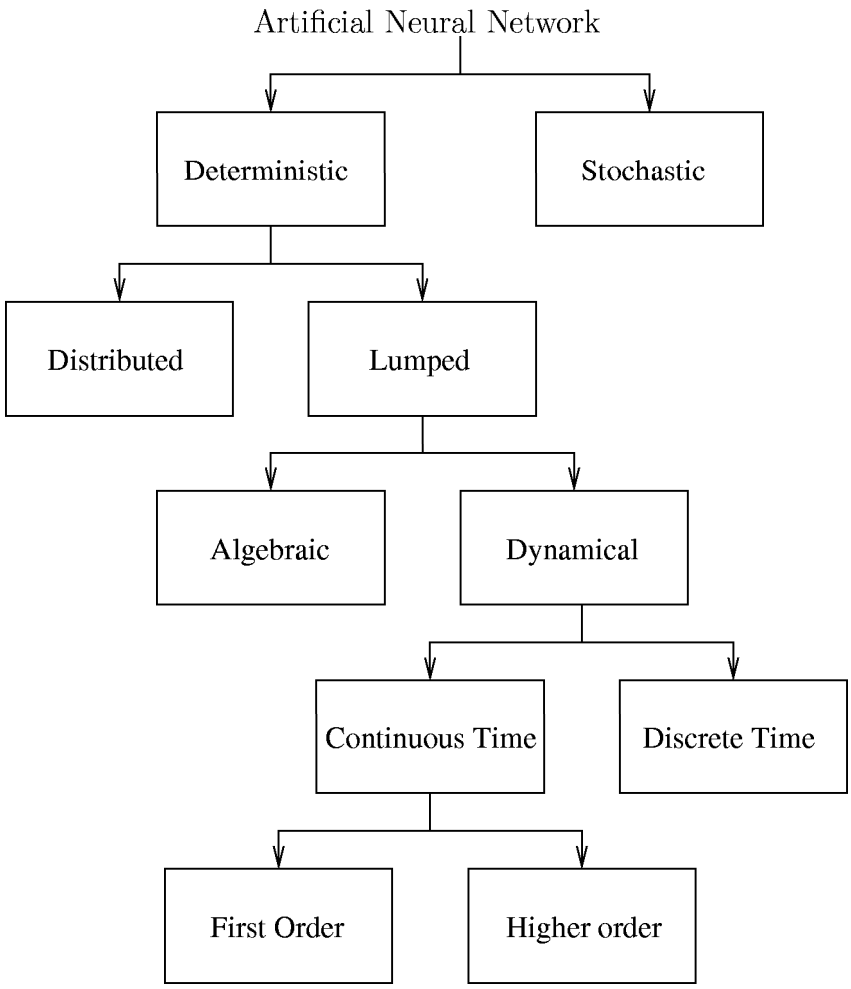


Figure 2.1: NNs functional taxonomy

2.2. Selected NN Models

2.2.1. Perceptron

Perceptron as one of the simplest forms among artificial neural network models consists of only a weighted summation unit and a nonlinear output unit as shown in Figure 2.2. Generally, sigmoid function is used as ouput function and as seen

from the definitions (2.1) and (2.2), perceptron realizes a nonlinear function from p-dimensional real space into the set of $[-1, 1] \subset \Re$.

$$y = f\left(\sum_{i=1}^{p} w_i \cdot x_i - \theta\right) = f\left(\mathbf{w} \cdot \mathbf{x} - \theta\right) \tag{2.1}$$

where $\mathbf{w} \in \Re^p$ are weights, $\mathbf{x} \in \Re^p$ is input vector and $\theta \in \Re$ is the threshold value.

Sigmoid function as common output function is defined as follows for bipolar and unipolar cases:

$$f(\alpha) = 2\frac{1}{1 + e^{-\alpha}} - 1 \tag{2.2}$$

$$f(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{2.3}$$

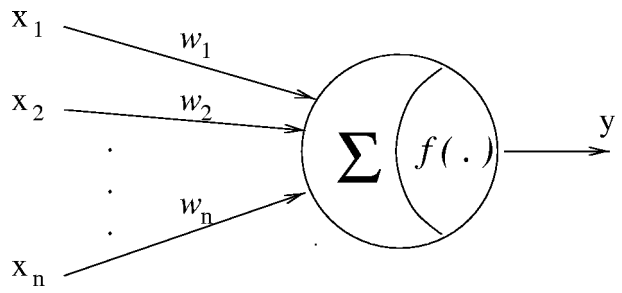


Figure 2.2: Perceptron (For $x_n = -1$, w_n input corresponds to θ)

2.2.2. Discrete Multilayer Perceptron

Discrete MLP is a connection of Perceptrons in a multilayered manner where output function $f(\cdot)$ is either sgn$(\cdot)$ or stp$(\cdot)$ function. Structure of a MLP is shown in Figure 2.3 for the 3-layer case.

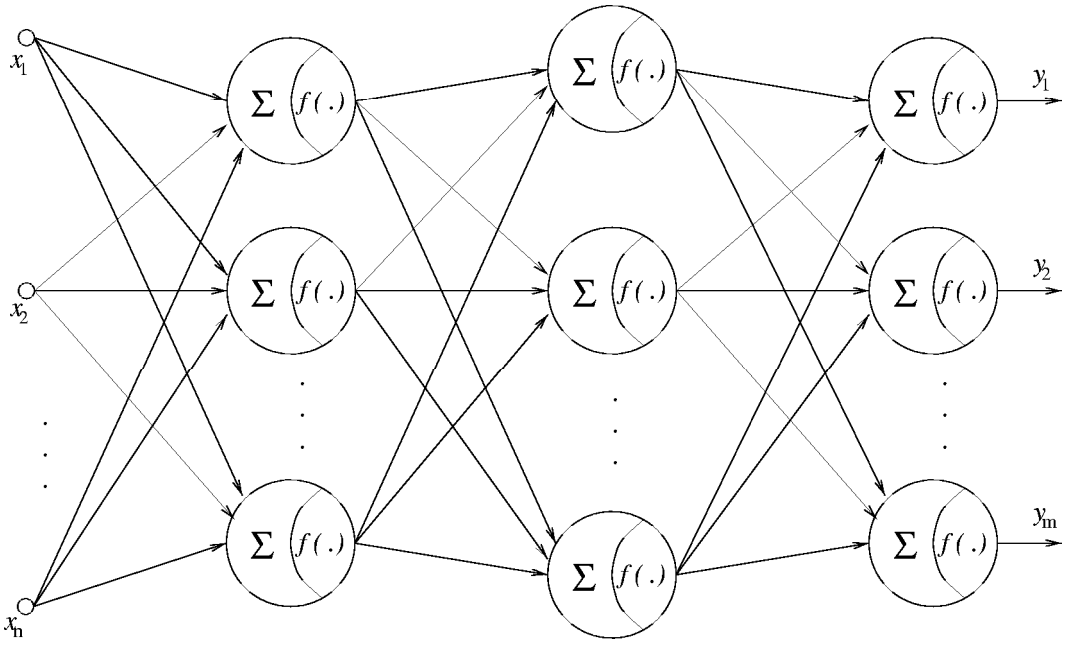


Figure 2.3: Multilayer Perceptron

2.2.3. Hopfield Network

A cell of continuous time Hopfield network consists of a summing input unit, an integral unit and a sigmoid output function unit as depicted in Figure 2.4. Integral unit of the cell can be of higher order as well as 1^{st} order.

Mathematical definition of HN is as follows for $i \in 1, 2, \ldots, n$ where n is the number of cells in the network:

$$\dot{x}_i = -A_i x_i + \sum_{j=1}^{n} w_{ij} f(x_j) + I_i \tag{2.4}$$

$$y_i = f(x_i) = \mathrm{sgm}(x_i) \tag{2.5}$$

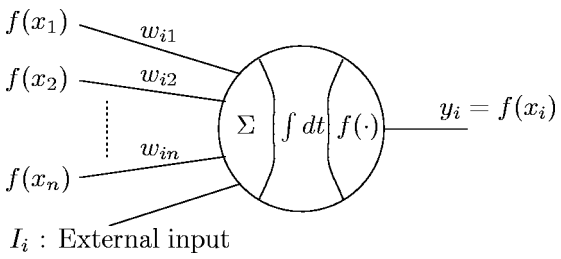


Figure 2.4: A model of continuous time Hopfield network cell

2.2.4. Discrete Hopfield Network

DT-HN is very similar to HN and defined as in (2.6).

$$x_i(k+1) = \begin{cases} 1 & , & > 0 \\ x_i(k) & , & \sum_{j=1}^{n} w_{ij}x_j - T_i \quad = 0 \\ -1 & , & < 0 \end{cases} \tag{2.6}$$

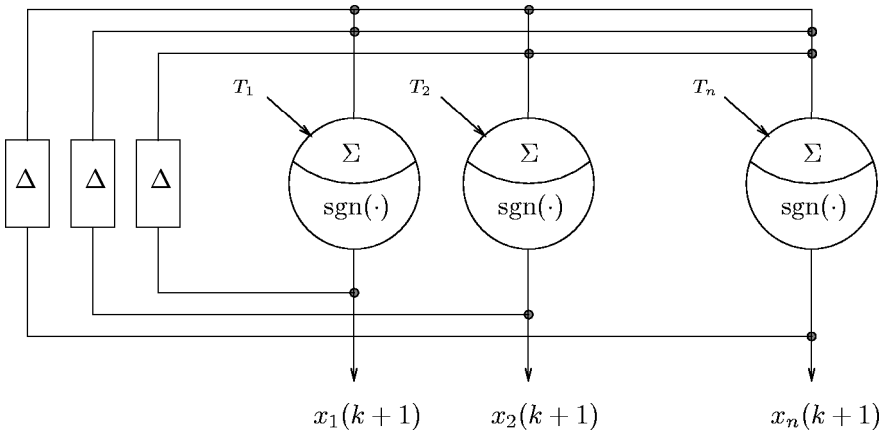


Figure 2.5: Discrete-time Hopfield network

2.2.5. Generalized Cellular Neural Networks

GCNN, is an interconnection of many subcircuits, called cells, each of which is an arbitrary order dynamical circuit and is connected only to its nearest neighbours.

A 2-D special case of Generalized CNN model [30] that covers most of the known neural network architectures including CNNs, HNs and MLPs. While a memoryless NN defines a nonlinear transformation from the input signals space into the output signals space; dynamical NNs such as HN and CNN, have usually been designed as dynamical systems whose trajectories approach to one of the stable equilibrium points depending upon the initial state and input value.

Each cell of the generalized CNN consists of three basic units.(See Fig. 2.6) The first unit which is a multi input, single output, linear, resistive circuit forms a weighted sum of external inputs and the outputs of neighbouring cells. The output of the first unit, e_i, is fed into the second unit. The second unit is a single input (e_i), single output (ξ_i), high order linear dynamical circuit. The only nonlinear part of the cell is the third unit which receives ξ_i and passes it through a nonlinearity $f_i(\cdot)$. Such a cell of 2-D generalized CNN is described by the following system of equations:

$$\dot{\mathbf{x}}_{\mathrm{i}}(t) = \mathbf{A}_{\mathrm{i}}\mathbf{x}_{\mathrm{i}}(t) + b_{\mathrm{i}}e_{\mathrm{i}}(t) \tag{2.7}$$

$$\xi_{\mathrm{i}}(t) = \mathbf{c}_{\mathrm{i}}^{\mathrm{T}}\mathbf{x}_{\mathrm{i}}(t) + h_{\mathrm{i}}e_{\mathrm{i}}(t) \tag{2.8}$$

$$y_{\mathrm{i}}(t) = f_{\mathrm{i}}(\xi_{\mathrm{i}}(t)) \tag{2.9}$$

$$e_{\mathrm{i}}(t) = \sum_{\hat{\mathrm{i}}\in Y} w_{\mathrm{i},\hat{\mathrm{i}}}y_{\hat{\mathrm{i}}}(t-\tau_{\mathrm{i},\hat{\mathrm{i}}}) + \sum_{\hat{\mathrm{i}}\in Y} z_{\mathrm{i},\hat{\mathrm{i}}}u_{\hat{\mathrm{i}}}(t-\sigma_{\mathrm{i},\hat{\mathrm{i}}}) + I_{\mathrm{i}} \tag{2.10}$$

where $\mathrm{A_i} \in \Re^{t_{\mathrm{i}}\times t_{\mathrm{i}}}$; b_{i}, $\mathrm{c_i} \in \Re^{t_{\mathrm{i}}}$; h_{i}, $w_{\mathrm{i},\hat{\mathrm{i}}}$, $z_{\mathrm{i},\hat{\mathrm{i}}}$, $\tau_{\mathrm{i},\hat{\mathrm{i}}}$, $\sigma_{\mathrm{i},\hat{\mathrm{i}}}$, $I_{\mathrm{i}} \in \Re$ are all constants; $\mathrm{x_i}(\cdot) : \Re \to \Re^{t_{\mathrm{i}}}$; $e_{\mathrm{i}}(\cdot)$, $\xi_{\mathrm{i}}(\cdot)$, $y_{\mathrm{i}}(\cdot) : \Re \to \Re$ are functions of time t; $\dot{\mathrm{x}}_{\mathrm{i}} = \mathrm{d}x_{\mathrm{i}}/\mathrm{d}t$, $f(\cdot) : \Re \to \Re$ is a nonlinear function; Y is the set of integers indexing the neighbour cells, and $\mathrm{i} = (\mathrm{i}_1, \mathrm{i}_2)$, $\hat{\mathrm{i}} = (\hat{\mathrm{i}}_1, \hat{\mathrm{i}}_2)$ with i_j, $\hat{\mathrm{i}}_j \in \{1, 2, \ldots, N_j\}$ for all $j \in \{1, 2\}$. Each cell has two different kinds of external inputs: $u_{\mathrm{i}}(t)$ and I_{i}. The controlling input associated with cell C_{i} is also applied through the weights $z_{\mathrm{i},\hat{\mathrm{i}}}$ to the neighbour cells, while the constant input I_{i} is fed only into cell C_{i}. The delay times $\tau_{\mathrm{i},\hat{\mathrm{i}}}$ and $\sigma_{\mathrm{i},\hat{\mathrm{i}}}$ in Eqn. 2.10 are introduced here to obtain a more realistic model: $\tau_{\mathrm{i},\hat{\mathrm{i}}}$ and $\sigma_{\mathrm{i},\hat{\mathrm{i}}}$, respectively, reflects the propagation time needed for cell C_{i} to receive the feedback signals $y_{\mathrm{i}}(t)$'s and the controlling input signals $u_{\mathrm{i}}(t)$'s. The

coefficients $w_{i,\hat{i}}$ and $z_{i,\hat{i}}$ in Eqn. 2.10 weight the delayed signals $y_i(t-\tau_{i,\hat{i}})$ and $u_{\hat{i}}(t-\sigma_{i,\hat{i}})$, respectively.

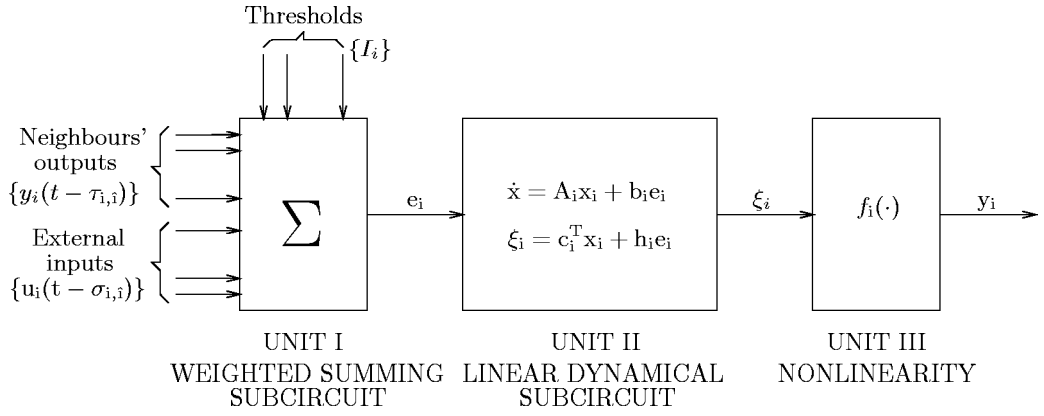


Figure 2.6: Block diagram of a GCNN cell.

CNNs and HNs are special cases of generalized CNNs so that they are 2-D, single layer and first order dynamical networks. MLP is also a special generalized CNN where the network is 1-D, multi-layer and algebraic.

2.3. Learning Algorithms

In the neural network literature, there exists a lot of learning algorithms. Here, only very basic learning algorithms are mentioned. These are Perceptron Learning Rule (PLR), gradient algorithm and backpropogation algorithm.

2.3.1. Perceptron Learning Rule

PLR is simply stated in Eqn. 2.11

$$w(n+1) = w(n) - \eta\left(-y^{s(n)} + \mathrm{sgn}(w^*(n)\cdot x^{s(n)})\right)\cdot x^{s(n)} \tag{2.11}$$

Some properties of the algorithm is given by the theorems below.

Definition 2.1 *Linear separability:*
For a given $\{\mathbf{x}_+^1, \ldots, \mathbf{x}_+^p\}$ *and* $\{\mathbf{x}_-^1, \ldots, \mathbf{x}_-^p\}$ *two set of elements are linearly separable* $\Leftrightarrow \exists \alpha \in \Re^n$, $\beta \in \Re \ni \mathrm{H} = \{\mathbf{x} \in \Re^n | \alpha^T \mathbf{x} - \beta = 0\}$ *(n-1) dimensional hyperplane separates into two sets given below:*

$$\alpha^T \mathbf{x} - \beta \geq 0 \quad \forall \mathbf{x} \in X^+$$

$$\alpha^T \mathbf{x} - \beta < 0 \quad \forall \mathbf{x} \in X^-.$$

Theorem 2.1 X^+ *and* X^- *are linearly separable* $\Leftrightarrow \exists \mathbf{w}^* \ni \varepsilon(\mathbf{w}^*) = 0$

$$\varepsilon(\mathbf{w}) = \sum_{s=1}^{L} \left(y^s - \mathrm{sgn}(\mathbf{w}^T x^s)\right)^2.$$

Theorem 2.2 $w(n+1) = w(n) - \eta \left(-y^{s(n)} + \mathrm{sgn}(w^T(n) \cdot x^{s(n)})\right) \cdot x^{s(n)}$

For a sufficiently small $\eta > 0$ *PLR finds* $\mathbf{w}^*$ *in finite step, if such a* $\mathbf{w}^*$ *exists.*

2.3.2. Backpropogation algorithm

Before given backpropogation algorithm, Eqn 2.12 formulates the Gradient Algorithm.

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \nabla_w \varepsilon(\mathbf{w}) \tag{2.12}$$

Backpropogation algorithm is an extension of gradient algorithm, because gradients are calculated at the output of the network and are propogated backwards to the inputs. A case example of backpropogation algorithm is given below for a 3-layer MLP below.

$$w_{ij}(n+1) = w_{ij}(n) - 2\eta \sum_{s=1}^{L} (y_i^s - \hat{y}_i^s)(-1)(1 - \hat{y}_i^s)(\hat{y}_i^s)\hat{v}_j^s \tag{2.13}$$

$$z_{jk}(n+1) = z_{jk}(n) - 2\eta \sum_{s=1}^{L} \sum_{i=1}^{M} \begin{array}{l} (y_i^s - \hat{y}_i^s)\,(-1)\,(1-\hat{y}_i^s)\,(\hat{y}_i^s) \\ w_{ij}(n)\left(1-\hat{v}_j^s\right)\hat{v}_j^s \hat{u}_k^s \end{array} \tag{2.14}$$

$$t_{kl}(n+1) = t_{kl}(n) - 2\eta \sum_{s=1}^{L} \sum_{i=1}^{M} \begin{array}{l} (y_i^s - \hat{y}_i^s)\,(-1)\,(1-\hat{y}_i^s)\,(\hat{y}_i^s) \\ \sum_{j=1}^{N} w_{ij}(n)\left(1-\hat{v}_j^s\right)\hat{v}_j^s z_{jk}\,(1-\hat{u}_k^s)\,(\hat{u}_k^s)\,\hat{x}_p^s \end{array} \tag{2.15}$$

where, t_{kl}, z_{jk} and w_{ij} are connection weights for the 1^{st}, 2^{nd} and 3^{rd} layer of the MLP, respectively.

3. ALGEBRAIC PWC and PWL NEURAL NETWORKS

Chapter 3 proposes two new algebraic PWC models, namely, "perceptron with input dependent threshold value" (PwIDTV) and "a cascaded network of multiplexed dual output discrete perceptron" (MDODP). Two learning algorithms for both models are also proposed in this chapter. The one is 3-step perceptron learning algorithm for PwIDTV and the other is a sequential learning algorithm for the MDODP. Convergence theorem of this sequential learning algorithm is proposed and proved here.

3.1. Perceptron with Input Dependent Threshold Value

In this section, a special discrete perceptron whose threshold is a function of input patterns is considered [2]. It is well known that discrete perceptron cannot classify linearly nonseparable sets. To overcome this problem, perceptron can be modified as its threshold depends on its input. Proposed structure can be used to classify some kind of linearly nonseparable sets. What is more important is that all connection weights can be learned using the perceptron learning rule. Demonstration of the method is done by edge detection task on black and white images —it is indeed a linearly nonseparable task formulated as a pixel classification problem.

Multilayer perceptron model can give a solution for the problem of classification of linearly nonseparable sets. However learning of the weights which solve the problem is not so straigthforward for the discrete case and backpropogation algorithm cannot be applied here. Generally heuristic methods are employed instead and it is not possible to employ PLR in finding the weights of these

structures, either. So, some dedicated models and learning algorithms brings efficiency for a subset of linearly nonseparable problems brings efficiency. This approach does not lead to generalized methods, models or algorithms, i.e., all may have some restrictions, constraints and assumptions. On the contrary they may have advantages of simplicity, ease of design/learning and straightforward solutions to some kind of problems. When the problem does not fit to the restriction of the method, one then may use the method proposed in Section 3.2., which is an universal model and algorithm for the all classification tasks.

A method proposed here is to use perceptron with nonconstant threshold together with a two-stage PLR. Threshold value of the modified perceptron defined in (3.1)-(3.2) is taken as piecewise constant (See Fig. 3.1.)

$$y = \text{sgn}(\mathbf{w}_1^T \cdot \mathbf{x} + \theta) \tag{3.1}$$

$$\theta = \theta_1 + \theta_3 \cdot \text{stp}(\mathbf{w}_2^T \cdot \mathbf{x} + \theta_2) \tag{3.2}$$

where $\text{stp}(\cdot)$ is unit step function, $\theta_i \in \Re \quad \forall i \in \{1, 2, 3\}$ and $\mathbf{w}_1$, $\mathbf{w}_2 \in \Re^p$ are weight vectors. Instead of unit step function, sign function can also be used. In this case only θ_1 and θ_3 changes.

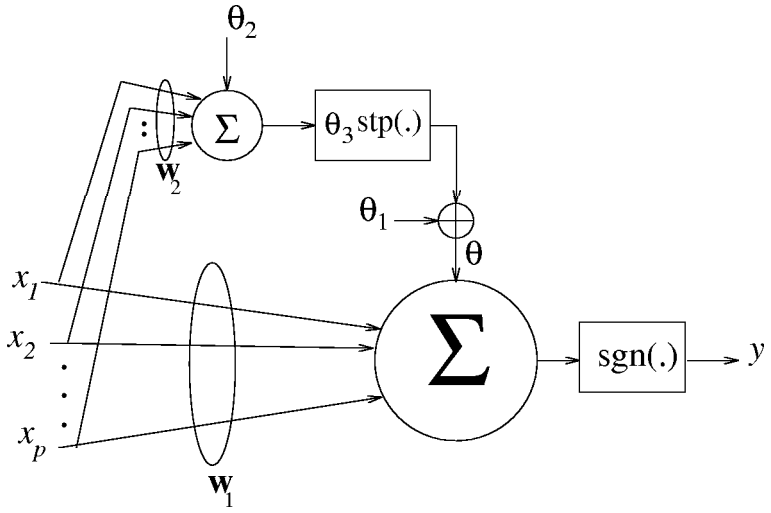


Figure 3.1: Discrete perceptron with input-dependent threshold value [2].

Perceptron with nonconstant threshold, defined in (3.1)-(3.2), can be employed for the classification of some kind of linearly nonseparable input sets described below. Also, all of the parameters, $\mathbf{w}_1$, $\mathbf{w}_2$, θ_i, $i \in \{1, 2, 3\}$, can be learned by an algorithm consisting of PLR's 3-staged application, given in the Section 3.1.1.

The perceptron with varying threshold, given in (3.1) and (3.2), defines a piecewise linear discriminant function so that it assigns inputs belonging to a piecewise linear half space to a class and the others to the second class. To illustrate such a separation of the space, two examples are shown in Figure 3.2. Here, squares and circles represent input vectors belonging to X_+ and X_- in 2-dimensional input space, respectively.

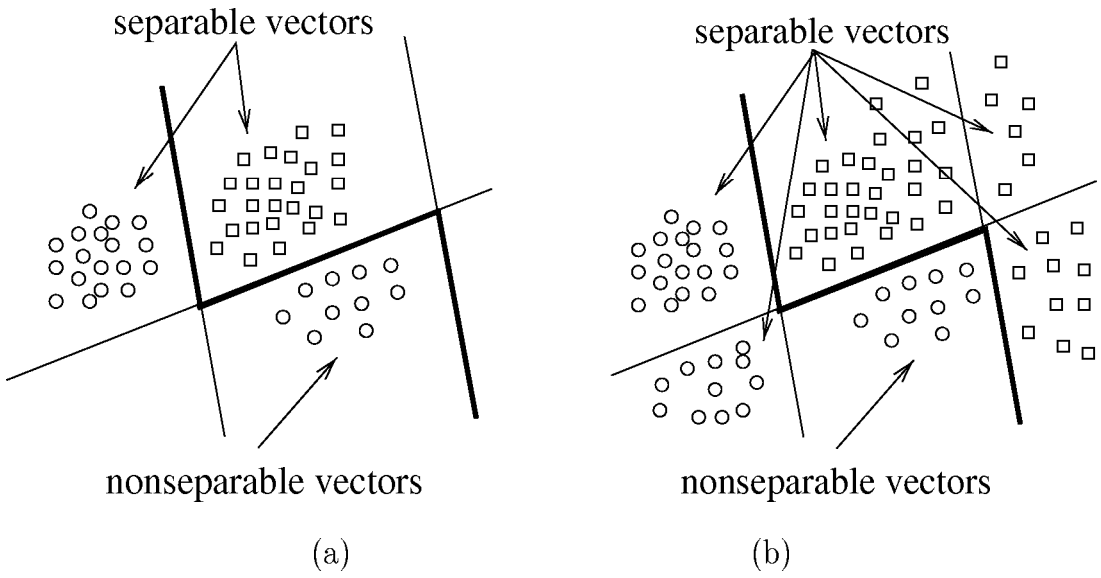


Figure 3.2: Piecewise linear discriminant hyperplane in 2-dimensional space. (a) Linearly nonseparable vectors which are in the same class can be linearly separated from the vectors belonging to a linearly separable subset. (b) Linearly nonseparable vectors which are in the same class cannot be linearly separated from linearly separable ones.

Piecewise linear separation surface contains the points which satisfy the equation $\mathbf{w}_1^T \cdot \mathbf{x} + \theta_1 + \theta_3 \cdot \text{stp}(\mathbf{w}_2^T \mathbf{x} + \theta_2) = 0$. Both of the two examples in Figure 3.2 are linearly nonseparable but can be solved by varying threshold perceptron. However, since linearly nonseparable patterns of the set shown in Figure 3.2.a

are linearly separable from the other patterns of the input set, only this set can be learned by three-step algorithm.

3.1.1. Three-Step Perceptron Learning Algorithm

Proposed learning algorithm, given in Table 3.1 can be applied to the linearly nonseparable set X if the complement of the largest linearly separable subset X_S, $X_{NS} = X \backslash X_S$ contains elements from only either X_+ or X_- and this linearly separable with X_S. By explaining verbally, linearly nonseparable vectors are all from the same class and these are linearly separable from the rest of the set.

To give an example, the problem considered in the Section (3.1.2.), edge detection is a linearly nonseparable problem but obeys these restrictions.

Table 3.1: Three step design algorithm

Step 1: Perceptron is trained by PLR for a given input patterns set X and the parameters $\mathbf{w}_1$ and θ_1 defining hyperplane which separates the largest linearly subset, X_S, of X are obtained.
Step 2: Another perceptron is trained by PLR again and weight vector $\mathbf{w}_2$ and threshold θ_2, both define the hyperplane which leaves linearly nonseparable patterns, X_{NS}, in its positive half-space and linearly separable patterns, X_S, in its negative half-space are obtained.
Step 3: θ_3 can be obtained easily so that its amplitude $\lvert\theta_3\rvert$ should be strictly greater than $\lvert\mathbf{w}_1^T\mathbf{x} + \theta_1\rvert$ for all inputs. θ_3 is taken greater than zero if θ_3, $X_{NS} \subset X_+$, and less than zero, otherwise.

3.1.2. Binary Edge Detection as a Linearly Nonseparable Threshold Function

We will justify our design method by computer simulations done on a specific example, namely the edge detection. First, we pose the binary edge detection task as a linearly nonseparable but piecewise-linearly separable threshold function of the type depicted in Figure 3.2.a. Second, we will train the modified perceptron to learn this task. Then we will examine the performance of this perceptron on the edge detection of some binary images.

Binary edge detection can be considered as a pixel classification problem. To prevent the enormous growth of input set, patterns are constructed from 3×3 window. For each pixel, it can be decided whether it is an edge pixel or not, by investigating the pixel's and its neighboring pixels' values. The input vector to the network is constructed by the values of these pixels. For the example pixel combinations in 1-neighbourhood window depicted in Figure 3.3 the corresponding input vectors and desired outputs are given in Table 3.2.

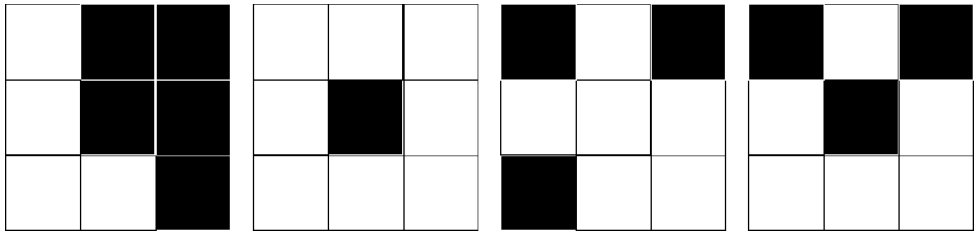


Figure 3.3: Some examples for pixel values in a 3×3 window.

The definition of the ideal edge class stated as "The pixel belongs to edge class if it is black and at least one of its east, west, south and north neighbours is white but not all of its 8 neighbours are white" results in that we identify 239 input patterns of 512 belonging to the edge class. With this definition, a classification problem

Table 3.2: Some examples for pixel values in a 3×3 window.

Figure No	Input vector	Desired output†
3.3(a)	0 1 1 0 1 1 0 0 1	1
3.3(a)	0 0 0 0 1 0 0 0 0	0
3.3(a)	1 0 1 0 0 0 1 0 0	0
3.3(a)	1 0 1 0 1 0 0 0 0	1

† The desired output '1' is used to define the center pixel as an edge pixel and '0' otherwise.

set is obtained containing 512 binary 9-dimensional vectors and corresponding desired outputs.

The set of input – desired output vectors of edge detection problem is investigated and concluded that (1) the set is linearly nonseparable and (2) the set is piecewise linearly separable. The explanations about the investigations are as follows.

A discrete perceptron is trained by PLR for the problem and every attempts to find a separating hyperplane fails with an error. This concludes that the set is linearly nonseparable. Because, according to the PLR convergence theorem if the sets are linearly separable then the algorithm will result in a perceptron separating them [23, p.164] and PLR can be used to test whether such an hyperplane exists or not, considering the perceptron cycling theorem [23, p.182].

The problem is piecewise linearly separable since the nonseparable vectors are linearly separable from the other input vectors. Obviously we can test this case by PLR. The key point here is to find the linearly nonseparable vectors which can be found by employing the PLR [38]. The pocket algorithm, which runs PLR long enough and at the end specifies the weight vector giving the smallest misclassification error, can be used to capture a minimal set of nonseparable vectors.

Since the input space is 9-dimensional we cannot illustrate the status of the set as seen in Figure 3.2 but we could find by experiments that it is linearly nonseparable but piecewise linearly separable. Only one linearly nonseparable vector is found for the problem considered and corresponding pixel structure to this vector is shown in Figure 3.3(b).

The 3-stage algorithm given above does, in fact, test of the input set for linear separability, define nonseparable vectors and test the linear separability of nonseparable vectors from other vectors of the set. Meanwhile, it finds the following weight vectors and thresholds if the cases are admissible with the prerequisites of the method.

For the defined edge detection problem the following weights are found by the algorithm.

$$\mathbf{w}_1 = [-0.013 \;\; -0.406 \;\; -0.024 \;\; -0.74 \;\; +2.55 \;\; -0.485 \;\; -0.21 \;\; -0.88 \;\; -0.13] \quad (3.3)$$

$$\mathbf{w}_2 = [-1.28 \;\; -1.28 \;\; -1.28 \;\; -1.28 \;\; +1.28 \;\; -1.28 \;\; -1.28 \;\; -1.28 \;\; -1.28] \quad (3.4)$$

$$\theta_1 = [-0.427] \qquad \theta_2 = [-10.53] \qquad \theta_3 = [-42.5] \quad (3.5)$$

The results obtained by the modified perceptron with the weights above, edge detector on some binary images are presented in Figure 3.4. For these images the modified perceptron finds the edge images which are ideal in the above sense. Note that the images obtained do not contain any isolated pixel and the edges in the resulting images are of one pixel width.

The proposed method could also be applied on learning edges for gray level images. However, to define ideal edges in an exact manner for gray level images seems very difficult (may be impossible) considering the following facts. We will have 256^9 training pairs for 256 gray-level images processed by perceptron in a 3×3 input window. In order to define ideal edges, one can try to use an available

edge detection method but it might yield an inconsistent training set for the considered 3x3 neighborhood.

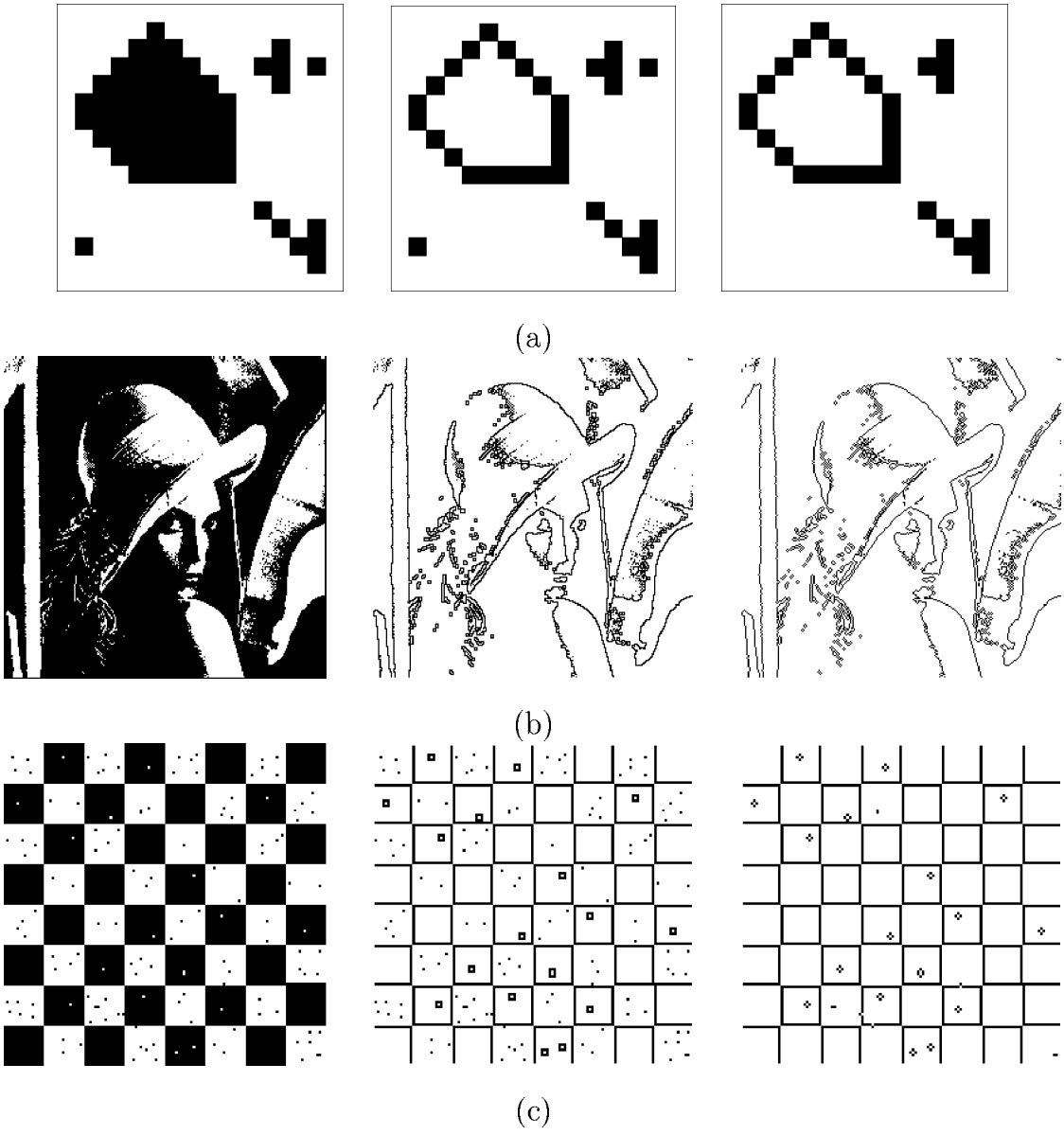


Figure 3.4: The images on the left side are the input images. The images in the middle are outputs of discrete perceptron with weight $\mathbf{w}_1$ and threshold θ_1. The images on the right side are the outputs of our modified perceptron. (a) A 16x16 binary example. (b) binary lenna image (c) A 128x128 noisy chessboard.

3.2. Design and Learning of a Multiplexed Dual Output Discrete Perceptron

In this section a new discrete perceptron model is introduced. The model forms a cascade structure and it is capable of realizing an arbitrary classification task designed by a constructive learning algorithm [3]. The main idea is to copy a discrete perceptron neuron's output to have a complementary dual ouptut for the neuron, then to select, by using a multiplexer, the true output which might be 0 or 1 depending on the given input. Hence, the problem of realization of the desired output is transformed into the realization of selector signal of the multiplexer. In the next step, the selector signal is taken as the desired output signal for the remaining part of the network. The repeated applications of the procedure renders the problem into a linearly separable one and eliminates the necessity of using the selector signal in the last step of the algorithm. The proposed modification on discrete perceptron brings the universality with the expense of getting just a slight complication in hardware implementation. This is shown in the thesis with an FPGA (Field Programmable Gate Array) hardware implementation of a discrete-weight version of the model.

A discrete perceptron whose output becomes 1 if the weighted sum of inputs exceeds a threshold and 0 otherwise, is known to be capable of realizing any linearly separable threshold function. A set of connection weights achieving the desired linear separation can be found by Perceptron Learning Rule (PLR) which ensures the convergence in a finite step when providing a proper learning rate. PLR can be run to find the optimal separating hyperplane which maximizes the margin to the class samples [39]. Furthermore, PLR can also be used to find the largest linearly separable subset of a given linearly nonseparable set of samples [38].

There are many attempts to extend the discrete perceptron model to realize any kind of threshold functions. Sequential Learning Algorithm (SLA) [40], as

stated in the name, consists of sequential applications of Perceptron Learning Rule (PLR) and at every step only one type of inputs (whose desired outputs are the same) are realized and those input vectors are removed from the learning set. It is proved in [40] that all samples can correctly be classified by SLA in a two-layer structure where the outputs of the first layer become linearly separable, and the second layer consists of just one neuron. However the work in [40] deals with Boolean inputs only. Another algorithm, based on SLA, is the Constructive Algorithm for Real-Valued Examples (CARVE) [41] which extends the SLA from Boolean inputs to the real-valued inputs case and it uses a convex hull method for the determination of the network weights instead of PLR. This algorithm gives a near-optimal solution since the task of finding the largest appropriate set is of NP-hard and the algorithm only finds good-sized appropriate sets.

Beside multilayer structures, cascade models are also investigated [2, 42, 43]. These models form a cascade structure where the inputs [42, 43] or the bias-inputs [2] of higher layers' neurons are fed by the outputs of the lower layers' neurons. These models propose structural and algorithmic solutions to linearly nonseparable classification problems against the ones [40, 41] which propose only algorithmic solutions.

There is still a demand for developing efficient learning algorithms for multilayer discrete perceptrons to realize arbitrary threshold functions. The novel discrete perceptron model introduced in this paper is a cascade structure accompanied by a kind of sequential learning algorithm and it is capable of realizing any given classification task. It is shown by the experiments that it is very convenient to run the algorithm in a discrete weight space so that calculation speed is increased and implementation complexity is also decreased. In the discrete case, the size of the network may increase since a suboptimal solution could be found as a consequence of the limitness of the number of possible weights. However, no increment greater than 5% has been observed in the experiments presented in Section 3.3.

The proposed modification of multiplexed dual output perceptron provides an advantage from hardware implementation viewpoint while complicating the implementation slightly as tested on a Field Programmable Gate Array (FPGA) simulation. As compared to the classical perceptron, new model adds only a few gate level logic operations which are the simplest devices to be used in FPGA.

The implementation differences between classical perceptron neuron and the proposed model are shown on FPGA.

As a method for implementation, FPGA approach which uses reprogrammable digital ICs, is chosen since the usage of FPGA for neural network implementation provides a flexibility as programmable systems along with the power and speed of parallel hardware architectures [44–50].

In Section 3.2.1., the proposed multiplexed dual output NN model is defined. Section 3.2.2. describes the proposed learning algorithm for that model. Some example problems are shown in Section 3.2.4. Section 3.4. investigates the model in terms of hardware implementation and an implementation example is given. Finally, Section 3.4.4. concludes the work presented here in Section 3.2. – 3.4.

3.2.1. Model

A function, $f : \Re^p \rightarrow \{0, 1\}$ separates the elements of a given input set, $X = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\} \subset \Re^p$ into two disjoint sets, X_f^+ and X_f^-, as

$$X_f^+ := \{\mathbf{x}^i \in \Re^p | f(\mathbf{x}^i) = 1\} \tag{3.6}$$

$$X_f^- := \{\mathbf{x}^i \in \Re^p | f(\mathbf{x}^i) = 0\} \tag{3.7}$$

For binary valued functions $f(\mathbf{x})$'s, there can always associate a function $g(\mathbf{x})$ such that $f(\mathbf{x}) = \text{stp}(g(\mathbf{x}))$ with $\text{stp}(\cdot)$ is the unit step function. Such $g(\cdot)$ functions

are called as separator. If $g(\cdot)$ is a linear function, it is said as a linear separator, so it defines a linear separation in the input space, $\Re^p$

A supervised classification task for a finite set of samples $X = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$ can be defined by a function $F(\cdot) : \Re^p \rightarrow \{0, 1\}$ which is given with its input–desired output pairs, i.e., the domain–range values:

$$S_F := \{(\mathbf{x}^i, d^i) | \mathbf{x}^i \in X, d^i \in \{0, 1\}, d^i = F(\mathbf{x}^i), i \in \{1, 2, \ldots, N\}\} \tag{3.8}$$

S_F can be decomposed into the two disjoint sets:

$$S_F^+ := \{(\mathbf{x}^i, d^i) | \mathbf{x}^i \in X_F^+ \text{ with } d^i = F(\mathbf{x}^i), i \in \{1, 2, \ldots, N\}\} \tag{3.9}$$

$$S_F^- := \{(\mathbf{x}^i, d^i) | \mathbf{x}^i \in X_F^- \text{ with } d^i = F(\mathbf{x}^i), i \in \{1, 2, \ldots, N\}\} \tag{3.10}$$

Any realized function $f(\cdot) : \Re^p \rightarrow \{0, 1\}$ partitions the above given input–desired output pairs into the following four sets. It should be mentioned that T^+ and T^- sets represent the correct classified pairs while the other two sets represent the missclassified pairs:

$$T^+ := \{(\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^+ \text{ and } \mathbf{x}^i \in X_f^+\} \tag{3.11}$$

$$T^- := \{(\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^- \text{ and } \mathbf{x}^i \in X_f^-\} \tag{3.12}$$

$$F^+ := \{(\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^+ \text{ and } \mathbf{x}^i \in X_f^-\} \tag{3.13}$$

$$F^- := \{(\mathbf{x}^i, d^i) \mid (\mathbf{x}^i, d^i) \in S_F^- \text{ and } \mathbf{x}^i \in X_f^+\} \tag{3.14}$$

Definition 3.1 (Correct separation) *For a given set S_F, if it can be found a linear separator $f(\cdot) : \Re^p \rightarrow \{0, 1\}$ yielding $F^+ \cup F^- = \emptyset$, then the set S_F is*

set to be linearly separable. Such separators are said to be (completely) correct separators.

Definition 3.2 (Semi correct separation) *A separator realized by* $f(\cdot) : \Re^p \rightarrow \{0,1\}$ *is said to be semi correct separator if it yields either* $F^+ = \emptyset$ *or* $F^- = \emptyset$

The proposed model realizes a correct separator for any given classification task of finite number of real sample vectors. The main idea behind the structure of the model is derived from the fact that the output of a perceptron is either true or false according to a desired output. To realize a given function, the model is designed to have neurons whose outputs are copied so that the copied one is the complement of the other and a selector signal is produced to choose one of them. Now, the problem is transferred from the realization of desired outputs to the realization of the selector signals for given input vectors. In the next step of the design, the selector signal is taken as the desired output of the remaining part of the network .

The proposed network is depicted in Figure 3.5.

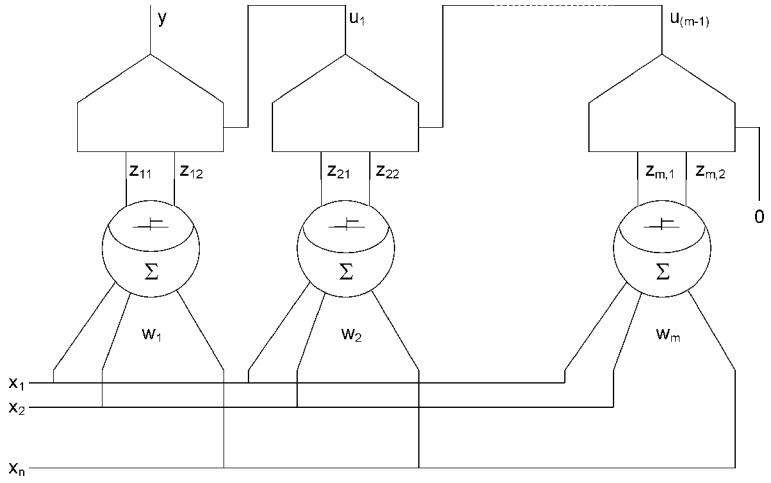


Figure 3.5: A cascade network of multiplexed dual output discrete perceptron [3]

The network of Figure 3.5 is constructed as starting from the left part of the illustration. The first attempt tries to realize S_F. Set $S_{F_0} = S_F$ and then define in a recursive way, the input–desired (selector) output pair sets as:

$$S_{F_j} = \{(\mathbf{x}^i, u_j^i) \,|\, \mathbf{x}^i \in X,\ u_j^i \in \{0,1\}, i \in \{1,2,\ldots,N\}\} \tag{3.15}$$

S_{F_j} is the set of input–desired (selector) output pairs which, indeed, defines the function $u_j = F_j(\mathbf{x})$ to be realized in the j^{th} stage. The function $u_j = F_j(\mathbf{x})$ is attempted to be realized by a selector network $y_j = f_j(\mathbf{x})$ which, in fact, partitions the S_{F_j} into the 4 sets again:

$$T_j^+ := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^+ \text{ and } \mathbf{x}^i \in X_{f_j}^+\} \tag{3.16}$$

$$T_j^- := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^- \text{ and } \mathbf{x}^i \in X_{f_j}^-\} \tag{3.17}$$

$$F_j^+ := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^+ \text{ and } \mathbf{x}^i \in X_{f_j}^-\} \tag{3.18}$$

$$F_j^- := \{(\mathbf{x}^i, u_j^i) \mid (\mathbf{x}^i, u_j^i) \in S_{F_j}^- \text{ and } \mathbf{x}^i \in X_{f_j}^+\} \tag{3.19}$$

The realization of desired function $d = F(\mathbf{x})$ by the networks's input–output function $y = f(x)$ follows from the below derivation:

$$y = \bar{u}_1 \cdot z_{11} + u_1 \cdot z_{12} \tag{3.20}$$

$$z_{11} = \text{stp}\left(\mathbf{w}_1^{\text{T}} \cdot \mathbf{x}\right) \tag{3.21}$$

$$z_{12} = \text{stp}\left(-\mathbf{w}_1^{\text{T}} \cdot \mathbf{x}\right) \tag{3.22}$$

$$u_j = \bar{u}_{(j+1)} \cdot z_{j1} + u_{(j+1)} \cdot z_{j2} \tag{3.23}$$

$$z_{j1} = \text{stp}\left(\mathbf{w}_j^{\text{T}} \cdot \mathbf{x}\right) \tag{3.24}$$

$$z_{j2} = \text{stp}\left(-\mathbf{w}_j^{\text{T}} \cdot \mathbf{x}\right) \tag{3.25}$$

$$u_m = 0 \tag{3.26}$$

(3.20) and (3.23) are realized by the multiplexers shown in Figure 3.5. Herein, m stands for the last stage.

Calculation of the selector signal u_j's is very straightforward. Selector signal is defined as '0' when the output of the neuron, z_{j1}, is equal to the desired output, and it is defined as '1' otherwise. Therefore, the selector signal is just a logical ex-or operation of actual output of the neuron,z_{j1}, and the desired output, d, as shown in (3.27) – (3.28).

$$u_1 = d \otimes z_{11} \tag{3.27}$$

$$u_j = u_{j-1} \otimes z_{j1} \tag{3.28}$$

where the symbol $\otimes$ represents logical ex-or operator, d could also be notated as u_0, and u_j is the desired value for the $j{+}1^{th}$ stage of the network.

Along the design procedure, not only the connection weights but also the structure of the model is learned.

Property 3.1 *From one stage to the next, training set changes as follows;*

- *if* $(\mathbf{x}^i, u_j^i) \in S_{Fj}^+$ *and* $\mathbf{x}^i \in T_j^+$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^-$; *we can say that realized '1's are changed to '0'.*

- *if* $(\mathbf{x}^i, u_j^i) \in S_j^-$ *and* $\mathbf{x}^i \in T_j^-$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^-$; *we can say that realized '0's remain '0'.*

- *if* $(\mathbf{x}^i, u_j^i) \in S_j^+$ *and* $\mathbf{x}^i \in F_j^+$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^+$; *we can say that non realized '1's remain '1'.*

- *if* $(\mathbf{x}^i, u_j^i) \in S_j^-$ *and* $\mathbf{x}^i \in F_j^-$, *then* $(\mathbf{x}^i, u_{j+1}^i) \in S_{F_{j+1}}^+$; *we can say that non realized '0's are changed to '1'.*

It is obvious from (3.8) – (3.14), (3.23). □

Since the structure of the network is built up by the procedure given above, the weights are the only parameters to be learned. The learning algorithm presented in the next section is employed to calculate weights of each layer of the network .

3.2.2. Algorithm

In the model, the number of layers of the network, m, is left undefined because it is not known at the beginning. It is determined along the learning process defined by the algorithm. A pseudocode for the algorithm is given in Table 3.3.

The proposed model and the accompanying algorithm assure that any given classification problem is solved by a finite number of layers. Convergence properties of the algorithm is given and proven in Section 3.2.3..

The critical stage of the algorithm is Step 2 where the weights of neurons are determined by using PLR. When the problem set is linearly separable, it is well known that PLR is capable of finding the solution very fast and algorithm stops at Step 2. For linearly nonseparable sets, it is known that PLR can provide very valuable information about the given sets. For instance, it can find input vectors violating the linear separability. Thus it can find the largest linearly separable subset of a given set [38]. On the other hand, the pocket algorithm [51], which uses PLR, can find weights providing minimum output error for linearly nonseparable problems.

Table 3.3: The learning algorithm for a cascade network of multiplexed dual output discrete perceptron.

1	a	$j = 1$
	b	d^i's are desired outputs. Set $u_0^i = d^i \ \forall i$
2	a	If the set is linearly separable, then the neuron of the j^{th} layer realizes the desired linear separation so **Stop here.**
	b	If the set is linearly nonseparable, a semi correct separation with a minimal output error could be find by CPA, and $\mathbf{w}_j$ is obtained.
3	Calculate $z_{j1} = f\left(\mathbf{w}_j^{\mathrm{T}} \cdot \mathbf{x}\right)$ $z_{j2} = f\left(-\mathbf{w}_j^{\mathrm{T}} \cdot \mathbf{x}\right)$	
4	Define u_j^i as $u_j^i = z_{j1}^i \otimes u_{j-1}^i$	
5	Take u_j^i as the desired output for the remaining part of the network	
6	Increase j by 1	
7	Go to step 2 for the realization of u_j^i	

A heuristic approach which is based on consecutive applications of Perceptron Learning Rule (PLR) for output error minimization is used in the design of the proposed cascade network of multiplexed dual output discrete perceptrons. The algorithm used to find the minimum output error for each layer might be called as Constrained Pocket Algorithm (CPA) since it is a pocket algorithm, and some constraints are added on it to ensure the convergence of the whole learning process. The weights giving minimum output error under the constraint that only one type of inputs (whose desired ouputs are the same) exist at the one side of the hyperplane are saved in the pocket.

Moreover, it should be noted that the algorithm is very convenient to add some other constraints. To get control of possible misclassifications due to the quantization of learned weights in an implementation and to make the implementation easier, training of the network would be better done in a discrete weight space. It is shown in [52] for perceptrons trained in discrete weight space that if the weights's depth is very large, i.e., there are many possible values for each weight, the learning behaviour of the discrete weights will be exactly same as those of a continuous weight. Furthermore, (i) the learning in the case of finite depth is possible by using a continuous precursor, (ii) in the case of binary output and on-line learning —this is exactly the case used in our algorithm— the generalization error decays superexponantially, (iii) perfect learning is obtained when N, the cardinality of the input set, is very large but finite. It seems to be the only disadvantage of the discrete weight case, the size of the network may increase since a suboptimal solution could be found because of limited weight space at some stages.

In the CPA, the error can be taken as absolute error (number of erroneous outputs) or alternatively as relative error. Some open points of the CPA are that there is no analytical stopping criteria also shared with the original Pocket algorithm and it may need too much time to find the optimal weights especially when the training set is large, which may be due to the constraint.

3.2.3. Convergence of the algorithm

The convergence of the algorithm will be proved based on a complexity definition such that as the algorithm runs the complexity decreases or remains the same and in a finite number of steps the complexity becomes zero which corresponds to zero misclassification error. The definition of the complexity and a proof for the convergence of the algorithm are given below.

For a given set, S, consisting of input-desired output pairs, a semi correct separation which minimizes the output error under the constraint can always

exist and can be found by CPA with repeated trials. This is also true for the training sets S_{F_j} constructed for the subnetworks used for the realization of selector signals. The complexity of any such set of inputs – desired outputs which is defined below is a measure of divergence away from the linear separability and it gives the minimum number of samples that the exclusion of them yields the linear separability.

Definition 3.3 (Complexity) *The complexity $c(S_{F_j})$ of an input-desired output set S_{F_j} is the cardinality of the set $E(S_{F_j})$ which is obtained by using (3.16) – (3.19) as in the following way:*

$$E(S_{F_j}) = \begin{cases} F_j^+, & \text{if } |F_j^+| < |T_j^-| \text{ and } F_j^- = \emptyset \\ T_j^-, & \text{if } |T_j^-| < |F_j^+| \text{ and } F_j^- = \emptyset \\ F_j^-, & \text{if } |F_j^-| < |T_j^+| \text{ and } F_j^+ = \emptyset \\ T_j^+, & \text{if } |T_j^+| < |F_j^-| \text{ and } F_j^+ = \emptyset \end{cases} \tag{3.29}$$

$$c(S_{F_j}) = |E(S_{F_j})| \tag{3.30}$$

□

The complexity defined above relies on the following observations. The separating hyperplane separates the input set into two subsets: T_j^- and F_j^+ are in one side of the hyperplane and T^+ and F^- are in the other side. Considering the constraint that either F_j^- or F_j^+ is empty, to obtain a linearly separable set from the original one, there are some possibilities depending on the sets, T_j^-, T_j^+, F_j^- and F_j^+. Assuming that F_j^+ is the empty set, the sets T_j^+ and F_j^- are on one side and T_j^- is on the other side of the hyperplane. If the elements of F_j^- are excluded from the training set, the remaining elements, T_j^+ and T_j^- are separated by the hyperplane, so the problem becomes linearly separable. The other option, which is the exclusion of the set T_j^+ results that only F_j^- and T_j^- remains in the training set. Since the desired outputs for the elements of both sets are all zero, and there

is no element with a desired output is one, the exclusion concludes that there is no need for a separation. This considerations are stated in Theorem 3.1.

Theorem 3.1 *For a semi correct separation which minimizes the output error under the exclusion of the elements of $E(S_{F_j})$, the remaining set is either linearly separable or needs no separation since all samples are of the same kind.*

Proof: *There are four cases: (i) $F_j^- = \emptyset$ and the erroneous elements $\mathbf{x}$'s which are in F_j^+ are excluded, then the elements yielding correct outputs remain only, i.e., the element $\mathbf{x}$'s are either in T_j^+ or T_j^-. (ii) $F_j^- = \emptyset$ and the correct elements $\mathbf{x}$'s which are in T_j^- are excluded, then the elements whose desired outputs are all the same remain only, i.e., $\mathbf{x} \in T_j^+$ or $\mathbf{x} \in F_j^+$. There is no need for separation. (iii) and (iv) can be proven by considering $F_j^+ = \emptyset$ and following the same approach as in the cases (i) and (ii). In all of the four possible options, it leads that remaining set is linearly separable.* □

Theorem 3.2 *The algorithm defined in Table 3.3 converges to zero output error in a finite number of steps.*

Proof: *At each stage of the algorithm which corresponds to the design of a layer of the network, a correct separation or a semi correct separation occurs. For a semi correct separation which gives a minimal output error, there are two cases :*

1. *Case 1: $F_j^- = \emptyset$*

   (a) *If $\forall \mathbf{x}^i \in T_j^+$, i.e., $\mathbf{x}^i \in X_j^+$ and $(\mathbf{x}^j, d^j) \in S_{F_j}^+$, then $\mathbf{x}^i \in S_{F_{j+1}}^-$.*

   (b) *As the worst case, assume that the same separator is used for the $(j+1)^{th}$ layer as with j^{th} layer but in the opposite direction, so $T_{j+1}^- = T_j^+$, $F_{j+1}^- = T_j^-$, $T_{j+1}^+ = F_j^+$ and $F_{j+1}^+ = \emptyset$. In this case the complexity does not change since the complexities, $c(S_{F_j}) = min\{|F_j^+|, |T_j^-|\}$ and $c(S_{F_{j+1}}) = min\{|T_{j+1}^+|, |F_{j+1}^-|\}$, are equal to each other.*

(c) *At each j^{th} step, the algorithm finds either a correct separation (see 2.a of Table 3.3) or semi correct separation with minimum output error (see 2.b of Table 3.3). In either case, the vectors in T_j^+ defines a convex hull whose intersection with the convex hull of the vectors in $F_j^+ \cup T_j^-$ is empty and the vertices of the convex hull of $F_j^+ \cup T_j^-$ that are the closest vertices to the convex hull of T_j^+ are necessarily in T_j^-. Because, any vertex of F_j^+ with the minimum distance to T_j^+ could be included by T_j^+ without violating the linear separability of T_j^+ and T_j^-. This can be seen as follows; (i) the extension of a convex set via including a point from its outside, possesses that point as a vertex, (ii) the convex hull of a set constructed by excluding a vertex of a considered set does not include that excluded vertex as one of its points and (iii) two convex hulls are linearly separable iff their intersection is empty. The vertices of the convex hull of $F_j^+ \cup T_j^-$ which are the closest to the convex hull of T_j^+ become at the $(j+1)^{th}$ stage the vertices of $T_{j+1}^+ \cup F_{j+1}^-$ which are now in F_{j+1}^-.*

(d) *From (1c), considering $S_{F_{j+1}}$ in the new separation for the $(j+1)^{th}$ layer, $|F_{j+1}^-| < |T_j^-|$ and $|T_{j+1}^+| = |F_j^+|$. Therefore the complexity from layer j to layer $j+1$ decreases or remains constant: $c(S_{F_{j+1}}) = \min\{|F_{j+1}^-|, |T_{j+1}^+|\} \leq c(S_{F_j}) = \min\{|F_j^+|, |T_j^-|\}$.*

(e) *Even for the cases of complexity remaining constant, the total cardinality $|T_j^+| + |F_j^-|$ decreases at each step which can cause a decrease of a certain amount in the complexity after a certain number of steps. When F_{j+1}^+ is empty which can be analysed (see Case 2) in a similar way to the analysis of F_j^- is empty, the complexity decreases or remains constant also.*

(f) *If at each stage F_j^- is empty then it can be concluded that the complexity decreases to zero within a finite number of steps.*

2. *Case 2: $F_j^+ = \emptyset$*

The analysis of this case is the same with the Case 1 but with;

(a) *Same separator function with the same direction is used from* j^{th} *layer to the* $j + 1^{th}$ *layer instead of opposite direction in (b) of Case 1,*

(b) *From* j^{th} *layer to* $(j + 1)^{th}$ *layer, sets change as follows* $T_{j+1}^{+} = F_{j}^{-}$, $F_{j+1}^{-} = T_{j}^{+}$, $T_{j+1}^{-} = T_{j}^{-}$ *and* $F_{j+1}^{+} = \emptyset$,

(c) *The complexities as* $c(S_{F_j}) = min\{|T_j^+|, |F_j^-|\}$ *and* $c(S_{F_{j+1}}) = min\{|F_{j+1}^-|, |T_j^+|\}$.

Combining two cases, the complexity is seen to decrease to zero in a finite number of steps. □

To demonstrate how the algorithm works, next section gives some example applications of the algorithm and experimental results.

3.2.4. Experimental Results

In this section, experimental results of the work are presented.

The proposed algorithm uses pocket algorithm at each step to find a largest separable subset. Because of the stochastic nature of the pocket algorithm, and also because of the constraints added to the original pocket algorithm, sometimes it is not possible to find the hyperplane which gives minimum constrained output error in finite number of steps. However, it may find the hyperplane which gives a minimal constrained output error. On the other hand, two different hyperplanes giving the same error may lead to the networks of different sizes. So, multiple trials for each specific classification problem were made in the experiments. The resulting networks for the considered problems are compared to the networks obtained with the other techniques available in the literature.

For the sake of clarity and a better understanding of the algorithm, synthetic example problems illustrated in Figure 3.6 and Figure 3.7 are constructed as the 4x5 grid where the desired output values for input samples are assigned randomly.

In each subfigure of Figure 3.6 and Figure 3.7, 'x's and 'o's represent input vectors whose desired outputs are '1' and '0', respectively. Lines in the figures correspond to the separating hyperplanes constructed by $\mathbf{w}_j$ and arrows show the positive sides of those hyperplanes. Therefore, for all vectors belonging to the positive side of an hyperplane, the output of the network realized a j^{th} step will be '1'. Please note that at every subfigure, either side of the hyperplane contains only one type of vectors, i.e., 'x' or 'o'.

Error logs of the algorithm for two examples depicted in Figure 3.6 and Figure 3.7 are listed in Table 3.4 and Table 3.5 respectively.

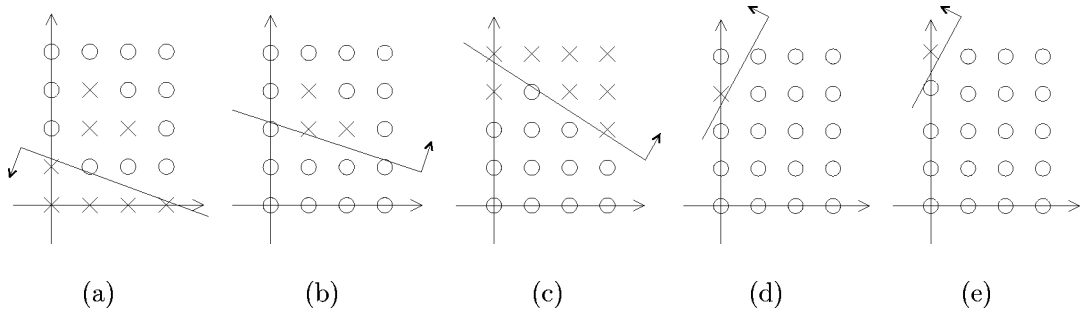


Figure 3.6: An example for learning steps of the algorithm for the cascaded network of multiplexed dual output discrete perceptron. Here 'x' denotes desired output value of '1' and 'o' denotes '0'. Line corresponds to the separating hyperplane and arrow shows the positive side of the hyperplane. Therefore, for all vectors belong to the positive side of the hyperplane, output will be '1'.

3.2.4.1. Parity

The parity problem is a common test to measure the performance of NN classificaiton algorithms. The problem is to classify binary input vectors into two sets in terms of the number of elements with value one of the input vector, i.e. the number is odd or even.

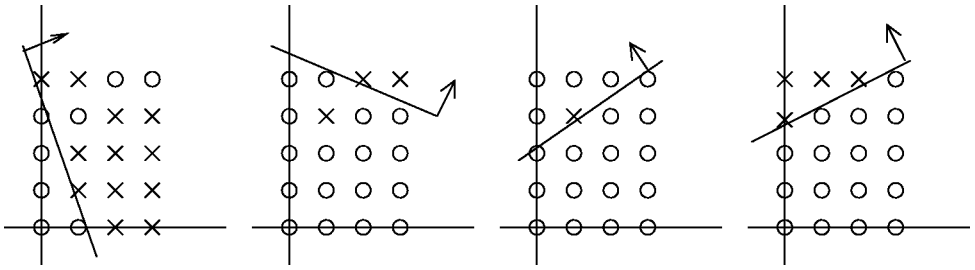


Figure 3.7: Another example for learning steps of the algorithm for the cascaded network of multiplexed dual output discrete perceptron.

Table 3.4: Complexity at each layer of the algorithm for the example shown in Figure 3.6

	$s(T_0)$	$s(F_0)$	$s(T_1)$	$s(F_1)$	Complexity
1	12	0	5	3	3
2	9	8	3	0	3
3	12	0	7	1	1
4	18	1	1	0	1
5	19	0	1	0	0

Table 3.5: Complexity at each layer of the algorithm for the first example shown in Figure 3.7

	$s(T_0)$	$s(F_0)$	$s(T_1)$	$s(F_1)$	Complexity
1	5	3	12	0	3
2	17	0	2	1	1
3	15	4	1	0	1
4	18	0	4	0	0

The parity problem requires, when using MLP, as many hidden neurons in the first layer [53]. Consequently, in case of four dimensional input space, network should contain at least five neurons.

Table 3.6: Random Boolean functions network sizes

n	This model	CARVE [41]	Sequential [40]	Tiling [54]	Regular [55]
4	2.44 ± 0.50	2.40 ± 0.69		3.9	
5	4.44 ± 1.00	3.73 ± 0.58			
6	8.40 ± 1.27	5.88 ± 0.67	7.28 ± 0.82	16.99	15.8±2.2
7	19.33± 2.08	9.47 ± 0.74			
8	35.75± 0.95	16.23 ± 0.86	18.3 ± 0.69	56.98	

The experiments with the proposed model and the algorithm shows that only n neurons are enough to solve the problem. However, when the input dimension increases, the training set becomes larger and to find the optimal hyperplane for each layer becomes difficult. Therefore, training time will increase otherwise suboptimal hyperplanes at some layers leads to grow in network size. In the experiments, for $n = 1 \ldots 5$ network size with n neuron are obtained fastly for all trials. However, for $n = 6 \ldots 8$ either long training periods are needed or sometimes network is constructed with neurons more than n. This is because of the inadequacey of CPA. In one layer CPA fails to find the optimal solution this affects also the subsequent layers and network size grows more than the minimum size of that model is capable to realize the given task. This CPA's inadequecy is appearent also on the other experiments.

3.2.4.2. Random Boolean Functions

A random n-bit Boolean function is a training problem consisting of all 2^n Boolean vectors. Every vector is randomly assigned to one of the classes with equal probability. For each dimension, $n = 4 \ldots 8$, 10 different test sets are produced and every test set is tested several times. The results of the experiments are given in Table 3.6.

3.2.4.3. Two Spirals

The two spirals problem [56] is a classification task which is highly nonlinearly separable. Single hidden layer networks trained by backpropagation generally fail to produce solutions to this problem and constructive algorithms have been more successful [57]. The model applied to the two spirals task generates a network solution. Some steps of the solution are given in Figure 3.8(a) through Figure 3.8(d).

The average network size obtained over 8 trials is 57.37 ∓ 2.56, with a minimum network size of 56 layers and maximum size of 62 layers.

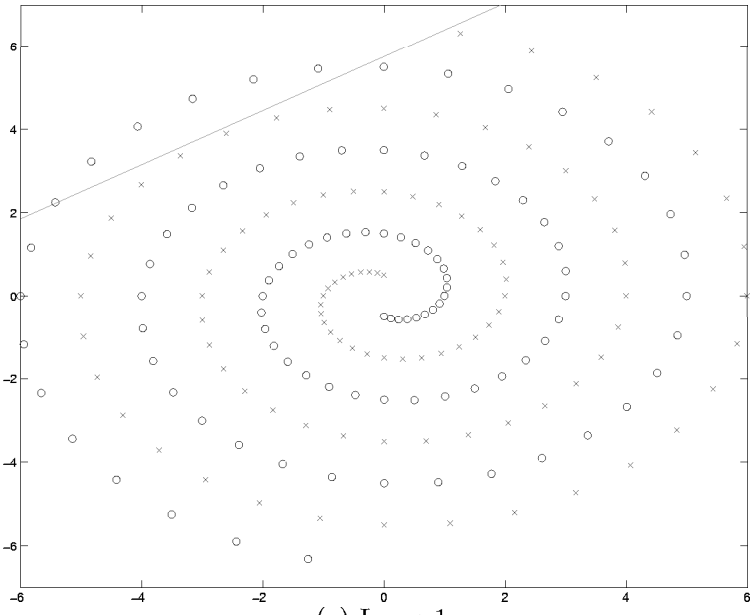
3.3. Learning in Discrete Weight Space

The learning algorithm is very convenient to add some other constraints. In CPA, the method for finding the optimal hyperplane of the algorithm, seeks for the weights which give the minimum error under prespecified constraints. So some extra constrains can be added to the algorithm, easily, so providing learning in discrete weight space with finite number of weights is achieved by considering the discreteness of the weights as a constraint.

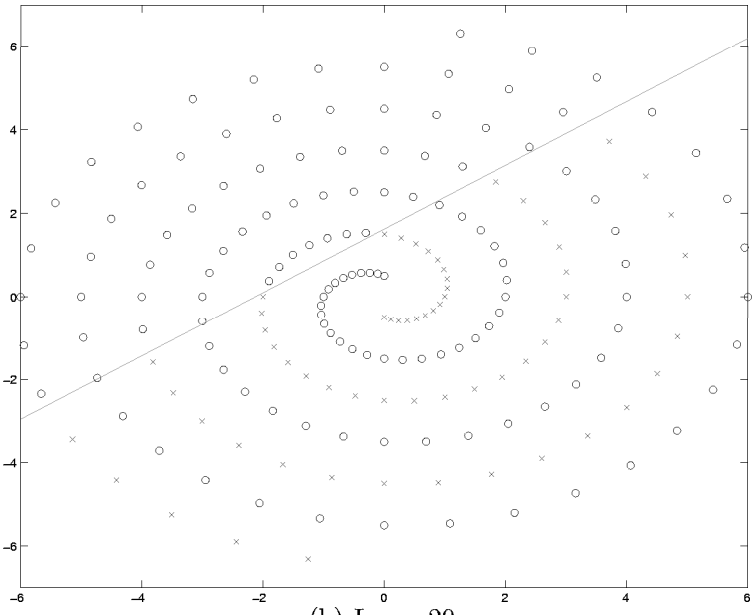
To prevent quantization effects and to make the implementation easier, training could be done in a discrete weight space. It is shown in [52] that if the weights's depth is very large, i.e., there are many possible values for each weight, the learning behaviour of the discrete weights will be exactly same as those of a continuous weight. Furthermore, (i) the learning in the case of finite depth is possible by using a continuous precursor, (ii) in the case of binary output and on-line learning —this is exactly the case used in our algorithm— the generalization error decays superexponantially, (iii) perfect learning is obtained when N, the cardinality of the input set, is very large but finite. It seems that the only

disadvantage of the discrete weight case, the size of the network may increase since a suboptimal solution could be find because of limited weight space at some stages.

Considering the convergence of the algorithm, the added constraint should not obstruct PLR to find the solution for linear separable sets. We can monitor by

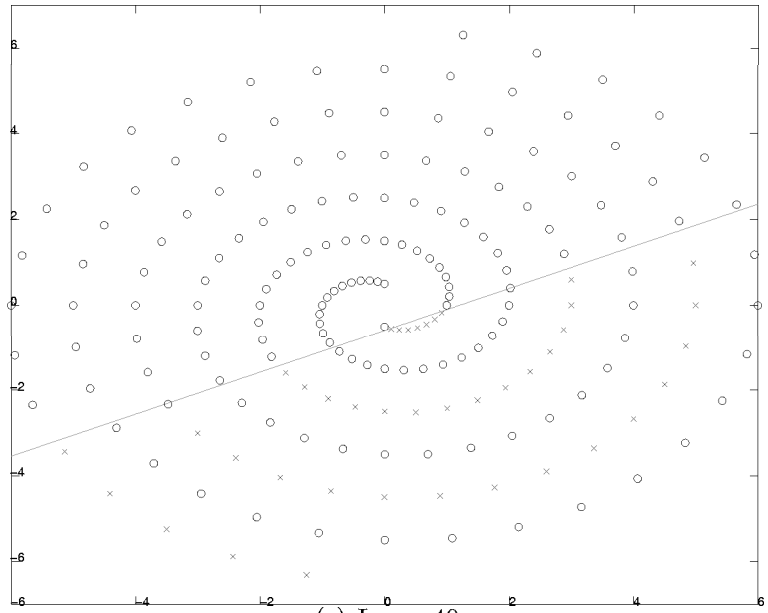


(a) Layer 1

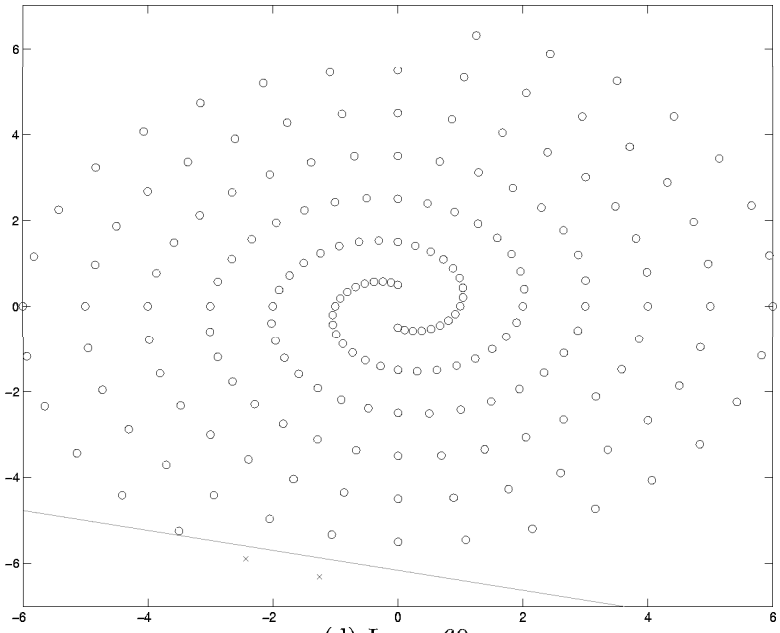


(b) Layer 20

Figure 3.8: Some example epochs from two-spirals solution. (a) and (b).



(c) Layer 40



(d) Layer 60

Figure 3.8: Some example epochs from two-spirals solution (c) and (d) (cont.)

observing the network output error for constraint and unconstraint algorithms together, in case of such a situation exists.

To show the results of learning in discrete weight space, training is run for random Boolean functions, parity functions and two-spirals function in a discrete weight

Table 3.7: Random Boolean functions network sizes in a discrete weight space learning. n is the input space dimension.

n	Reel numbers	16bit Integers
4	2.44 ± 0.50	2.60 ± 0.70
5	4.44 ± 1.00	4.60 ± 1.17
6	8.40 ± 1.27	8.70 ± 1.70
7	19.33 ± 2.08	19.80 ± 2.10

space. As can be seen from the results that the produced network sizes are similar without any considerable difference where even some increase is already expected.

3.3.1. Experimental Results

3.3.1.1. Parity

The experiments of the proposed model in a discrete weight space are done for parity problems in four, five and six dimensional input spaces. They results that only n neurons are enough to solve the problem with discrete weights too, similar to examples mentioned in Section 3.2.4.1.. The discrete weight space is taken as 16-bit signed integers.

3.3.1.2. Random Boolean Functions

To show the results of learning in discrete weight space, training is run for random boolean functions in a discrete weight space and results are listed in Table 3.7. The same functions as the ones used in the examples in Section 3.2.4.2. are chosen and weight space is taken as 16-bit signed integers. As be seen from the results that the produced network sizes are similar without any considerable difference where some increase is expected.

3.3.1.3. Two Spirals Function

To show the results of learning in discrete weight space, training is also run for two-spirals function in a discrete weight space and results that the average network size obtained over 6 trials is 60.17 ∓ 3.81. Weight space is taken as 16-bit fixed point real values with 12 bit fraction length.

3.4. Hardware Implementation

A hardware implementation of the proposed model is presented here. For the implementation, FPGA (Field Programmable Gate Arrays), as reprogrammable digital ICs, is selected since the usage of the FPGA (Field Programmable Gate Array) for neural network implementation provides flexibility of programmable systems and has an increasing trend in the neural networks literature [44–47].

FPGAs are ICs containing programmable logic components and programmable interconnects including tens of thousands up to few millions gates, dedicated memory blocks, multiplier circuits, PLLs etc. Moreover, new models of FPGA chips are produced with microprocessors embedded in it. The programmability of reconfigurable FPGAs yields the availability of fast special purpose hardware for wide applications, so FPGA based Artificial Neural Networks (ANN) is now becoming a focus of ANN reserch [50]. The hardware implementation of NNs is superior comparing with software approach because FPGA supports the advantage of parallel processes that is the key feature of NN structures.

A lot of powerful design, programming, syntheses and simulation tools provided by FPGA manufacturing and software development companies along with reprogramming capability of the chips brings short design cycle reduced design and development phase.

In this section, while the implementation of the proposed model in FPGA is done, it is also compared to the classical perceptron neuron in terms of hardware implementation, i.e., the resource usage. The proposed modification of multiplexed dual output perceptron provides an advantage from hardware implementation viewpoint while complicating the implemantation slightly. Because, new implementation trend of NNs goes to FPGA in which gates are the simplest devices to use and new model adds only a few gate level logic operations to the classical perceptron.

The implementation differences between classical perceptron neuron and the proposed model are shown on FPGA (Field Programmable Gate Arrays).

FPGA approach in implementation as a method which uses reprogrammable digital ICs, is chosen since the usage of the FPGA for neural network implementation provides the flexibility of programmable systems along with power and speed of parallel hardware architectures so it has an increasing trend in the neural networks literature [44–50].

3.4.1. System Architecture

By using of the FPGA, hardware implementation of ANN's could be done in two main architectures. The first one is the fully parallel architecture so that the number of multipliers and the number of full adders per neuron are equal to the number of inputs of the neuron. The main advantage of this architecture is to be fast as it directly realizes the parallelism of the NNs. However, this approach increases the usage of the resource in FPGA and may lead to select bigger and more expensive chip models especially for huge NNs with higher input space dimensions. The other architecture saves the resource usage but works a bit slower. In this architecture only one multiplier and one accumulator are used per neuron and at each step, one input is multiplied by the corresponding weight and added to the accumulator. The calculation of the output is done in n steps, where n is the number of inputs of the neuron. Please note that, the second

architecture has still the advantages of parallel processing in the network level, i.e. all neurons have their own multipliers/adders and work parallel in the neural network.

These two FPGA design architectures of NNs are depicted in Figure 3.9 and Figure 3.10 in a block scheme level.

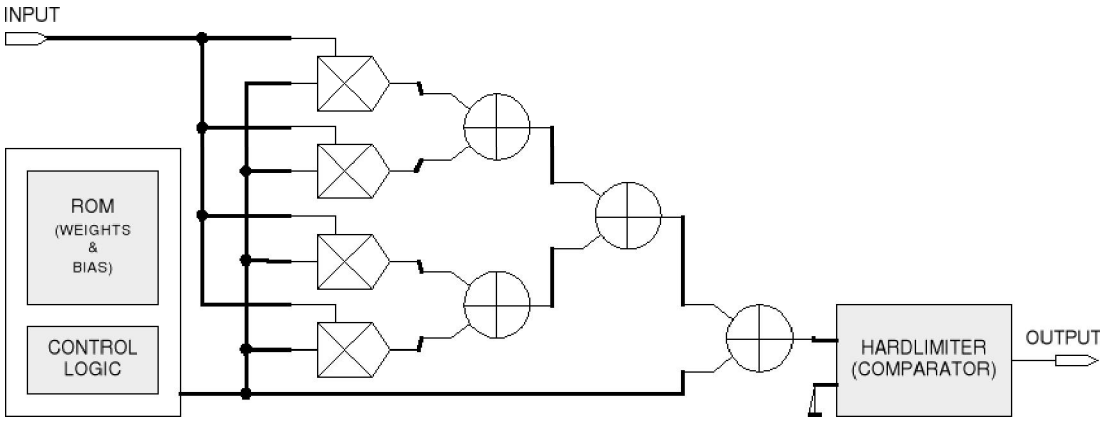


Figure 3.9: Hardware Architecture for fully parallel method.

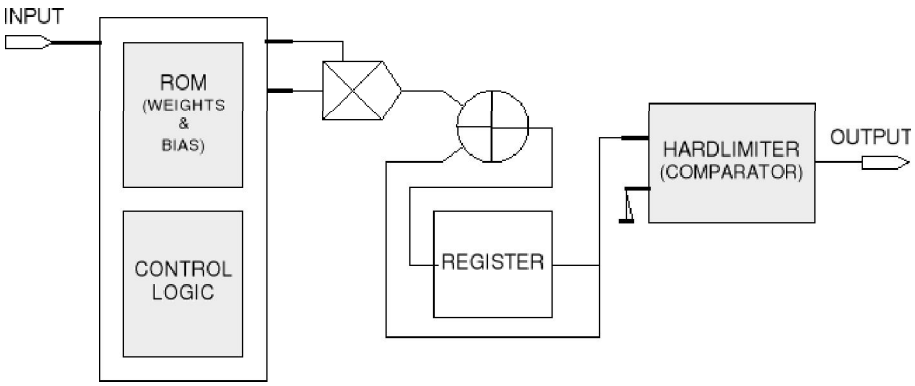


Figure 3.10: Hardware Architecture for multiply-add method.

3.4.1.1. Data Representation

To implement neural networks in FPGA, the input data and the weights of the neuron should be well handled to present them to the hardware. First of all,

since FPGA is a digital environment, the data should also be digital[1]. Then, a decision is needed in design phase of the FPGA, about the precision (number of bits) and number format (signed/unsigned integer, floating/fix point real value, etc). Here it is very obvious that, more data precision (the number of bits in representation) more resource usage in FPGA, on the contrary less quantization error for output.

In the hardware implementation of NNs data representation is an inportant design parameter. Floating point representations supply more accuracy while the speed reduces and resource usage increases [58]. Instead of fixed point or floating point representations, a special data representation method is proposed for NN implementations so that the data precision changes adaptively according to the weights's histogram. In this method, the precision is higher where the weights are very close to each other and the precision is lower where the weights are dispersed [59]. Error rate is very low in this representation system but it has a serious disadvantage so that different designs and representation sets are needed for each problem.

3.4.2. Implementing 4-bit parity problem

It is shown in Section 3.2.4.1. that using the proposed model and the algorithm, only n neurons are enough to solve the n-bit parity problem. Running of the algorithm ends up to a 4-neuron network and a set of weights for these neurons.

As explained in Section 3.4.1.1. data representation is another important aspect. Altough the input values are binary for the parity case, i.e., we can represent them as only one bit, the weights obtained by the algorithm is double precision real values. So it should be decided about the number format without causing

[1]The new generation mixed-signal FPGAs are recently introduced. These chips integrates configurable analog, large Flash memory blocks, comprehensive clock generation and management circuitry, and high performance programmable logic in a monolithic device. These chips have embedded ADC(s) and also in-system configurable analog supports for some applications. However analog functions are still very limited and in our case we have to be in the digital side for now.

error on the working of the network. The most FPGA design tools and chips supports IEEE double precision floating point number format, but in this case the complexity of the implementation increase.

On the other hand, the proposed algorithm is very suitable to work in discrete weight space, so during the learning phase it is known the effects of the number format whether it causes an error or not. The example runs given in Section 3.3.1.1. gives a set of weight vectors in 16 bit signed integer number format, and implementing arithmetic operations in integers is much easier than floating point.

In this work, the model is implemented with one multiplier and one accumulator per neuron. The inputs enter the neuron parallel and multiplied serially by their corresponding weights stored in a ROM. The results of multiplication are saved in an accumulator. At the otput, there is a comparator functioning as hardlimiting. To organize the data flow, timing and serial processes, a control unit is added to the neuron design. Digital system architecture is modeled using Very High Speed Integrated Circuits Hardware Description Language (VHDL) and is implemented in FPGA chip. The top-level schematic of the implemantation is depicted in Figure 3.11 and implementaition details are shown in following Figures (See Fig. 3.12 – Fig. 3.15).

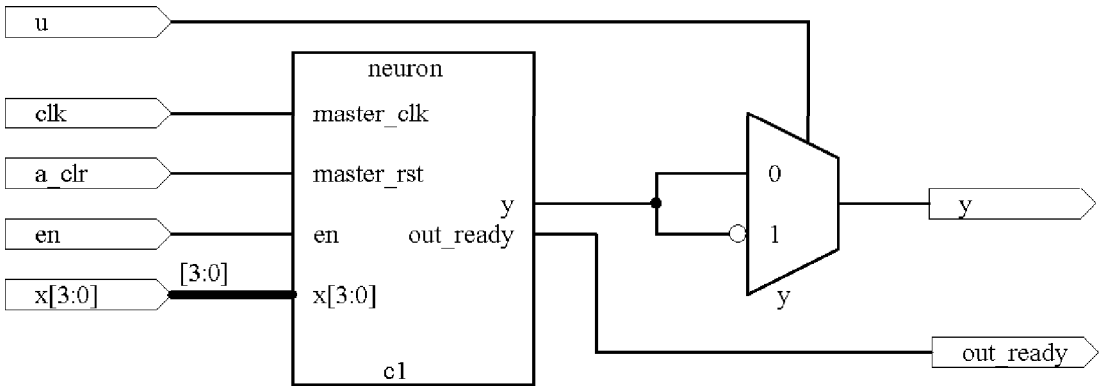


Figure 3.11: Proposed model

Details of the main functional blocks of the design, namely counter, controller, multiplexer, multiplier, ROM, accumulator and comparator, are given in the following sections.

3.4.2.1. Counter

The counter function could be considered in the controller block in spite of that it is shown out of it since the counter is used to address the weights in the ROM and load the appropriate input vector to the multiplier. At every clock, an input and corresponding weight is fed to the multiplier according to the counter output.

The counter is designed as a generic counter whose number of bits and modulo value can be set as a parameter in design phase. Asynchron clear input and clock enable input are also defined. The VHDL design file is listed in Table 3.8 and corresponding RTL (Register Transfer Level) [2] schematic view is shown in Figure 3.13.

3.4.2.2. Controller

The controller block coordinates all other parts; input signals, the counter, ROM, multiplier, multiplexer, accumulator, comparator and output signals.

The neuron unit starts working when the enable signal is activated. After it is enabled, controller loads x_1 and w_1, multiplies them and load to the accumulator, then enabling the counter it continues with x_2, x_3 ... x_n. At the end, when the output ready, it enables the output and asserts the `output_ready` signal. All

[2]In integrated circuit design, RTL description is a way of describing the operation of a synchronous digital circuit. In RTL design, a circuit's behavior is defined in terms of the flow of signals (or transfer of data) between hardware registers, and the logical operations performed on those signals.

Register transfer level abstraction is used in hardware description languages (HDLs) like Verilog and VHDL to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived.

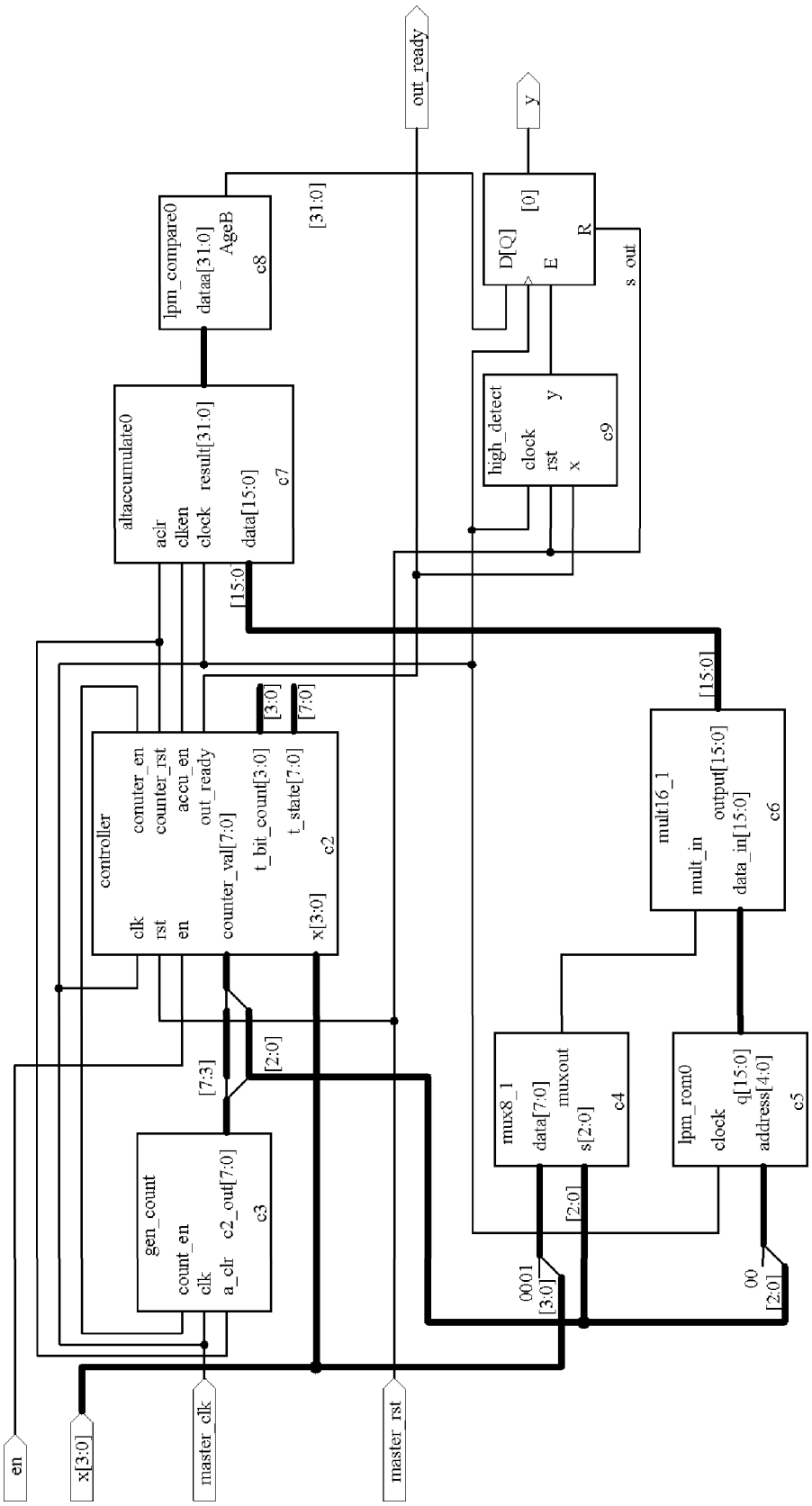


Figure 3.12: Internal structure of 'neuron' block in Figure 3.11. In fact, this is an implementation of discrete perceptron neuron.

Table 3.8: VHDL design file of the generic counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- count value cannot be 0 (zero)
-- if it is needed that counter working without restarting
-- take count as 2^(width+1) - 1
entity gen_count is
    generic (width : integer := 8; count : integer := 255 );
    Port (
        clk : in std_logic;
        a_clr : in std_logic;
        c2_out : out std_logic_vector (width-1 downto 0);
        count_en: in std_logic
    );
end gen_count;

architecture arch1 of gen_count is

signal out_buf : std_logic_vector(width-1 downto 0);

begin

c2_out <= out_buf;

process(clk, a_clr, count_en)
    begin
        if (a_clr = '1') then
            out_buf <= (others => '0');
        elsif (count_en = '1' and clk'event and clk = '1') then
            if (out_buf = count) then
                out_buf <= (others => '0');
            else
                out_buf <= out_buf + 1;
            end if;
        end if;
end process;

end arch1;
```

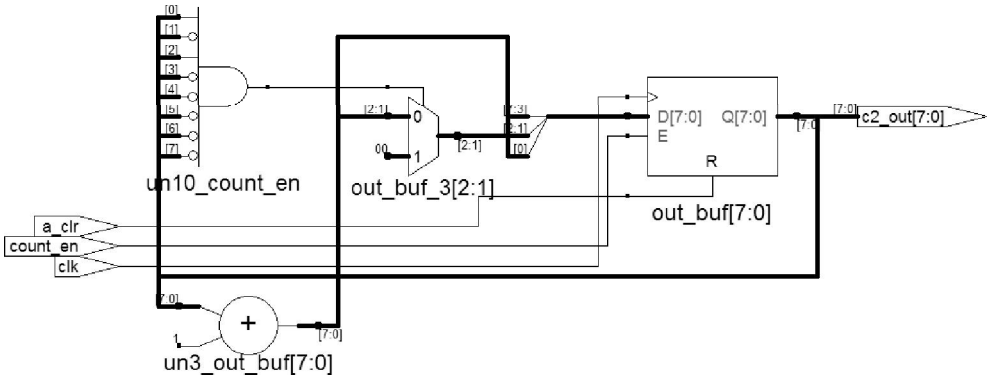


Figure 3.13: RTL view of an FPGA Implementation of mode 5 counter.

this process is done by using a state machine along with some simple logic. The RTL view of the design is shown in Figure 3.14 and the state diagram of the state machine used in controller is depicted in Figure 3.15.

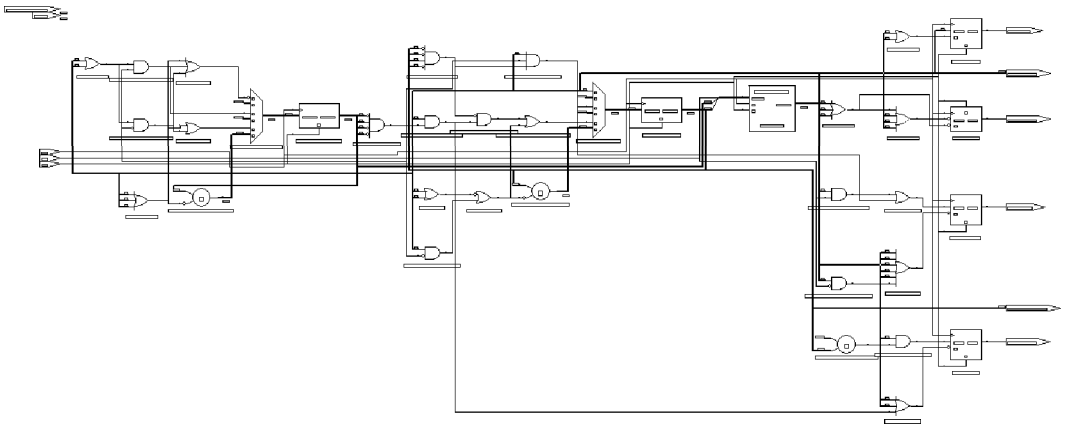


Figure 3.14: RTL view of an FPGA Implementation of the controller.

3.4.2.3. Other components

The multiplier for the 4-bit parity implementation is relatively simple since the input x is only one bit. Multiplication result is either all zero or equal to the weight so the operation could be realized by a multiplexer only.

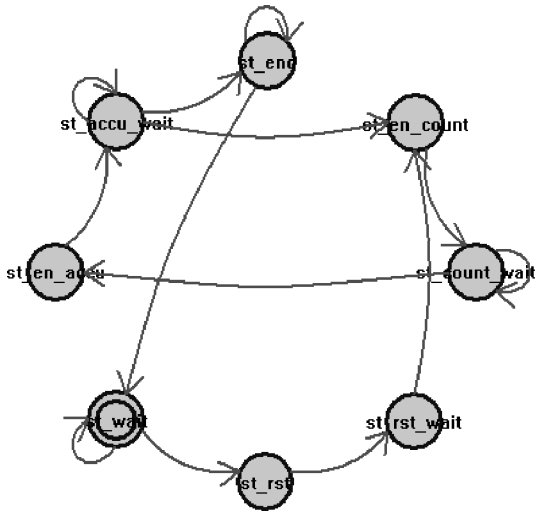


Figure 3.15: State diagram of the controller.

ROM and comparator blocks are used as core blocks supplied in FPGA IDE by the vendor. Here ROM is selected with 16 bit data bus, 32 bit depth, i.e., 5 bits address bus. Weights learned by the algorithm are recorded in the ROM by using a memory initializatin file. The mif file used for the first neuron of the four neurons realizing the parity problem is listed in Table 3.9.

Weight values are represented and recorded as 16 bit signed integers. Corresponding weight vector is given in Eqn 3.31–3.32.

$$\mathbf{w}_1 = [355 \quad -253 \quad 16 \quad -221] \tag{3.31}$$

$$\mathrm{b}_1 = [471] \tag{3.32}$$

Comparator is also used as core block. It is preferred to make the design some more general. In fact, for the 4-bit parity problem, the sign bit of the accumulator is also be used as the output value since the output nonlinearity is just a sign function. More generally speaking, output nonlinearities such as sigmoid functions are implemented as look-up-tables.

Table 3.9: Weights recorded in ROM for the first neuron of four neurons parity network.

```
-- Memory Initialization File (.mif)

WIDTH=16;
DEPTH=32;

ADDRESS_RADIX=UNS;
DATA_RADIX=UNS;

CONTENT BEGIN
        0   :   355;
        1   :   65283;
        2   :   16;
        3   :   65315;
        4   :   471;
        [5..31]  :   0;
END;
```

3.4.2.4. Verification of the implementation and Results

The designed and implemented model is verified by using as FPGA simulation tool [60].

In verification phase all internal signal timings are observed and verified. The internal hardware signalling is shown in Figure 3.16.

After the input data are fed to the network and the enable signal is asserted, network computes the output and when ready, it loads output data to the output pins and asserts `out_ready` signal. As seen from the timing diagrams the parity network gives correct outputs, for instance, '1' for '0001', '0' for '0110' and so on. For a summary, functional input - output signal relations are shown in Figure 3.17.

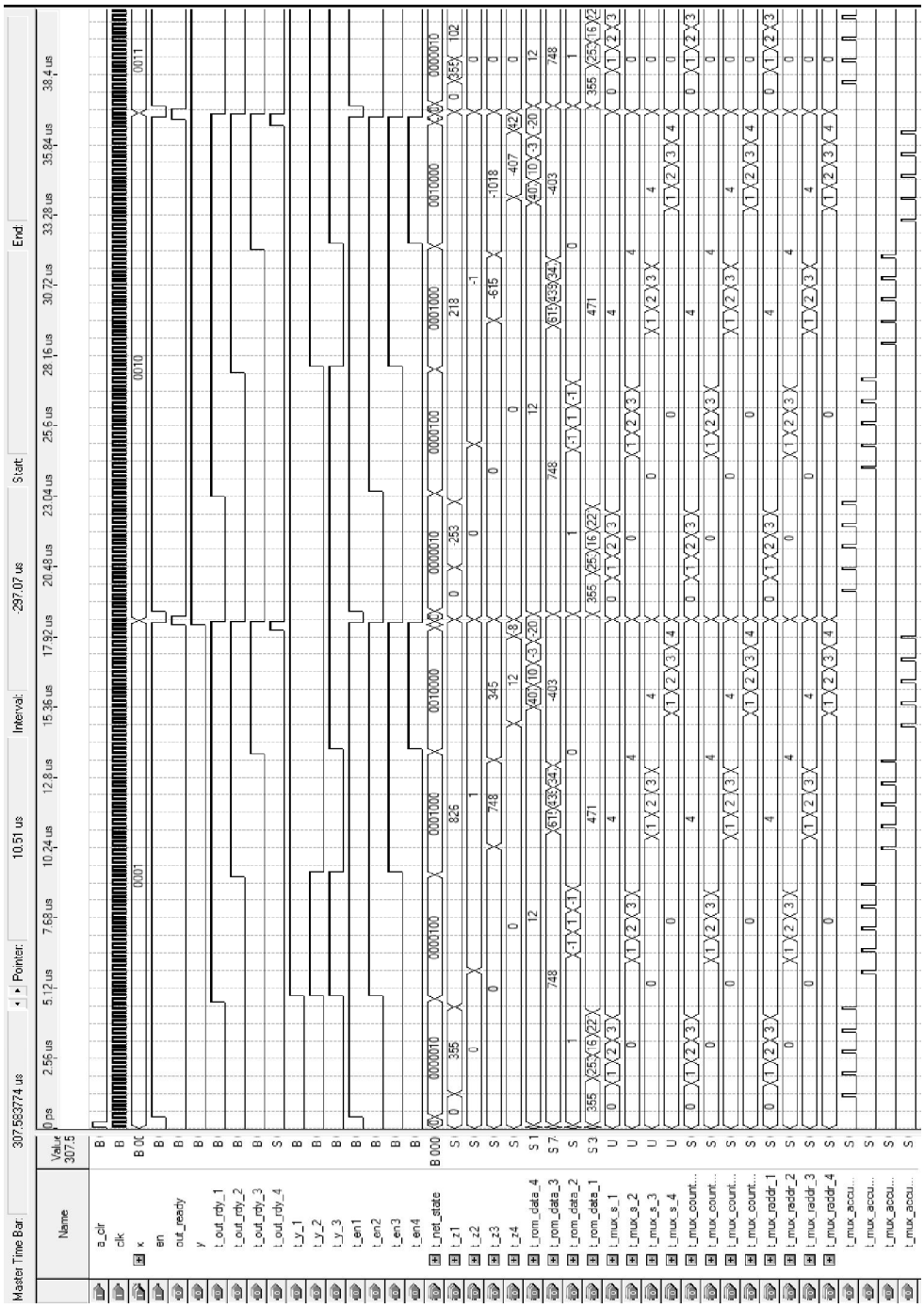


Figure 3.16: 4-bit parity FPGA Implementation output timing signals.

3.4.3. A Comparison

In the previous paragraph, the hardware implementation of the proposed NN model for 4-bit parity problem is explained and as seen from the Figure 3.11 that

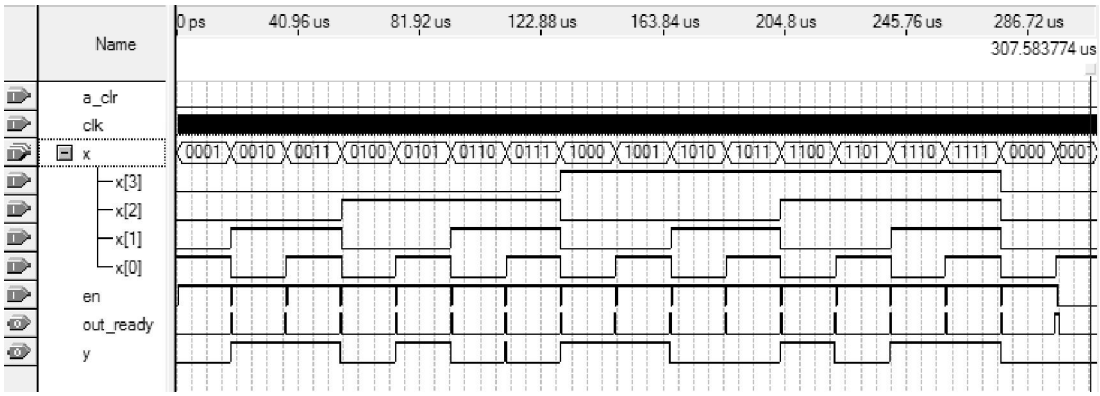


Figure 3.17: 4-bit parity FPGA Implementation output timing signals.

the proposed model's neuron is very similar to the perceptron neuron. When the proposed modifications on the perceptron neuron are considered from the hardware implementation viewpoint, it can be concluded that these modifications do not result any significant increase neither in complexity nor in resource usage. A comparison in terms of resource usage between perceptron neuron and proposed model neuron is given in Table 3.10 in terms of ALUTs [3], registers, total pins, memory bits and maximum clock frequency.

Table 3.10: The resource usage of neurons in FPGA implementation.

	Perceptron Neuron	Proposed Neuron
ALUTs	99	100
Registers	63	63
Total Pins	9	10
Memory Bits	128	128
Max Clock Freq	213.45 MHz	207.21 MHz

As a result, it can be seen from the resource usages of two different neuron models that the increase in resurce usage is less than 0.7% per neuron, including the pin

[3]ALUT, Adaptive Look-Up Table, is the cell in FPGA chip that is used as the output of logic synthesis. A single ALUT contains a register and a combinational pair.

number. Please note that, the number of pins used will be remain constant independent from number of neurons in the network. Considering the max clock frequency values, the max frequencies listed in Table 3.10 are obtained without any special optimization or setting any constraint set. By applying some appropriate constraints on FPGA compiler software, one may force it to synthesize more efficiently, and fit into chip to work faster. In many cases, setting a constraint which declares a need for a max clock frequency even impossbily higher, results higher max clock frequencies higher than the ferquencies obtained without any constraint.

3.4.4. Discussion

A new cascaded NN model and a learning algorithm associated with it is proposed for linearly nonseparable classification problems. Any given function from the set of R^n to the set of $\{0, 1\}$ can be realized by the model in a finite number of steps, resulting in a network of finite number of neurons. Because of the heuristic nature of the algorithm, the minimum network size is not always guaranteed.

The algorithm seeks for the weights which give the minimum error under prespecified constraints. There is a possibility of adding extra constraints to the algorithm, so providing learning in discrete weight space with finite number of weights is achieved by considering the discreteness of the weights as a constraint.

The proposed modification on discrete perceptron brings the universality with the expense of getting just a slight complication in hardware implementation. This is shown by comparing FPGA implementation of the model and discrete perceptron where the increase in the FPGA resource usage is only about 0.3%.

The proposed model for two-class classification problems can be extended to multiple class separation problems by assigning one output neuron for each class.

4. DYNAMICAL PWC AND PWL NEURAL NETWORKS

Cellular Neural Network (CNN) is a dynamical model with a PWL output function. CNN has many applications especially in image processing. However design or training of CNNs is a great deal and there is not a well defined and good-working learning algorithm for this type of NNs.

In this chapter, the convergence properties of a learning algorithm for CNNs, Recurrent Perceptron Learning Algorithm which is firstly introduced in [61] is analysed thoroughly [62]. This chapter also proposes a new class of CNN in which the initial state of the network are input-dependent so that the piecewise constant function determining the initial conditions in terms of the external inputs can be learned also by a three-stage Perceptron learning rule [1].

4.1. Recurrent Perceptron Learning Algorithm for CNNs

A Cellular Neural Network (CNN) is a 2-dimensional array of cells [5]. Each cell is made up of a linear resistive summing input unit, an R-C linear dynamical unit, and a 3-region, symmetrical, piecewise-linear resistive output unit. The cells in a CNN are connected only to the cells in their nearest neighborhood through a set of templates, i.e., some parameters as connection weights for external inputs, initial states and neighboring cells' outputs.

Several design methods and supervised learning algorithms for determining templates coefficients of CNNs are proposed in the literature [5, 61, 63–76]. As template design methods, well-known relaxation methods for solving linear inequalities are used in [63,64] for finding one of the connection weights providing that desired outputs are in the equilibrium set of a considered CNN. However, for

the methods in [63,64], there is not a general procedure on how to specify initial state vector yielding the desired output for the given external inputs and the found weight vector. A trivial solution in determination of such a proper initial state vector is to take the desired output as the initial state; but this requires the knowledge of desired output which is not available for external inputs outside the training set. On the other hand, a number of supervised learning algorithms to find connection weights of CNNs which yield desired outputs for the given external inputs and the predetermined initial states are developed in the past [61,69–73,76] (see [66] for a review). The backpropagation through time algorithm is applied in [70] for learning desired trajectories in continuous-time CNNs. A modified alternating variable method is used in [77] for learning steady-state outputs in discrete-time CNNs. Both of these algorithms are proposed to be used for any kind of CNNs since they do not impose any constraint needed to be imposed on connection weights for ensuring the complete stability and the bipolarity of steady-state outputs. It is described in [71] that the supervised learning of steady-state outputs in completely stable generalized CNNs [30] is a constrained optimization problem, where objective function is the output error function and constraints are due to some qualitative and quantitative design requirements such as the bipolarity of steady-state outputs and complete stability. The recurrent backpropagation algorithm [78] is applied in [72,79] to a modified version of CNN differing from the original CNN model in the following respects: i) Cells are fully-connected, ii) Output function is a differentiable sigmoidal one, and iii) Network is designed as a globally asymptotically stable network. In [73], the modified versions of the backpropagation through time and the recurrent backpropagation algorithms are used for finding a minimum point of an error measure of the states instead of the output.

The lack of the derivative of error function prevents to use gradient-based methods for finding templates minimizing the error. In order to overcome this problem, the output function can be replaced [61] with a continuously differentiable one which is close to the original piecewise-linear function in (4.3). Whereas the gradient

methods are now applicable, the error surfaces have almost flat regions resulting in extremely slow convergence [61]. An alternative solution to this problem is to use methods not requiring the derivative of error. Such a method is given in [69] by introducing genetic optimization algorithms for supervised learning of the optimal template coefficients. The learning algorithm analyzed in this work, RPLA, constitutes another solution in this direction. RPLA is, indeed, a reinforcement type learning algorithm: It terminates if output mismatching error is zero, otherwise it penalizes connection weights in a manner similar to the perceptron learning rule.

RPLA is firstly presented in [61] for finding template coefficients of a completely-stable CNN to realize an input-(steady-state)output map which is pointwise defined, i.e., described by a set of training samples. Here, the input consists of two parts: The first part is the external input and the second is the initial state. RPLA is a global learning type algorithm in the sense of [66]. This means that it aims to learn not only equilibrium outputs but also their basins of attraction. RPLA has been applied to nonlinear B-template CNNs [80] as well as linear B-template CNNs. A modified version of it has been used for learning regularization parameters in CNN-based early vision models [81, 82]. After the introduction of RPLA, research on designing CNN templates are continued by some other researchers [83–86].

This section is concerned with the convergence properties of RPLA. At first, an introductory information about CNNs and RPLA is given. Then, it is shown here that RPLA with a sufficiently small constant learning rate converges, in finite number steps, to a solution weight vector if such a solution exits and if the conditions of Theorem 4.3 are satisfied. RPLA is indeed reduced to perceptron learning rule [16] if feedback template coefficients except for self-feedback one are set to zero, i.e., the corresponding CNN is in the linear threshold class [67]. This means that a CNN trained with RPLA for a sufficiently small constant learning rate is capable of learning any locally defined function $F_{local}(\cdot) : [-1, 1]^9 \rightarrow \{-1, 1\}$ of the external input whenever its domain space

specified by 3x3 nearest neighborhood is linearly separable. The performance of the algorithm is also demonstrated in learning image processing tasks .

The structure of the section is as follows. In Section 4.1.1., the model and definition of cellular neural networks is given. Section 4.1.2. formulates supervised learning of completely stable CNNs as the minimization of an error function. The dynamics of the difference equations defining the proposed learning algorithm RPLA is analyzed in Section 4.1.3.. Some simulation results on the image processing applications of RPLA are reported in Section 4.1.4.

4.1.1. CNNs

A Cellular Neural Network (CNN) is a 2-dimensional array of cells [5]. Each cell is made up of a linear resistive summing input unit, an R-C linear dynamical unit, and a 3-region, symmetrical, piecewise-linear resistive output unit. The schematic of a cell model is depicted in Figure 4.1. The cells in a CNN are connected only to the cells in their nearest neighborhood defined by the following metric:

$$d(i,j;\hat{\imath},\hat{\jmath}) \;=\; max\{|i-\hat{\imath}|,|j-\hat{\jmath}|\} \tag{4.1}$$

where (i,j) is the vector of integers indexing the cell $C(i,j)$ in the ith row jth column of the 2-dimensional array. The system of equations describing a CNN with the neigborhood size of one is given in (4.2)-(4.3).

$$\dot{x}_{i,j} \;=\; -A\cdot x_{i,j} + \sum_{k,l\in\{-1,0,1\}} w_{k,l}\cdot y_{i+k,j+l} + \sum_{k,l\in\{-1,0,1\}} z_{k,l}\cdot u_{i+k,j+l} + I \tag{4.2}$$

$$y_{i,j} \;=\; f(x_{i,j}) := \frac{1}{2}\cdot\{|x_{i,j}+1| - |x_{i,j}-1|\}. \tag{4.3}$$

Where, A, I, $w_{k,l}$ and $z_{k,l} \in \mathbf{R}$ are constant parameters. $x_{i,j}(t) \in \mathbf{R}$, $y_{i,j}(t) \in [-1,1]$, and $u_{i,j} \in [-1,1]$ respectively denotes the state, output, and (time-invariant) external input associated to a cell $C(i,j)$.

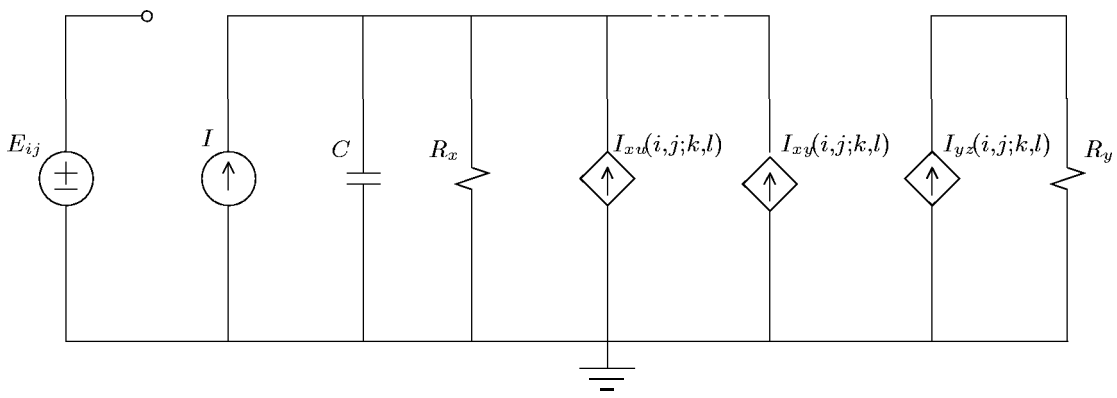


Figure 4.1: CNN cell circuit

It is known in [5] that a CNN is completely stable if the feedback connection weights $w_{k,l}$ are symmetric. Hereafter, the input connection weights $z_{k,l}$ are chosen symmetric for reducing computational costs while the feedback connection weights $w_{k,l}$ are chosen symmetric for ensuring the complete stability, i.e.,

$w_{-1,-1} = w_{1,1}{:=}a_1$, $w_{-1,0} = w_{1,0}{:=}a_2$, $w_{-1,1} = w_{1,-1}{:=}a_3$, $w_{0,-1} = w_{0,1}{:=}a_4$, $w_{0,0}{:=}a_5$;

$z_{-1,-1} = z_{1,1}{:=}b_1$, $z_{-1,0} = z_{1,0}{:=}b_2$, $z_{-1,1} = z_{1,-1}{:=}b_3$, $z_{0,-1} = z_{0,1}{:=}b_4$, $z_{0,0}{:=}b_5$.

Hence, the number of connection weights to be adapted is a small number, 11, for the chosen neighborhood size of one. So, the learning is accomplished through modification of the following weight vector $\mathbf{w} \in \mathbf{R^{11}}$ whose entries are the feedback template coefficients a_i's, the input template coefficients b_j's, and the threshold I.

$$\mathbf{w} := \left[\mathbf{a}^T\ \mathbf{b}^T\ I\right]^T := [\,a_1\ \ a_2\ \ a_3\ \ a_4\ \ a_5\ \ b_1\ \ b_2\ \ b_3\ \ b_4\ \ b_5\ \ I\,]^T. \tag{4.4}$$

4.1.2. Supervised Learning of Completely Stable CNNs

The input vector, for the sake of generality, can be defined as $\mathbf{v}= [\mathbf{v}_u^T\ \mathbf{v}_x^T]^T$. Where, $\mathbf{v}_u= [\ldots, u_{i,j}, \ldots]^T \in \mathbf{R}^m$, and respectively $\mathbf{v}_x = [\ldots, x_{i,j}(0), \ldots]^T \in \mathbf{R}^m$ denotes the vector of external inputs, and respectively the vector of initial states.

For a given input vector $\mathbf{v}$, a completely stable CNN with a chosen weight vector $\mathbf{w}$ in (4.4) will produce an output vector $\mathbf{y}(t)= [\ldots, y_{i,j}(t), \ldots]^T \in \mathbf{R}^m$ tending to a constant vector $\mathbf{y}(\infty)$, called the steady-state output vector. Such CNNs define an algebraic mapping between the input and the (steady-state) output vector spaces. Where, the existence and uniqueness of $\mathbf{y}(\infty)$ for each $\mathbf{v}$ which is needed for defining the mapping is a consequence of the fact that equations in (4.2) together with the piecewise linearity of the function in (4.3) define a state equation system having Lipschitz continuous right hand side.

The supervised learning of steady-state outputs in a CNN can be described as an attempt to approximate an unknown map $\mathbf{d}= \mathcal{H}(\mathbf{v})$ which is defined in a pointwise manner from the input space to the (steady-state) output space by minimizing an output error function $\hat{\mathcal{E}}[\mathbf{w}]$. The network is trained with the following set of pairs which are samples of the map $\mathbf{d} = \mathcal{H}(\mathbf{v})$:

$$\{(\mathbf{v}^1, \mathbf{d}^1), (\mathbf{v}^2, \mathbf{d}^2), \ldots, (\mathbf{v}^L, \mathbf{d}^L)\} \quad . \tag{4.5}$$

Where $\mathbf{v^s}$ and $\mathbf{d^s}$ represents the input and desired (steady-state) output for the s^{th} sample, respectively. The error function $\hat{\mathcal{E}}[\mathbf{w}]$ to be minimized is a measure of the difference between the desired and actual (steady-state) output sets. $\hat{\mathcal{E}}[\mathbf{w}]$ is defined as the following summation of the instantaneous errors $\hat{\mathcal{E}}^s[\mathbf{w}]$ each of which is the square of the Euclidean distance between the desired and actual output vectors corresponding to the s^{th} input vector $\mathbf{v}^s$.

$$\hat{\mathcal{E}}[\mathbf{w}]:= \sum_{s=1}^{L} \hat{\mathcal{E}}^s[\mathbf{w}] \quad = \quad \sum_{s} \sum_{i,j} \left(y^s_{i,j}(\infty) - d^s_{i,j} \right)^2 \tag{4.6}$$

Now, the supervised learning of the steady-state outputs in a completely stable CNN which operates in the bipolar binary steady-state output mode can be formulated as a constrained optimization problem where the objective function is $\hat{\mathcal{E}}[\mathbf{w}]$ and the constraints are i) The bipolarity assumption $a_5 > A$ [5], ii) Any $\mathbf{y}^s\ (\infty)$ should satisfy the state equation system in (4.2)-(4.3) as its steady-state

solution for given $\mathbf{v}_u^s$ and $\mathbf{v}_x^s$. The symmetry conditions imposed on the feedback connection weights are not mentioned here as constraints since these weights were already chosen symmetric in the definition of weight vector $\mathbf{w}$.

Discarding the constant terms from $\hat{\mathcal{E}}(\mathbf{w})$ and dividing it by 4, we can obtain a new error function $\mathcal{E}[\mathbf{w}]$ as in (4.7) under the bipolarity assumption of the steady-state outputs. The bipolarity can be ensured as choosing $a_5 > A$ and choosing initial state vectors different from the equilibrium points in the center or partial saturation regions in the state space.

$$\mathcal{E}[\mathbf{w}]:=\frac{1}{2}\cdot\sum_{i,j,s} y_{i,j}^s(\infty)\cdot\left(y_{i,j}^s(\infty)-d_{i,j}^s\right)=\sum_{(i,j,s)\in D^+} y_{i,j}^s(\infty)-\sum_{(i,j,s)\in D^-} y_{i,j}^s(\infty). \quad (4.7)$$

Where, $D^+:=\{(i,j,s) \mid y_{i,j}^s(\infty)=-d_{i,j}^s=1\}$ and $D^-:=\{(i,j,s) \mid y_{i,j}^s(\infty)=-d_{i,j}^s=-1\}$. In the sequel, the cells indexed by D^+ are called as $+1$ mismatching cells and the cells indexed by D^- are called as -1 mismatching cells. $\mathcal{E}[\mathbf{w}]$ is a sum of the actual steady-state outputs $y_{i,j}^s(\infty)$ mismatching the desired outputs and called as Output Mismatching ERror (OMER) function.

The relation in (4.8) helps us to see how OMER depends on the connection weight vector $\mathbf{w}$. The relation (4.8) describes a cell in the steady-state and it is obtained by setting the left-hand side $\dot{x}_{i,j}$ of the equation (4.2) to zero.

$$\begin{aligned} A\cdot x_{i,j}^s(\infty) &= \sum_{k,l\in\{-1,0,1\}} w_{k,l}\cdot y_{i+k,j+l}^s(\infty)+\sum_{k,l\in\{-1,0,1\}} z_{k,l}\cdot u_{i+k,j+l}^s+I \\ &:= \left[\mathbf{Y}_{i,j}^s\right]^T\cdot\mathbf{w} \quad . \end{aligned} \quad (4.8)$$

where

$$\mathbf{Y}_{i,j}^s=\left[\begin{array}{ccc}\left[\mathbf{y}_{i,j}^s\right]^T & \left[\mathbf{u}_{i,j}^s\right]^T & 1\end{array}\right]^T \quad (4.9)$$

and

$$[\mathbf{y}^s_{i,j}(\infty)] := [\, y^s_{i-1,j-1}(\infty) + y^s_{i+1,j+1}(\infty) \quad y^s_{i-1,j}(\infty) + y^s_{i+1,j}(\infty)$$
$$y^s_{i-1,j+1}(\infty) + y^s_{i+1,j-1}(\infty) \quad y^s_{i,j-1}(\infty) + y^s_{i,j+1}(\infty) \quad y^s_{ij}(\infty) \;]^T \quad (4.10)$$

$$[\mathbf{u}^s_{i,j}] := [\; u^s_{i-1,j-1} + u^s_{i+1,j-1} \quad u^s_{i-1,j} + u^s_{i+1,j}$$
$$u^s_{i-1,j+1} + u^s_{i+1,j-1} \quad u^s_{i,j-1} + u^s_{i,j+1} \quad u^s_{ij} \;]^T \quad (4.11)$$

With the above definitions and with the bipolarity assumption, the steady-state output of a cell in a completely stable CNN can be given as the following implicit relation of connection weights, external inputs and also steady-state outputs of neighboring cells.

$$y^s_{i,j}(\infty) \;=\; \mathrm{sgn}\left[\left[\mathbf{Y}^s_{i,j}\right]^T \cdot \mathbf{w}\right] \;=\; \mathrm{sgn}\left[\; [\mathbf{y}^s_{i,j}]^T \cdot \mathbf{a} + [\mathbf{u}^s_{i,j}]^T \cdot \mathbf{b} + I \;\right] \quad (4.12)$$

For linear threshold class of CNNs, (4.12) is reduced to (4.13) which also does not explicitly describe the steady-state output $y^s_{i,j}(\infty)$ in terms of external inputs, connection weights and initial states.

$$y^s_{i,j}(\infty) \;=\; \mathrm{sgn}\left[w_{0,0} \cdot y^s_{i,j}(\infty) + [\mathbf{u}^s_{i,j}]^T \cdot \mathbf{b} + I\right] \quad (4.13)$$

The dependence of $\mathbf{Y}^s_{i,j}$ and OMER on initial states and connection weights is, in general, quite complicated which makes design and learning problems in CNNs so difficult.

4.1.3. Recurrent Perceptron Learning Algorithm

4.1.3.1. Description of the Algorithm

The algorithm is inspired by the similarity between the input-output relation of a perceptron and the relation (4.12) which describes steady-state behaviour of a cell of completely stable CNNs operating in bipolar mode.

The connection weights characterizing these functions can be found by the following perceptron learning rule.

$$\begin{bmatrix} \mathbf{b}(n+1) \\ \hat{I}(n+1) \end{bmatrix} = \begin{bmatrix} \mathbf{b}(n) \\ \hat{I}(n) \end{bmatrix} - \eta \cdot \begin{bmatrix} \mathbf{u}^s_{i,j} \\ 1 \end{bmatrix} \cdot (d^s_{i,j} - y^s_{i,j}(\infty)) \tag{4.14}$$

Where, $\hat{I} := (w_{0,0} - \frac{1}{A}) \cdot y^s_{i,j}(0) + I$ defines the perceptron threshold for a fixed $w_{0,0}$ with $w_{0,0} > \frac{1}{A}$ and for an $y^s_{i,j}(0)$ chosen identical for all cells and samples, learning rate η is a small constant, and there exists a unique $n := n(i,j,s)$ corresponding to each (i,j,s) for each cycle meaning that the algorithm runs in a data-adaptive mode over training samples and cells until convergence.

Each cell of the linear threshold class CNNs trained by the above algorithm can perform the same local function on its 3x3 external input neighborhood. However, choosing initial conditions $x^s_{i,j}(0)$'s different from one cell to another, one can obtain a CNN where its cells, each of which now has own threshold, realize different but still linearly seperable local functions on external inputs.

The algorithm, RPLA, which is defined by the difference equations in (4.15) is an attempt to generalize the simple perceptron rule into the whole class of completely stable CNNs operating in bipolar mode. CNNs which are not in the linear threshold class can realize some linearly nonseperable local functions of external inputs. This comes from the nonlinear dependence on external inputs of the first part $\mathbf{y}^s_{i,j}$ of the input of the perceptron-like transfer function in (4.12).

RPLA is introduced as considering the perceptron-like relation in (4.12) describing the steady-state behaviour of cells. RPLA updates connection weight vector $\mathbf{w}$ as the same as in perceptron learning rule treating $\mathbf{Y}_{i,j}^{s}$'s as constant inputs to the cells. Due to this nonvalid assumption of $\mathbf{Y}_{i,j}^{s}$'s being constant, the convergence properties of RPLA are different from those of perceptron learning rule. (See Section 2.3.1.) RPLA searches for a solution weight vector providing a set of desired outputs as actual equilibrium outputs for a set of initial states and external inputs. If such a weight vector is found, then RPLA terminates. Otherwise, it updates the weight vector towards annihilating these actual equilibrium outputs.

$$\mathbf{w}(n+1) = [\mathbf{w}(n) - \eta(n) \cdot \mathcal{Y}[\mathbf{w}(n)]]^{+}. \tag{4.15}$$

Where, the vector $\mathcal{Y}[\mathbf{w}(n)]$, which is defined in (4.16), can be viewed as the normal vector of an hyperplane to be crossed while $\mathbf{w}$ tends to a solution weight vector in the $\mathbf{w}$-space.

$$\mathcal{Y}[\mathbf{w}(n)] := \left(\sum_{(i,j,s)\in D^{+}} \mathbf{Y}_{i,j}^{s}(n) - \sum_{(i,j,s)\in D^{-}} \mathbf{Y}_{i,j}^{s}(n) \right) \tag{4.16}$$

$\eta(n)$ is the learning-rate which might be a time varying function but usually chosen as a small positive constant. $[\hat{\mathbf{w}}]^{+}$ denotes the projection of the vector $\hat{\mathbf{w}}$ onto the convex set $\mathcal{A} = \{\mathbf{w} \in \mathbf{R}^{11} \,|\, a_5 > A\}$. The projection $[\cdot]^{+}$ is used for ensuring the bipolarity of the steady-state outputs and is defined as follows. $[\hat{\mathbf{w}}(n)]^{+} = \hat{\mathbf{w}}(n)$ if $\hat{\mathbf{w}}(n) \in \mathcal{A}$. $[\hat{\mathbf{w}}(n)]^{+} = K_n \cdot \hat{\mathbf{w}}(n)$ if $\hat{\mathbf{w}}(n) \notin \mathcal{A}$. Here, $K_n := \mu \cdot \frac{A}{\hat{a}_5(n)}$ with $\mu > 1$ is a constant usually chosen as 1.5.

The steps of RPLA are as follows.

Given: A set of training pairs $\{\mathbf{v}^s, \mathbf{d}^s\}_{s=1}^{L}$, state feedback coefficient A, magnification rate K_n of the projection, learning rate $\eta(n)$.

Step 1: Choose an initial weight vector $\mathbf{w}(0)$ satisfying the bipolarity constraint $a_5 > A$. Set $n = 0$.

Step 2: For the present weight vector $\mathbf{w}(n)$, compute all steady-state outputs $y_{i,j}^s(\infty)$'s by solving the differential equations in (4.2)-(4.3) for each initial state $\mathbf{v}_x^s$ and input $\mathbf{v}_u^s$ vectors belonging to the given training set. Then, construct $\mathcal{Y}[\mathbf{w}(n)]$ in (4.16) and find the next weight vector $\mathbf{w}(n+1)$ according to the difference equation in (4.15).

Step 3: If the updated weight vector $\mathbf{w}(n+1)$ is the same with the previous weight vector $\mathbf{w}(n)$, then terminates the iteration. Otherwise, set $n = n+1$ and go to step 2.

RPLA has the following features first two of them distinguish it from perceptron learning rule:

i : RPLA is block-adaptive since, at each step, it updates the weight vector taking into account the contributions of all the training samples and cells.

ii : The vector $\mathcal{Y}[\mathbf{w}(n)]$ changes while weight vector $\mathbf{w}(n)$ is updated.

iii : If actual steady-state outputs $y_{i,j}^s(\infty)$'s are replaced with desired steady-state outputs $d_{i,j}^s$'s in the definition of $\mathbf{Y}_{i,j}^s$, then RPLA becomes to an algorithm which learns equilibrium outputs for the given external inputs but can not learn their basins of attraction.

4.1.3.2. Neurophilosophical Properties of RPLA

The following properties are very useful for understanding the behaviour of RPLA. Property 4.1 is quite meaningfull from the neurophilosophical point of view: The self-feedback template coefficient should be decreased to soften the positive feedback causing output value mismatch.

Property 4.1 *The 5^{th} element $\mathcal{Y}_5[\mathbf{w}(n)]$ of the vector $\mathcal{Y}[\mathbf{w}(n)]$ is equivalent to the OMER $\mathcal{E}[\mathbf{w}(n)]$; and consequently, for learning rates $\eta(n) > 0$, the 5^{th} element*

$a_5(n)$ of $\mathbf{w}(n)$ is always nonincreasing unless $a_5(n) - \eta(n) \cdot \mathcal{E}[\mathbf{w}(n)] \leq 1$ and remains constant if the OMER is zero.

Proof : The equivalence of $\mathcal{Y}_5[\mathbf{w}(n)]$ to $\mathcal{E}[\mathbf{w}(n)]$ follows from the definitions in (4.7), (4.9), (4.10) and (4.16). If $a_5(n) - \eta(n) \cdot \mathcal{E}[\mathbf{w}(n)] > 1$, then $a_5(n+1) = a_5(n) - \eta(n) \cdot \mathcal{Y}_5[\mathbf{w}(n)]$. The proof is concluded by the observations of $\eta(n) > 0$ and $\mathcal{Y}_5[\mathbf{w}(n)] = \mathcal{E}[\mathbf{w}(n)] \geq 0$. □

Property 4.2 *The 11^{th} element $\mathcal{Y}_{11}[\mathbf{w}(n)]$ of $\mathcal{Y}[\mathbf{w}(n)]$ is equal to the number $\#(D^+[\mathbf{w}(n)]) - \#(D^-[\mathbf{w}(n)])$. Where, $\#(D^+[\mathbf{w}(n)])$ and $\#(D^-[\mathbf{w}(n)])$ denotes the cardinality of the set of +1 mismatching cells and the set of -1 mismatching cells, respectively.*

Proof : The proof is immediate by the definitions in (4.9), (4.10) and (4.16). □

Ignoring the effects of initial value $I(0)$ and of the magnification by factor K in the steps requiring the projection, the final I obtained can be considered as a cummulative sum of past differences between the numbers of +1 mismatching cells and -1 mismatching cells.

Property 4.3 *Assume that the actual (steady-state) output of any boundary cell in a CNN matches the desired value. Then, the 1^{st} element $\mathcal{Y}_1[\mathbf{w}]$ of $\mathcal{Y}[\mathbf{w}]$ is equal to $2 \cdot [\#(ULLR)_s - \#(ULLR)_o]$. $(ULLR)_s$ denotes the set of mismatching cells each of which has the (steady-state) output value same with its Upper Left neighbor's output as well as same with its Lower Right neighbor's output. $(ULLR)_o$ denotes the set of mismatching cells each of which has the (steady-state) output value opposite to its Upper Left neighbor's output as well as opposite to its Lower Right neighbor's output.*

Proof : Note that the cell $C(i-1, j-1)$ and $C(i+1, j+1)$ is the upper left and the lower right neighbor of the cell $C(i, j)$. The proof follows from the definition of $\mathcal{Y}[\mathbf{w}]$ in (4.16) and the definitions in (4.9)-(4.10). □

4.1.3.3. Relation between Fixed Points and Zero Error

The next two properties describes the correspondence of the fixed points of RPLA to the minimum points of OMER. It will be shown by the Properties 4.4 and 4.5 that the problem of finding a weight vector $\mathbf{w}^\star$ providing the desired outputs $\mathbf{d}^s$'s as the actual outputs $\mathbf{y}^s$'s for the chosen initial states $\mathbf{v}_x^s$'s and for the given inputs $\mathbf{v}_u^s$'s is equivalent to the problem of finding one of the nonpathological fixed points of the RPLA.

Property 4.4 *Any weight vector* $\mathbf{w}^\star$ *yielding a zero OMER is a fixed point of RPLA defined by the set of difference equations in (4.15).*

Proof : Observe that $\mathcal{E}[\mathbf{w}^\star] = 0$ if and only if $\#(D^+[\mathbf{w}]) = \#(D^-[\mathbf{w}]) = \emptyset$. The equality $\#(D^+[\mathbf{w}]) = \#(D^-[\mathbf{w}]) = \emptyset$ implies that $\mathcal{Y}[\mathbf{w}^\star] = \mathbf{0} \in \mathbf{R}^{11}$ and then $\mathbf{w}^\star$ is a fixed point of RPLA. □

Property 5 explains if there is a fixed point $\mathbf{w}^\star$ of RPLA which gives a nonzero OMER.

Property 4.5 *Except for the pathological weight vectors* $\mathbf{w}$*'s satisfying the set of equations* $\mathcal{Y}[\mathbf{w}] = \frac{\mathcal{E}[\mathbf{w}]}{a_5} \cdot \mathbf{w}$*, each fixed point of the RPLA with a learning rate* $\eta(n) \neq 0$ *for all* n *yields a zero OMER.*

Proof : If a weight vector $\mathbf{w}$ satisfies $\mathcal{Y}[\mathbf{w}] = \frac{\mathcal{E}[\mathbf{w}]}{a_5} \cdot \mathbf{w}$, then $\mathbf{w}^\star$ satisfies the set of equations $\mathbf{w}^\star = [\mathbf{w}^\star - \epsilon \cdot \mathcal{Y}[\mathbf{w}^\star]]^+ = K^\star \cdot \left[\mathbf{w}^\star - \epsilon \cdot \frac{\mathcal{E}[\mathbf{w}^\star]}{a_5} \cdot (\mathbf{w}^\star)\right]^+$ for $K^\star = \frac{a_5}{a_5 - \epsilon \cdot \mathcal{E}[\mathbf{w}^\star]}$. On the contrary, if such a weight vector does not exist, then the only possibility for a weight vector $\mathbf{w}$ to be a fixed point of RPLA is that $\mathbf{w}$ satisfies $\mathcal{Y}[\mathbf{w}^\star] = \mathbf{0}$ which implies $\mathcal{E}[\mathbf{w}^\star] = 0$. □

4.1.3.4. How to Start and Restart the RPLA

A necessary condition for the existence of a nonpathological fixed point is that each saturation region $\mathbf{B}_J$ whose associated output $\mathbf{y}_J$ coincides with one of the desired outputs $\mathbf{d}^s$'s contains an equilibrium point. The saturation region $\mathbf{B}_J$ is defined as

$$\mathbf{B}_J := \{\mathbf{x} \in \mathbf{R}^m \mid \mathbf{x}_i \geq 1 \; for \; i \in J; \mathbf{x}_i \leq -1 \; for \; i \in J\}$$

where, $\mathbf{x} = [\ldots, x_{i,j}, \ldots]^T \in \mathbf{R}^m$, $J \subseteq \{1, 2, \ldots, m\}$, and $(\mathbf{y}_J)_i = 1 \; for \; i \in J$, $(\mathbf{y}_J)_i = -1 \; for \; i \in J$. Theorem 4.1 gives a condition ensuring that each saturation region $\mathbf{B}_J$ has an equilibrium point, and hence it provides a set of template coefficients for which any desired output can be reached with a suitably chosen initial condition.

Theorem 4.1 *Assume that the connection weights satisfy the inequality* $a_5 > A + T$. *Here,* $T := 2 \cdot \{\sum_{i=1}^{4}(|a_i| + |b_i|)\} + |b_5| + I$. *Then, there exists a unique equilibrium point in each of* 2^m *saturation regions* $\mathbf{B}_J$*'s.*

Any weight vector $\mathbf{w}$ satisfying the condition stated in Theorem 4.1 is a solution to the linear inequality system considered in [63, 64]. For such a weight vector, the initial state $\mathbf{v}_x^s$ chosen properly, i.e. chosen in the basin of attraction of the equilibrium point in the saturation region whose associated output coincides the desired output $\mathbf{d}^s$, yields the desired output. The proposed learning algorithm RPLA is usually started at an initial weight vector $\mathbf{w}(0)$ satisfying the condition in Theorem 4.1. The initial states which are not chosen properly give a nonzero OMER. Then, the weight vector should be changed for suppressing the equilibrium points yielding undesired outputs by violating the condition in Theorem 4.1. The RPLA stops at the weight vectors providing that, for all s, the chosen initial state $\mathbf{v}_x^s$ is in the basin of attraction of the equilibrium point whose

associated output is $\mathbf{d}^s$. Since a_5 is always decreasing for an arbitrary learning rate $\eta(n) > 0$ and for nonzero OMER, then the algorithm RPLA may need a projection before terminating. One can think that the magnification by factor K used for projecting the updated weight vector onto the bipolarity constraint set may destroy the learned outputs and create new equilibrium points giving undesired outputs. However, this is not the case as explained in Theorem 4.2.

Theorem 4.2 *Assume that there exists an equilibrium point* $\mathbf{x}_J$ *in the saturation region* $\mathbf{B}_J$ *for the given external input* $\mathbf{v}_u$ *and for the weight vector* $\mathbf{w}$. *Then, the saturation region* $\mathbf{B}_J$ *contains an equilibrium point* $\bar{\mathbf{x}}_J = K \cdot \mathbf{x}_J$ *for the external input* $\mathbf{v}_u$ *and for the weight vector* $\bar{\mathbf{w}} = K \cdot \mathbf{w}$ *with* $K \geq 1$.

Whereas the magnification by factor K does not destroy any existing equilibrium point, it may create a new equilibrium point in a saturation region. Moreover, the actual steady-state outputs $\mathbf{y}(\infty)$'s which are obtained for the same external input $\mathbf{v}_u$ but for different weight vectors $K \cdot \mathbf{w}$ and $\mathbf{w}$, may differ from each other depending on the magnification factor K. The OMER may therefore increase at the steps requiring the projection. Inspite of the mentioned facts, the weight vector obtained after the magnification is a good initial vector for restarting the RPLA.

4.1.3.5. Sufficient Conditions for Convergence to Fixed Points

The Properties and Theorems in Subsections 4.1.3.1. - 4.1.3.4. describe several aspects of the learning process ruled by the RPLA. The main concern in any iterative algorithm is the convergence of the sequence produced by the algorithm to a desired pattern usually a fixed point. Theorem 4.3 presents a sufficient condition for ensuring the convergence of the sequence of weight vectors to one of the nonpathalogical fixed points of the RPLA.

Theorem 4.3 is based on the following three assumptions.

Assumption 1 *There exists a solution weight vector* $\mathbf{w}^\star$ *so that it satisfies the bipolarity constraint and yields the zero OMER.*

Assumption 2 *For a chosen initial vector* $\mathbf{w}(0)$ *and learning rate* $\hat{\eta}(n)$, $\hat{\mathbf{w}}(n) := \mathbf{w}(n) - \hat{\eta}(n) \cdot \mathcal{Y}[\mathbf{w}(n)]$ *satisfies the bipolarity condition for each* n.

Assumption 3 *There exists a solution weight vector* $\mathbf{w}^\star$ *satisfying the inequality in (4.17) for each* n.

$$A \cdot \left(\sum_{(i,j,s) \in \{D^+ \cup D^-\}} |x_{i,j}^s(\infty)(n)| \right) \quad > \quad [\mathbf{w}^\star]^T \mathcal{Y}[\mathbf{w}(n)]. \tag{4.17}$$

Theorem 4.3 *Under the Assumptions 1–3, the RPLA with a sufficiently small constant learning rate converges, in finite iteration steps, to a weight vector yielding the desired outputs as the actual outputs for the given initial states and external inputs.*

Proof : By the Assumption 2, the magnification by factor K is not needed to be applied in any iteration step, i.e. $\mathbf{w}(n+1) = \mathbf{w}(n) - \hat{\eta}(n) \cdot \mathcal{Y}[\mathbf{w}(n)]$. Hence, using the properties of the Euclidean norm $||\cdot||_2$, the equation in (4.18) is obtained for any solution weight vector $\mathbf{w}^\star$.

$$\begin{aligned} \|\mathbf{w}(n+1) - \mathbf{w}^\star\|_2^2 \quad &= \quad \|\mathbf{w}(n) - \mathbf{w}^\star\|_2^2 + \hat{\eta}^2(n) \cdot \| \mathcal{Y}[\mathbf{w}(n)] \|_2^2 \\ &\quad -2 \cdot \hat{\eta}(n) \cdot [\mathbf{w}(n) - \mathbf{w}^\star]^T \mathcal{Y}[\mathbf{w}(n)] \ . \end{aligned} \tag{4.18}$$

Assumption 3 implies that there exists a positive number $\bar{\eta}$ satisfying (4.19).

$$\frac{1}{||\mathcal{Y}[\mathbf{w}(n)]||_2^2} \cdot [\mathbf{w}(n) - \mathbf{w}^\star]^T \mathcal{Y}[\mathbf{w}(n)] \geq \bar{\eta}(n) > 0. \tag{4.19}$$

This fact can be seen from that i) the lefthand side of the inequality in (4.19) is equal to $[\mathbf{w}(n)]^T \mathcal{Y}[\mathbf{w}(n)]$, ii) the equations in (4.8) and the definition of $\mathcal{Y}[\mathbf{w}(n)]$ in (4.16).

Under the assumption of the nonviolation of the bipolarity condition, the equation (4.20) is obtained.

$$\begin{aligned} \|\mathbf{w}(n+1) - \mathbf{w}^\star\|_2^2 &= \|\mathbf{w}(n) - \mathbf{w}^\star\|_2^2 + \bar{\eta}^2(n) \cdot \| \mathcal{Y}[\mathbf{w}(n)] \|_2^2 \\ &\quad -2 \cdot \bar{\eta}(n) \cdot [\mathbf{w}(n) - \mathbf{w}^\star]^T \mathcal{Y}[\mathbf{w}(n)] \,. \end{aligned} \tag{4.20}$$

The inequality in (4.19) implies that the third term in the righthand side of (4.20) dominates the second term and hence the distance between the weight vector $\mathbf{w}(n)$ and the solution weight vector $\mathbf{w}^\star$ is reduced by a positive amount:

$$|| \mathbf{w}(n+1) - \mathbf{w}^\star||_2^2 - || \mathbf{w}(n) - \mathbf{w}^\star||_2^2 \leq -\bar{\eta}(n) \cdot [\mathbf{w}(n) - \mathbf{w}^\star]^T \mathcal{Y}[\mathbf{w}(n)] \,. \tag{4.21}$$

The equation (4.21) completes the proof. □

Unfortunately, Theorem 4.3 does not give a constructive way for obtaining a positive constant learning rate which ensures the convergence of the RPLA to a solution weight vector. Instead, it describes a condition forwhich the RPLA with a positive learning rate chosen sufficiently small converges to a solution weight vector satisfying the condition.

4.1.4. Learning Image Processing Using RPLA

The CNN with its 2-dimensional array architecture is a natural candidate for image processing. On the other hand, any input-output function to be realized by CNNs can be visualised as an image processing task where the external input, the initial condition and the output vector arranged as a 2-dimensional array is the external input image, the initial image and the output image, respectively. The external input image together with the initial image constitutes the input

image of the CNN. In the applications, either one of the external input image and the initial image is used as the image to be processed while the other is set to a suitable constant image or both of them are used as the input image to be processed.

The supervised learning algorithm, RPLA, can be considered as a tool for finding a feasible weight vector providing that the actual output images match the desired images for the given input image.

An image processing application, corner detection, of the RPLA is shown here as an example. 16×16 images are used in the training phase of the applications. In the example of corner detection, the initial image were chosen equal to the external input image and the image to be processed were taken as the external image. the initial images and the external input images were chosen bipolar, i.e. each pixel is either $+1$ (black) or else -1 (white). In the simulation example, the same external input images were used as the input parts of the training pairs. For the same external input image, the solution weight vectors obtained perform different tasks. This shows that the success of the RPLA does not, at least for the three image processing problems considered in this paper, come from the suitable choice of the input images.

In the sequel, the following matrix notations standard in CNN literature will be used for presenting the connection weights.

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_4 \\ a_3 & a_2 & a_1 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_4 \\ b_3 & b_2 & b_1 \end{bmatrix}, \qquad I \tag{4.22}$$

where, $\mathbf{A}$, $\mathbf{B}$ and I denotes the feedback template, the input template and the threshold template, respectively.

In the application given below, the initial value $\epsilon(0)$ of the learning rate is chosen as 0.0004. The learning rate $\epsilon(n)$ is remained constant if the OMER changes in 10

iteration steps and magnified by 2 if the OMER does not change in 10 iteration steps.

4.1.4.1. Corner Detection

The initial templates were chosen as in (4.23) which are the initial templates used also in the edge detection aplication. Figure 4.a shows the initial images which are identical to the external images in Figure 4.b. The desired steady-state outputs are given in Figure 4.g. The RPLA was run by the positive constant learning rate $\epsilon = 0.0002$. The actual steady-state outputs at the first through third steps are obtained with the OMER equal to 544 given in Figure 4.b. Figure 4.c, Figure 4.d, Figure 4.e, Figure 4.f and Figure 4.g shows the actual steady-state outputs at the 4^{th}, 10-15^{th}, 19^{th}, 28-29^{th} and 48^{th} steps, respectively. The final templates found at the end of the 48 iterations yield zero OMER for the 5 training samples used. These solution templates are given in (4.24).

$$\mathbf{A}_1^\circ = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B}_1^\circ = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad I_1^\circ = 0. \tag{4.23}$$

$$\begin{aligned} \mathbf{A}_2^\star &= \begin{bmatrix} -0.210844 & -0.153426 & -0.198075 \\ -0.084514 & 3.331127 & -0.084514 \\ -0.198075 & -0.153426 & -0.210844 \end{bmatrix}, \\ \mathbf{B}_2^\star &= \begin{bmatrix} -0.345449 & -0.450396 & -0.349939 \\ -0.510285 & -0.583862 & -0.510285 \\ -0.349939 & -0.450396 & -0.345449 \end{bmatrix}, \\ I_2^\star &= -0.621101 \ . \end{aligned} \tag{4.24}$$

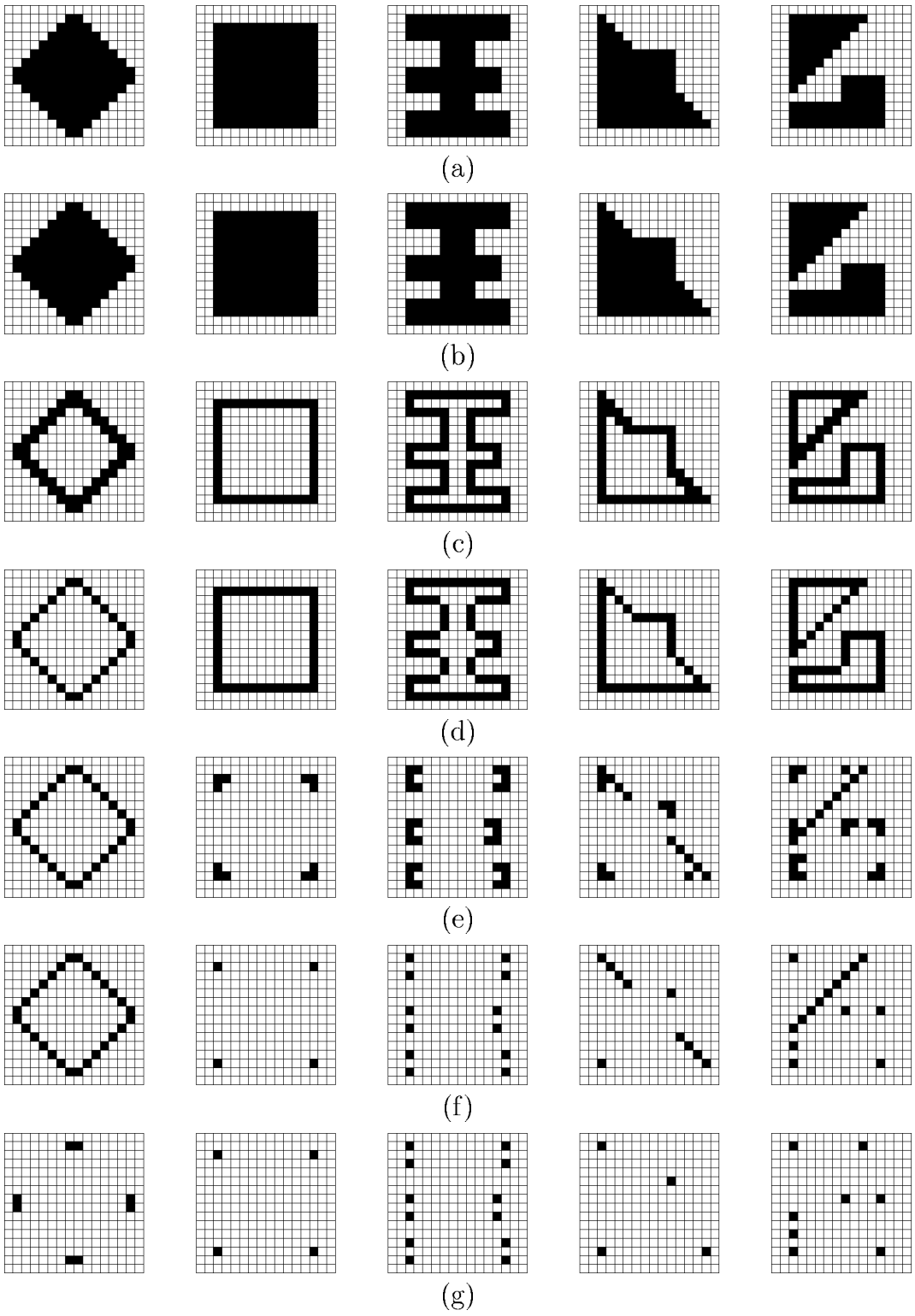


Figure 4.2: Learning corner detection. (a) The initial images. (b) The input images. (c–f) The actual output images at some intermediate steps. (g) The desired output images [4].

4.1.4.2. Discussion

We can conclude that some sufficient conditions for the recurrent perceptron learning algorithm for CNNs have been given. Also, the performance of the developed algorithm has been tested on learning an image processing task, corner detection. The algorithm can be used for learning algebraic mappings from $[-1,+1]^m$ to $\{-1,+1\}^m$; but it has been observed that it is succesfull in learning binary mappings.

4.2. Threshold Class CNNs with Input-Dependent Initial State

CNNs are made up of R-C flip-flop like cells arranged as a 2-dimensional grid with a space invariant connection weight pattern [5]. They are usually operated in a completely stable mode with bipolar steady-state output property. Such CNNs with the neighborhood size of one perform an algebraic (local) mapping $f_{local}(\cdot) : [-1,1]^9 \rightarrow \{-1,1\}$, here any point belongs to the domain $[-1,1]^9$ is associated with a 3×3 window on the external input grid.

Many design methods and learning algorithms [65], [61] have been proposed for determining feedback ($\mathbf{A}$), input ($\mathbf{B}$), and threshold (I) templates to perform a given task. The design and training of CNNs are, in general, very complex and problematic. In most of the applications, researchers use the linear threshold class CNNs since they can accomplish a lot of tasks and they are simple to design as a consequence of the structure of their $\mathbf{A}$ template having a center as the unique nonzero element. As will be cleared by this paper, threshold class CNNs can be trained by Perceptron learning rule to perform linearly separable (local) threshold functions and furthermore this training approach can be applied also to linearly nonseparable function cases by using the input-dependent initial conditions providing nonconstant thresholds for our Perceptron like CNN cells. The advantage of using the introduced class of CNNs over the multilayer linear threshold class CNNs lies on the fact that the piecewise constant function

determining the initial conditions in terms of the external inputs can be learned also by a three-stage Perceptron learning rule. By no means, the developed learning procedure can be applied also for designing a nonconstant-threshold Perceptron to perform linearly nonseparable threshold functions. Hence, the design of discrete Perceptron (thereafter will be called as Perceptron) networks in performing linearly nonseparable functions can be accomplished by a three-stage Perceptron learning rule applied on a layer of input-dependent threshold Perceptron.

Here the equivalence of the cells of the linear threshold class CNNs and the Perceptrons is described. The proposed three-stage learning procedure for learning the template coefficients of the considered CNNs and the coefficients defining the initial states in terms of the external inputs are presented and for the edge detection task as a special linearly nonseparable threshold function the proposed learning procedure is applied on the design of the considered CNNs.

4.2.1. Linear Threshold Class CNN Cells as Perceptron

A CNN with the neighborhood size of one is a 2-dimensional array of cells described by the equations given in Equations 4.25-4.26.

$$\dot{x}_{i,j} = -S \cdot x_{i,j} + \sum_{k,l \in \{-1,0,1\}} a_{k,l} \cdot y_{i+k,j+l} + \sum_{k,l \in \{-1,0,1\}} b_{k,l} \cdot u_{i+k,j+l} \; + \; I \qquad (4.25)$$

$$y_{i,j} = f(x_{i,j}) = \frac{1}{2}\{|x_{i,j}+1| - |x_{i,j}-1|\} \qquad (4.26)$$

Where S, I, $a_{k,l}$'s and $b_{k,l}$'s are real constants called as state feedback, threshold, feedback template, and input template coefficients, respectively. $x_{i,j}(t) \in R$, $y_{i,j}(t) \in [-1,1]$ and $u_{i,j} \in [-1,1]$ denotes the state, output and external input associated to a cell $C(i,j)$, respectively.

By setting the left-hand side $\dot{x}_{i,j}$ of the equation (4.25) to zero, the relation (4.27) describing a cell in the steady-state is obtained.

$$S \cdot x_{i,j}(\infty) = \sum_{k,l \in \{-1,0,1\}} a_{k,l} \cdot y_{i+k,j+l} + \sum_{k,l \in \{-1,0,1\}} b_{k,l} \cdot u_{i+k,j+l} + I \tag{4.27}$$

Equation (4.27) together with $y_{i,j}(\infty) = \text{sgn}[x_{i,j}(\infty)]$ which is valid under the bipolar steady-state output assumption yields the implicit relation in (4.28) [61]. The relation (4.28) gives the steady-state output of a cell in a completely stable CNN having bipolar output property. It resembles the input-output relation of a discrete Perceptron [16].

$$y_{i,j}(\infty) = sgn\left[[\mathbf{y}_{i,j}(\infty)]^T \cdot \mathbf{a} + [\mathbf{u}_{ij}]^T \cdot \mathbf{b} + I\right] \tag{4.28}$$

where,

$$\begin{aligned}[\mathbf{y}_{i,j}(\infty)] \quad = [\;& y_{i-1,j-1}(\infty) + y_{i+1,j-1}(\infty) \quad y_{i-1,j}(\infty) + y_{i+1,j}(\infty) \\ & y_{i-1,j+1}(\infty) + y_{i+1,j-1}(\infty) \quad y_{i,j-1}(\infty) + y_{i,j+1}(\infty) \quad y_{ij}(\infty)\;]^T\end{aligned}$$

$$\begin{aligned}[\mathbf{u}_{i,j}] \quad = [\;& u_{i-1,j-1} \quad u_{i-1,j} \quad u_{i-1,j+1} \quad u_{i,j-1} \\ & u_{i,j} \quad u_{i,j+1} \quad u_{i+1,j-1} \quad u_{i+1,j} \quad u_{i+1,j+1}\;]^T\end{aligned}$$

$$\mathbf{a} = [\; a_{-1,-1} \quad a_{-1,0} \quad a_{-1,1} \quad a_{0,-1} \quad a_{0,0} \;]$$

$$\mathbf{b} = [\; b_{-1,-1} \quad b_{-1,0} \quad b_{-1,1} \quad b_{0,-1} \quad b_{0,0} \quad b_{0,1} \quad b_{1,-1} \quad b_{1,0} \quad b_{1,1} \;]$$

Note that the feedback template coefficients $a_{k,l}$'s are chosen symmetric for ensuring the completely stability: $a_{-1,-1} = a_{1,1}$, $a_{-1,0} = a_{1,0}$, $a_{-1,1} = a_{1,-1}$, $a_{0,-1} = a_{0,1}$.

For a linear threshold class CNN, (4.28) reduced to (4.29) since, except for the center element, all elements of the **A** template are zero: $a_{k,l} = 0$ for all $(k,l) \neq (0,0)$ and $a_{0,0} \neq 0$.

$$y_{i,j}(\infty) = \text{sgn}\left[a_{0,0} \cdot y_{i,j}(\infty) + [\mathbf{u}_{i,j}]^T \cdot \mathbf{b} + I\right] \tag{4.29}$$

Both of (4.28) and (4.29) do not explicitly describe the steady state output $y_{i,j}(\infty)$ in terms of external inputs, connection weights and initial states. However, for the linear threshold class CNNs, $y_{i,j}(\infty)$ can be obtained in the following explicit form [67].

$$y_{i,j}(\infty) = \text{sgn}\left[\left(a_{0,0} - \frac{1}{S}\right) \cdot y_{i,j}(0) + [\mathbf{u}_{i,j}]^T \cdot \mathbf{b} + I\right] \tag{4.30}$$

Equation 4.30 exactly corresponds to the Perceptron's input-output relation given below.

$$y = \text{sgn}[\mathbf{w}^T \cdot \mathbf{u} + \theta] \tag{4.31}$$

Where $\mathbf{w}$, $\mathbf{u}$ and θ stands for $\mathbf{b}$, $[\mathbf{u}_{i,j}]$ and $\left(a_{0,0} - \frac{1}{S}\right) y_{i,j}(0) + I$, respectively.

We can draw the conclusion that any cell of a linear threshold class CNN behaves exactly in the same manner with a modified Perceptron where the threshold is determined by the initial state. So any learning algorithm and design method developed for Perceptrons can be applied also to the linear threshold class CNNs.

4.2.2. Linearly Nonseparable Functions and Modified Perceptron

The problem of learning in a Perceptron can be stated as follows: given a set $U = U^+ \cup U^- = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m\} \subset \mathbf{R}^d$ of m input vectors, determine a hyperplane such that the vectors $\mathbf{u}_i \in U^+$ lay on the positive halfplane of the hyperplane

while the vectors $\mathbf{u}_i \in U^-$ lay on the negative halfplane. If such a hyperplane exists for the given set U, then it is called as linearly separable and if not then linearly nonseparable.

It is well-known that a Perceptron trained by Perceptron learning rule can learn a linearly separable input vector set in finite time steps for a sufficiently small learning factor. Furthermore, the behaviour of Perceptron learning rule, when the input patterns are linearly nonseparable, provides the possibility of learning one of the largest linearly separable subsets U_S's of the given linearly nonseparable training set [38]; here, a U_S is defined as a linearly separable subset of U with maximum cardinality. Although this possibility, which we will exploit in our three-stage learning procedure, is quite usefull to identify such a separable subset, some structural modifications are still be needed for solving the problem of performing nonseparable threshold functions by Perceptron networks. Multilayer networks of Perceptrons offer such a solution but with lacking the possibility of using Perceptron learning algorithm to find their connection weights. Another solution that we are proposing here together with a three-stage Perceptron learning procedure is to use a single nonconstant-threshold Perceptron whose threshold value is a piecewise constant function of the inputs:

$$y = \text{sgn}(\mathbf{w}_1^T \mathbf{u} + \theta) \tag{4.32}$$

$$\theta = \theta_1 + \theta_3 \cdot \text{stp}(\mathbf{w}_2^T \mathbf{u} + \theta_2) \tag{4.33}$$

Where $\text{stp}(\cdot)$ is the step function, $\theta_i \in \mathbf{R}$ for all $i \in \{1, 2, 3\}$ and $\mathbf{w}_1$, $\mathbf{w}_2$ are weight vectors. Instead of the step function, signum function also could be used affecting the values of θ_1 and θ_3 only.

The nonconstant-threshold of the modified Perceptron defined by (4.32)-(4.33) enables us to classify some linearly nonseparable input sets explained below. Furthermore, all parameters $\mathbf{w}_1$, $\mathbf{w}_2$, θ_i, $i \in \{1, 2, 3\}$ can be learned by Perceptron

Table 4.1: Three step learning algorithm for Threshold class CNNs with Input dependent initial state [1].

Step 1: Train a Perceptron by Perceptron learning rule for a given input vector set U. Hence, obtain the weight vector $\mathbf{w}$ and the threshold θ which define a hyperplane separating a largest linearly separable subset U_S of U.

Step 2: Train another Perceptron again by Perceptron learning rule to determine the weight vector $\mathbf{w}_2$ and the threshold θ_2 which define a hyperplane such that the nonseparable vectors in U_{NS} lay on its positive halfplane while the separable vectors in U_S lay on its negative halfplane.

Step 3: Determine θ_3 is a real number such that its amplitude $|\theta_3|$ is strictly greater than the maximum of $|\mathbf{w}_1^T\mathbf{u} + \theta_1|$ over all inputs, and θ_3 is positive if $U_{NS} \subset U_+$ and is negative if $U_{NS} \subset U_-$.

---

learning rule applied within a three-stage procedure given in Table 1. The proposed learning procedure can be applied on a linearly nonseparable set U if the complement $U_{NS} = U \backslash U_S$ of a largest linearly separable subset U_S of the set U consists of vectors all of which belongs either to U^+ or to U^- and furthermore they are linearly separable from the vectors in U_S. The binary edge detection task which we will consider in Section 4.2.3. can be defined as the classification of such a nonseparable input vector set.

The nonconstant-threshold Perceptron given by (4.32)-(4.33) defines a piecewise-linear discriminant function which assigns the inputs belonging to a piecewise-linear halfplane into one class and assigns the others into a second class. To illustrate such kind of partitions of the space, two specific examples are given in Figure 3.2. Where, the circles and squares correspond to the vectors belonging to the sets U_+ and U_-, respectively. The piecewise-linear discriminator represents the set of points satisfying the equation $\mathbf{w}_1^T\mathbf{u} + \theta_1 + \theta_3 \cdot \text{stp}(\mathbf{w}_2^T\mathbf{u} + \theta_2) = 0$. Both examples of Figure 3.2 show linearly nonseparable input sets which can

be realized by a single nonconstant-threshold Perceptron of (4.32)-(4.33). But, only the set illustrated in Figure 3.2.a can be learned by the given three-stage procedure since the nonseparable vectors are linearly separable from the others.

It is obvious by Section 4.2.2. and by (4.32)-(4.33) that linear threshold class CNNs can perform certain linearly nonseparable (local) threshold functions and furthermore their template coefficients and initial conditions can be learned by the above three-stage learning procedure.

4.2.3. Binary Edge Detection as a Linearly Nonseparable Threshold Function

We will justify our design method by computer experiments done on a specific example, namely the edge detection. First, we pose the binary edge detection task as a linearly nonseparable but piecewise-linearly separable threshold function of the type depicted in Figure 3.2.a. Second, we will train a threshold class CNN with input-dependent initial state to learn this task. Then, we will examine the performance of this CNN trained with binary input vectors on the edge detection of some binary and gray level images.

Since the neighborhood size is chosen one, a CNN cell is fed by external inputs in its 3x3 neighborhood. Therefore, there are $2^9 = 512$ possible input patterns for binary images. Edge detection task can be viewed as a pixel classification problem such that a part of the pixels is assigned to edge class and the other part to nonedge class. The pixels in the edge class are represented by black $(+1)$ while the others by white (-1). We define the edges as at least two-pixel lenght continuous lines with one-pixel width. As a consequence of this definition, we identify 239 input patterns belonging to the edge class and 273 ones to the nonedge class. We examined whether the set of input vectors each of which corresponds to one of the 512 possible 3x3 window patterns in the input image space is linearly separable or not. Then, we found the set is linearly nonseparable but the complement of a largest linearly separable subset is included by the

nonedge class and the nonseparable vector can be linearly separated from the others. This gives us the possibility of using the three-stage learning procedure of Section 4.2.2. to train the introduced piecewise-constant threshold Perceptron for performing the ideal edge detection problem. The solution weight vectors and thresholds found are as follows.

$$\begin{aligned}\mathbf{w}_1 \;=\; & [-0.013 \quad -0.406 \quad -0.024 \quad -0.74 \\ & +2.55 \quad -0.485 \quad -0.21 \quad -0.88 \quad -0.13 \;]^T \end{aligned} \tag{4.34}$$

$$\begin{aligned}\mathbf{w}_2 \;=\; & [-1.28 \quad -1.28 \quad -1.28 \quad -1.28 \\ & +1.28 \quad -1.28 \quad -1.28 \quad -1.28 \quad -1.28]^T \end{aligned} \tag{4.35}$$

$$\theta_1 = [-0.427] \qquad \theta_2 = [-10.53] \qquad \theta_3 = [-42.5\,] \tag{4.36}$$

The above parameters yield a threshold class CNN with input-dependent initial conditions having the following template values and the piecewise constant function defining the initial conditions in terms of the inputs.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -0.013 & -0.406 & -0.024 \\ -0.74 & +2.55 & -0.485 \\ -0.21 & -0.88 & -0.13 \end{bmatrix} \qquad I = -0.427 \tag{4.37}$$

$$x(0) = \theta_3 \cdot \mathrm{stp}\left(\omega_2^T \cdot \mathbf{u}\right) - 10.53 \tag{4.38}$$

We apply the found edge detection CNN having the above parameters on some images. The results are presented in Figure 4.3. The performance of our CNN is compared with the one of the constant initial condition CNN given in [5]. For binary edge images, our CNN finds the edge images which are ideal in the above sense. Note that the edge image obtained by our CNN does not contain

an isolated pixel and its edges are of one pixel width. The 7-bit Lenna image was obtained from 256 gray level image by thresholding with the threshold level of 128. It could be obtained also by a linear threshold class CNN. As compared to the CNN in [1], the performance of our CNN is much better for gray level images. The proposed three-stage learning procedure could be applied also on learning edges for gray level images. However, for gray level images, it seems very difficult (may be impossible) to define ideal edges in an exact manner. If it would be possible for considering 3x3 windows only, then we will have 256^9 training pairs for 256 gray level images. On the other hand, the usage of an edge image obtained by an available efficient algorithms might cause inconsistency of the training set. Such a possible ill-defined training set gives a multi-valued threshold function which can not be learned by any network defining an algebraic input-output function.

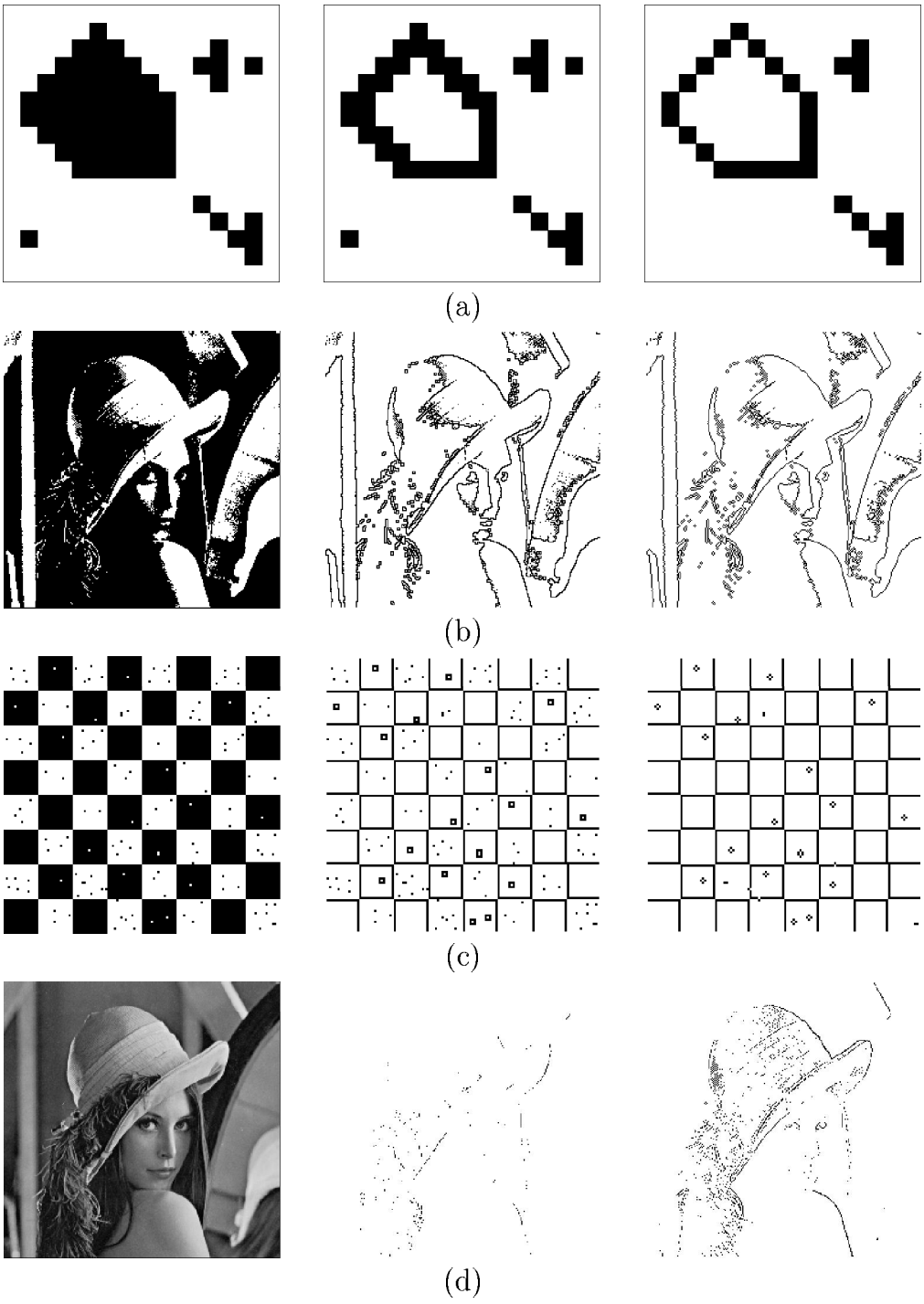


Figure 4.3: The images on the left side are the input images. The images in the middle are the steady-state outputs of the CNN in [5]. The images on the right side are the steady-state outputs of our CNN. (a) A 16x16 binary example. (b) 256x256 7-bit binary Lenna. (c) A 128x128 noisy chessboard. (d) 256x256 gray level Lenna.

5. CONCLUSION

In this thesis, it is shown that PWL and PWC structures are very useful in the solution of classification problems. In fact PWL and PWC models are already used especially in the NNs society however it is not emphasized enough. The simplicity, and the speed of such models do not mean that they are incapable or inefficient.

With a small modification on discrete perceptron, perceptron with input dependent threshold value is introduced. This is one solution step towards the classification of linearly nonseparable sets. The model brings a solution for a subset of linearly nonseparable classification problems. This is obtained by making the threshold value as a function of input in a PWC manner. It is very useful since perceptron learning rule is very fast and efficient learning algorithm which can be used in the design of this NN model.

A new discrete perceptron model forming a cascade structure and being capable of realizing an arbitrary classification task designed by a constructive learning algorithm proposed also in the thesis. It is shown that the proposed cascaded network of multiplexed dual output discrete multilayer perceptron with a new sequential learning algorithm is capable of realizing any given classification task. Convergence of the algorithm is analyzed and the related facts are proved.

Another context of NNs is the design and learning of NNs. Some NN models are obstructed just because there is not any good learning algorithm for those. Discrete Multilayer Perceptron and Cellular Neural Networks can be considered among these. In this study, successful and effective learning algorithms are introduced and analyzed thoroughly. Two of them are mentioned in the previous paragraphs. The third one is the recurrent perceptron learning algorithm which

is an important achievement for the CNNs and its convergence properties are investigated and proven in the thesis. Moreover, it is shown here that the three step learning algorithm can be applied to the threshold class CNNs with input dependent initial states.

Threshold class CNNs with input dependent initial state is another improvement on the plain CNN model. Thus, some linearly nonseparable problems can be solved by using this model while they cannot be solved by regular CNN structure.

All proposed models and learning algorithms are verified by hardware implementations and computer simulations. The results of the examples are given in the thesis. The example applications are just for demonstrative. The applicability and the efficiency of the PWL and PWC NN models and the associated algorithms prove that the proposed models can be used as tools for a wide variety of classification applications where the tasks are inherently linearly nonseparable.

This thesis extend the literature in some directions: i) The discrete perceptron with input-dependent threshold is introduced and is shown to be trained by well known perceptron learning rule to learn some piecewise linearly separable threshold functions such as edge detection in black-white images, ii) It is shown that any kind of linearly nonseparable threshold function can be realized by a cascaded network of multiplexed dual output discrete perceptrons which are also introduced in the thesis and for which a convergent (design) algorithm is proposed in the thesis, iii) an analysis of the recurrent perceptron learning rule for designing cellular neural networks which are a kind of piecewise linear dynamical neural networks is given.

5.1. Recommendations for Future Work

Piecewise linear systems are quite simple models, employing absolution as the unique nonlinear operation in addition to linear ones. They are also powerful

since they can model even the most complicated nonlinear dynamical behaviours such as chaos. Piecewise linear systems are universal systems that can perform any kind of algebraic and dynamical task. It can be argued that piecewise linear systems will be always of interest for their simplicity and universality.

On the other hand, piecewise linear systems has some drawbacks: i) Lacking of differentiability which constitutes an obstacle not only in the numerical and but in the analytical analysis but also in the design of piecewise linear systems. ii) Lacking of efficient methods for constructing piecewise linear models for a given system.

Research on piecewise linear systems continues in the two main streams: One is to show the modeling abilities and usefulness of the piecewise linear systems as implied by their simplicity and universality. Circuits, systems, signal processing, pattern recognition, control and neural networks are some areas which can be given as examples for such type of research areas. The other is to introduce new methods for finding piecewise linear models for a system. Developing compact piecewise linear representations which are more general and/or easier to obtain from a given system and introducing piecewise linear approximation methods more efficient than the available ones.

The piecewise linear models which will be developed in the neural networks domain might be a solution to the parallel hardware realization problem of neural networks. It is a need for developing new piecewise linear models such that they are capable of representing larger classes of functions with greater generalization abilities, having efficient representation, constituting robust approximators rejecting noise and outliers, easy implementation ability in both hardware and software. It is also a need to demonstrate the abilities of piecewise linear models in the new application areas of circuits, systems, signal processing, pattern recognition, control and neural networks.

REFERENCES

[1] **Genç, İ. and Güzeliş, C.**, 1998. Threshold class cellular neural networks with input-dependent initial state, in *5th. IEEE Int. Workshop on Cellular Neural Networks and their Appl.*, London–UK, pp. 130–135.

[2] **Genç, İ. and Güzeliş, C.**, 1998. Discrete perceptron with input dependent threshold value, in *Conference on Signal Processing and its Appl.*, vol. 1, Ankara–Turkey, pp. 36–41, (In Turkish).

[3] **Genç, İ. and Güzeliş, C.**, 2003. A new discrete perceptron model and a learning algorithm, in *Proc. of the Int. Twelfth Turkish Symposium on Artificial Intelligence and Neural Networks – TAINN-03*, vol. E-1, pp. 132–134.

[4] **Karamahmut, S.**, 1994. Two learning algorithms for cellular neural networks and their image processing applications, Master's Thesis, İstanbul Technical Unversity, In Turkish.

[5] **Chua, L.O. and Yang, L.**, 1988. Cellular neural networks: Theory and Applications, *IEEE Trans. on Circuits and Syst.*, 35, 1257–1290.

[6] **Chua, L.O., Desoer, C.A. and Kuh, E.S.**, 1987. Linear and Nonlinear Circuits, McGraw Hill.

[7] **Güzeliş, C. and Göknar, İ.C.**, 1991. A canonical representation for piecewise-affine maps and its applications to circuit analysis, *IEEE Trans. Circuits Syst*, 1342–1354.

[8] **Lin, J.N. and Unbehauen, R.**, 1995. Canonical piecewise-linear networks, *IEEE Trans. on Neural Networks*, 6, 43–50.

[9] **Julián, P., Jordán, M. and Desages, A.**, 1998. Canonical piecewise-linear approximation of smooth functions, *IEEE Trans. on Circuits and Systems-I*, 45, 567–571.

[10] **Chua, L.O. and Kang, S.M.**, 1977. Section-wise piecewise-linear functions: Canonical representation, properties, and applications, *Proc. IEEE*, 65, 915–929.

[11] **Lin, J.N. and Unbehauen, R.**, 1992. Canonical piecewise-linear approximations, *IEEE Trans. on Circuts Syst.*, 39, 697–699.

[12] **Chua, L.O. and Ying, R.**, 1983. Canonical piecewise-linear analysis, *IEEE Trans. Circuits Syst.*, 30, 125–140.

[13] **Blum, E.K. and Li, L.K.**, 1991. Approximation theory and feedforward networks, *Neural Networks*, 4, 511–515.

[14] **Batruni, R.**, 1991. Multilayer neural network with piecewise-linear structure and back-propogation learning, *IEEE Trans. on Neural Networks*, 2, 395–403.

[15] **McCulloch, W.S. and Pitts, W.**, 1943. A logical calculus of ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5, 115–133.

[16] **Rosenblatt, F.**, 1962. Principles of Neurodynamics, Spartan Books, New York.

[17] **Hebb, D.O.**, 1949. The Organization Of Behaviour, Wiley.

[18] **Minsky, M.L.**, 1954. Theory of neural-analog reinforcement systems and its application to the brain-model problem, PhD Thesis, Princton University.

[19] **Rosenblatt, F.**, 1958. The perceptron: A probabilistic model for information storage and organization in the brain, *Psycological Review*, 58, 386–408.

[20] **Rosenblatt, F.**, 1962. Principles of Neurodynamics, Spartan Books, New York.

[21] **Widrow, B.**, 1962. Generalization and information storage in networks of adeline 'neurons', in *Self-Organizing Systems*, Eds. Yowitz, M., Jacobi, G. and Goldstein, G., pp. 435–461.

[22] **Minsky, M. and Papert, S.**, 1969. Perceptrons, An Introduction to Computational Geometry, MIT, 1st edn.

[23] **Minsky, M. and Papert, S.**, 1990. Perceptrons, An Introduction to Computational Geometry, MIT, 2nd edn.

[24] **Grossberg, S.**, 1972. Neural expectation: Cerebellar and retinal analogs of cells fired by learnable or unlearned pattern classes, *Kybernetik*, 10, 49–57.

[25] **Grossberg, S.**, 1980. How does a brain build a cognitive code?, *Psychological Review*, 87, 1–51.

[26] **Hopfield, J.**, 1982. Neural networks and physical systems with emergent collective computational abilities, *Proc. of the National Academy of Sciences, USA*, 79, 2554–2558.

[27] **Kohonen, T.**, 1990. The self organizing map, *Proc. of the IEEE*, 78, 1464–1480.

[28] **Rumelhart, D., Hinton, G. and Williams, R.**, 1986. Learning representations of back-propagation errors, *Nature*, 323, 533–536.

[29] **Broomhead, D. and Lowe, D.**, 1988. Multivariable functional interpolation and adaptive networks, *Complex Systems*, 2, 321–355.

[30] **Güzeliş, C. and Chua, L.O.**, 1993. Stability analysis of generalized cellular neural networks, *Int. J. Circuit Theory and Appl.*, 21, 1–33.

[31] **Cortes, C. and Vapnik, V.**, 1995. Support vector networks, *Machine Learning*, 20, 273–297.

[32] **Amari, S.I.**, 1972. Characteristics of random nets of analog neuron-like elements, *IEEE T. on Systems Man. and Cybernetics*, SMC-2, 643–657.

[33] **Gelenbe, E.**, 1989. Random neural networks with positive and negative signals and product form solution, *Neural Computation*, 1, 502–510.

[34] **Hodgkin, A. and Huxley, A.**, 1952. A quantitative description of membrane current and its application to conduction and exitation in nerve, *J. Physiol.*, 117, 500–544.

[35] **Fukushima, K., Miyake, S. and Ito, T.**, 1975. Cognitron: A self organizing multi-layered neural network, *Biological Cybernetics*, 20, 11–136.

[36] **Grossberg, S.**, 1967. Nonlinear difference —differential equations in prediction and learning theory, *Proceedings of the National Academy of Sciences*, 58, 1329–1334.

[37] **Freeman, W.**, 1987. Simulation of chaotic eeg patterns with a dynamic model of the olfactory system, *Biological Cybernetics*, 56, 139–150.

[38] **Roychowdhury, V.P., Siu, K.Y. and Kailath, T.**, 1995. Classification of linearly nonseparable patterns by linear threshold elements, *IEEE Trans. on Neural Networks*, 6, 318–331.

[39] **Anlauf, J. and Biehl, M.**, 1989. The AdaTron: An adaptive perceptron algorithm, *Europhysics Letters*, 10, 687–692.

[40] **Marchand, M., Golea, M. and Ruján, P.**, 1990. A convergence theorem for sequential learning in two-layer perceptrons, *Europhys. Lett.*, 11, 487–492.

[41] **Young, S. and Downs, T.**, 1998. CARVE—a constructive algorithm for real-valued examples, *IEEE Trans. on Neural Networks*, 9, 1180–1190.

[42] **Martinelli, G., Mascioli, F.M. and Bei, G.**, 1993. Cascade neural network for binary mapping, *IEEE Trans. on Neural Networks*, 4, 148–150.

[43] **Martinelli, G. and Mascioli, F.M.**, 1992. Cascade perceptron, *Electronics Letters*, 28, 947–949.

[44] **Qinruo, W., Bo, Y., Yun, X. and Bingru, L.**, 2003. The hardware structure design of perceptron with FPGA implementation, in *IEEE Int. Conf on Systems, Man and Cybernetics*, vol. 1, pp. 762–767.

[45] **Şahin, S., Becerikli, Y. and Yazıcı, S.**, 2006. Learning internal representations by error propagation, in *Neural Information Processing: 13th Int. Conf. ICONIP'06*, Eds. King, I., Wang, J., Chan, L.W. and Wang, D., pp. 1105 – 1112, Springer-Verlag.

[46] **Ferrer, D., Gonzàlez, R., Fleitas, R., Acle, J.P. and Canetti, R.**, 2004. NeuroFPGA – Implementing artificial neural networks on programmable logic devices, in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Designers Forum (DATE04).*

[47] **Maeda, Y. and Wakamura, M.**, 2005. Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation, *IEEE Trans. on Neural Networks*, 16, 1664–1672.

[48] **Cadenas, O., Megson, G. and Jones, D.**, 2005. A new organization for a perceptron-based branch predictor and its FPGA implementation, in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design.*

[49] **Vitabile, S., Conti, V., Gennaro, F. and Sorbello, F.**, 2005. Efficient MLP digital implementation on FPGA, in *Proceedings of the 2005 8th Euromicro conference on Digital System Design (DSD05).*

[50] **Liu, J. and Liang, D.**, 2005. A survey of FPGA-based hardware implementation of ANNs, in *Proceedings of ICNN&B International Conference on Neural Networks and Brain*, pp. 915–918.

[51] **Muselli, M.**, 1997. On convergence properties of pocket algorithm, *IEEE Trans. on Neural Networks*, 8, 623–629.

[52] **Rosen-Zvi, M. and Kanter, I.**, 2001. Training a perceptron in a discrete weight space, *Physical Review E*, 64, 046109–1–9.

[53] **Rumelhart, D., Hinton, G. and Williams, R.**, 1986. Learning internal representations by error propagation, in *Parallel Distributer Processing: Exploration in the Microstructure of Cognition*, Eds. Rumelhart, D. and McClelland, J., Chap. 3.

[54] **Mézard, M. and Nadal, J.P.**, 1989. Learning in feedforward layered networks: The tiling algorithm, *J. Phys. A: Math. Gen.*, 22, 2191–2203.

[55] **Rujan, P. and Marchand, M.**, 1989. Learning by minimizing resources in neural networks, *Complex Syst.*, 3, 229–241.

[56] **Wieland, A.** Two spirals, *http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/ neural/bench/cmu/0.html*, Current as of September 2004.

[57] **Fahlman, S. and Lebiere, C.**, 1990. The cascade-correlation learning architecture, in *Advances Neural Inform. Processing Syst. (NIPS)*, vol. 2, San Mateo, CA, pp. 524–532.

[58] **Shirazi, N., Walters, A. and Athanas, P.**, 1995. Quantitative analysis of floating point arithmetic on fpga based custom computing machines, in *In Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 155–162.

[59] **Wust, H., Kasper, K. and Reininger, H.**, 1998. Hybrid number representation for the fpga-realization of a versatile neuro-processor, in *Proceedings of 24th Euromicro Conference*, vol. 2, pp. 694–701.

[60] **Altera Corp.**, 2007. Quartus II Development Software Handbook, vol. 1–5, California.

[61] **Güzeliş, C. and Karamahmut, S.**, 1994. Recurrent perceptron learning algorithm for completely stable cellular neural networks, in *3rd. IEEE Int. Workshop on Cellular Neural Networks and their Appl.*, pp. 177–182.

[62] **Güzeliş, C., Karamahmut, S. and Genç, İ.**, 1999. A recurrent perceptron learning algorithm for cellular neural networks, *ARI*, 51, 296–309.

[63] **Vanderberghe, L. and Vandewalle, J.**, 1989. Application of relaxation methods to the adaptive training of neural networks, in *Proc. Math. Theory of Networks and Systems, MTNS'89*, Amsterdam.

[64] **Zou, F., Schwartz, S. and Nossek, J.A.**, 1990. Cellular neural network design using a learning algorithm, in *1st. IEEE Int. Workshop on Cellular Neural Networks and their Appl.*, Amsterdam, pp. 73–81.

[65] **Nossek, J.A., Seiler, G., Roska, T. and o. Chua, L.**, 1992. Cellular neural networks: Theory and circuit design, *Int. J. of Circuit Theory and Appl.*, 20, 533–553.

[66] **Nossek, J.A.**, 1996. Design and learning with cellular neural networks, *Int. J. of Circuit Theory and Appl.*, 24, 15–24.

[67] **Chua, L.O. and Shi, B.E.**, 1991. Multiple layer cellular neural networks: A tutorial, in *Algorithms and Parallel VLSI Architecture*, Eds. Deprette, F. and der Veen, A.V., vol. A, pp. 137–168, Elsevier.

[68] **Chua, L.O. and Thiran, P.**, 1991. An analytical method for designing simple cellular neural networks, *IEEE Trans. on Circuits and Syst.*, 38, 1332–1341.

[69] **Kozek, T., Roska, T. and Chua, L.O.**, 1993. Genetic algorithm for CNN template learning, *IEEE Trans. on Circuits and Syst.*, 40, 392–402.

[70] **Schuler, A.J., Nachbar, P., Nossek, J.A. and Chua, L.O.**, 1992. Learning state space trajectories in cellular neural networks, in *2nd. IEEE Int. Workshop on Cellular Neural networks and their Appl.*, pp. 68–73.

[71] **Güzeliş, C.**, 1992. Supervised learning of the steady-state outputs in generalized cellular neural networks, in *2nd. IEEE Int. Workshop on Cellular Neural Networks and their Appl.*, pp. 74–79.

[72] **Balsi, M.**, 1993. Recurrent back-propagation for cellular neural networks, in *European Conf. on Circuit Theory and Design*, Losanne–Swiss, pp. 677–682.

[73] **Schuler, A.J., Nachbar, P. and Nossek, J.A.**, 1993. State-based backpropagation-through-time for CNNs, in *European Conf. on Circuit Theory and Design*, pp. 33–38.

[74] **Lu, Z. and Liu, D.**, 1998. A new synthesis approach for a class of cellular neural networks based with space-invariant cloning template, *IEEE Trans. on Circuits and Systems-II*, 45, 1601–1605.

[75] **Zarándy, Á.**, 1999. The art of CNN template design, *Int. J. of Circuit Theory and Appl.*, 27, 5–23.

[76] **de Souza, S.X., Yalçın, M.E., Suykens, J.A. and Vandewalle, J.**, 2004. Toward CNN chip-specific robustness, *IEEE Trans. on Circuits and Systems-I*, 51, 892–902.

[77] **Magnussen, H. and Nossek, J.A.**, 1992. Towards a learning algorithm for discrete-time cellular neural networks, in *2nd. IEEE Int. Workshop on Cellular Neural Networks and their Appl.*, pp. 80–85.

[78] **Pineda, F.J.**, 1988. Generalization of backpropagation to recurrent and higher order neural networks, in *Neural Information Processing Systems*, Ed. Anderson, D.Z., pp. 602–611, American Inst. of Phys., New York.

[79] **Balsi, M.**, 1992. Generalized CNN: Potentials of a CNN with non-uniform weights, in *2nd IEEE Int. Workshop on Cellular Neural Networks and their Appl.*, pp. 129–134.

[80] **Yalçın, M.E. and Güzeliş, C.**, 1996. CNNs with radial basis input function, in *4th. Int. Workshop on Cellular Neural Networks and their Appl.*, Seville–Spain, pp. 231–236.

[81] **Güzeliş, C. and Günsel, B.**, 1995. Cellular neural networks for early vision, in *European Conf. on Circuit Theory and Design*, pp. 785–788.

[82] **Günsel, B. and Güzeliş, C.**, 1995. Supervised learning of smoothing parameters in image restoration by regularization under cellular neural networks framework, in *IEEE Int. Conf. Image Processing*, pp. 470–473.

[83] **Hanggi, M. and Moschytz, G.**, 1999. An exact and direct analytical method for the design of optimally robust CNN templates, *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, 46.

[84] **Hanggi, M. and Moschytz, G.**, 1999. Analytic and VLSI specific design of robust CNN templates, *Journal of VLSI Signal Processing*, 23, 415–427.

[85] **Mirzai, B., Reutemann, R., Rüegg, M. and Moschytz, G.S.**, 1997. Isolated word recognition utilizing a CNN encoder, in *European Conf. on Circuit Theory and Design*, Budapest–Hungary, pp. 615–620.

[86] **Shou, Y. and Lin, C.**, 2004. Image descreening by GA-CNN-based texture classification, *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, 51.

CURRICULUM VITAE

İbrahim Genç was born on 1971 in Hereke. He received the high school diploma from Ankara Science Lycée in 1988. He graduated from İstanbul Technical University Electronics and Communicaiton Engineering Department in 1992. He has the Master of Science Degree on Biomedical Engineering from İstanbul Technical University. He worked as a research and teaching assistant at Ondokuz Mayıs University Electrical and Electronics Engineering Department between 1994 and 2000. He worked as a senior researcher at TÜBİTAK Marmara Research Center, Information Technologies Research Institute between 2000 and 2005. He works as Software Projects Director at C Tech Bil. Tek. San. ve Tic. A.Ş. since 2005 and currently he is a member of the Executive Board. He has been honoured by TÜBİTAK Münir Birsel Foundation with doctoral scholarship.