

39613

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

39613

**HÜCRESEL YAPAY SİNİR AĞLARI İÇİN İKİ ÖĞRENME
ALGORİTMASI VE GÖRÜNTÜ İŞLEME UYGULAMALARI**

YÜKSEK LİSANS TEZİ

Müh. Sinan KARAMAHMUT

Tezin Enstitüye Verildiği Tarih : 13 Haziran 1994

Tezin Savunulduğu Tarih : 4 Temmuz 1994

Tez Danışmanı : Doç. Dr. Cüneyt Güzelış

Diğer Jüri Üyeleri : Prof. Dr. Cem Gökmar

Prof. Dr. Uğur Çilingiroğlu

**E.Ö. YÜSEKÖĞRETİM KURULU
DÖNÜMANTASYON MERKEZİ**

TEMMUZ 1994

ÖNSÖZ

Bu çalışmada beni destekleyen, özellikle kuramsal açıdan tezime önemli katkılarda bulunan değerli hocam Doç. Dr. Cüneyt Güzeliş 'e teşekkür ederim.

Ayrıca bana çalışmam boyunca manevi destekte bulunan Derya İshakoğlu Vardal, Murat Bülbül, Volkan Sezer, Mehmet Devrim Azak ve aileme de teşekkürü bir borç bilirim.

Haziran 1994

Sinan Karamahmut



İÇİNDEKİLER

ŞEKİL LİSTESİ	v
ÖZET	vi
SUMMARY	vii
BÖLÜM 1 GİRİŞ	1
BÖLÜM 2 YAPAY SİNİR AĞLARI	3
2.1 Yapay Sinir Ağlarının bir Tanımı ve Özellikleri	5
2.2 Yapay Sinir Ağlarının Üstünlükleri	6
2.3 Önemli 15 Yapay Sinir Ağı Modeli	7
BÖLÜM 3 HÜCRESEL YAPAY SİNİR	10
3.1 Hücresel Yapay Sinir Ağlarının Yapısal Özellikleri	10
3.2 Hücresel Yapay Sinir Ağlarının Görüntü İşleme Uygulamaları ..	15
BÖLÜM 4 ÖĞRENME	17
4.1 Yapay Sinir Ağlarda Öğrenme	18
4.2 Eğitici ve Eğitici Olmayan Öğrenme	18
BÖLÜM 5 YAPAY SİNİR AĞLARINDA ÖĞRENME ALGORİTMALARI ..	20
5.1 Hebb Öğrenme Algoritması	20
5.2 Algılayıcı Öğrenme Algoritması	21
5.3 Eğilim Öğrenme Algoritması	21
5.4 Widrow-Hoff Öğrenme Algoritması	22
5.5 İnti Öğrenme Algoritması	22
5.6 Kazanan-Herşeyi-Alır Öğrenme Algoritması	23
5.7 " Outstar " Öğrenme Algoritması	23
BÖLÜM 6 HÜCRESEL YAPAY SİNİR AĞLAR İÇİN DİNAMİK ÖĞRENME ALGORİTMALARI	24
6.1 Dinamik Geriye-Yayılım Öğrenme Algoritması	25
6.2 Dinamik Algılayıcı Öğrenme Algoritması	30

BÖLÜM 7 SLAT PROGRAMI KULLANIM KILAVUZU	34
7.1 Giriş	34
7.2 Gerekli Bilgisayar Donanımı	34
7.3 Programın İşleyişi	35
7.4 Ekran Görüntüsü	35
7.4.1 Menü Bloğu	35
7.4.2 Şablon Bloğu	38
7.4.3 Görüntü Bloğu	38
7.4.4 Veri Bloğu	38
7.5 Başlangıç Ayarları	39
7.6 Kütüphaneler	39
7.6.1 Eğitim Kümesi Kütüphanesi	39
7.6.2 Şablon Kütüphanesi	40
7.6.3 Görüntü Kütüphanesi	41
BÖLÜM 8 RBLA VE RPLA İLE GÖRÜNTÜ İŞLEME	43
8.1 RBLA ile Görüntü İşleme	43
8.2 RPLA ile Görüntü İşleme	45
SONUÇLAR VE ÖNERİLER	56
KAYNAKLAR	57
EK SLAT PROGRAMI	60
ÖZGEÇMİŞ	106

ŞEKİL LİSTESİ

Şekil 2.1	Hücrelerin Genel Yapısı	6
Şekil 3.1.1	Bir $C(i, j)$ Hücresinin İç Yapısı	11
Şekil 3.1.2	$f(\bullet)$ Karakteristiği	12
Şekil 6.1.a	$\beta = 3$ için Çıkış Fonksiyonu	27
Şekil 6.1.b	$\beta = 10$ için Çıkış Fonksiyonu	27
Şekil 7.1	SLAT Programının Ekran Görüntüsü	36
Şekil 7.2	SLAT.cfg Dosyası	39
Şekil 7.3	Eğitim Kümesi Örneği	40
Şekil 7.4	Şablon Dosyası Örneği	41
Şekil 7.5	Görüntü Dosyası Örneği	42
Şekil 7.6	EDCNI Görüntüsü	42
Şekil 8.1	RBLA ile Kenar Saptama	50
Şekil 8.2	Test Görüntüleri	51
Şekil 8.3	Test Görüntülerinin Kenarları	51
Şekil 8.4	Test Görüntülerinin Köşeleri	51
Şekil 8.5	RPLA ile Kenar Saptama	52
Şekil 8.6	RPLA ile Köşe Saptama	53
Şekil 8.7	RPLA ile Boşluk Doldurma	54

ÖZET

Bu tezde, tam kararlı Hücresel Yapay Sinir Ağları (HYSA) için istenen kararlı hal çıkışlarının eğitici öğrenilmesi amacıyla yönelik olarak Dinamik Algılayıcı Öğrenme Algoritması ("Recurrent Perceptron Learning Algorithm : RPLA ") ve Dinamik Geriye-Yayılım Öğrenme Algoritması (" Recurrent Backpropagation Learning Algorithm : RBLA ") adında iki farklı eğitici öğrenme algoritması geliştirilmiştir. Bu algoritmaları kullanan, HYSA 'lar için şablon öğrenme programı olan SLAT (" Supervised Learning Algorithm Tool ") programı yazılarak, bir kişisel bilgisayar ortamında benzetim düzeni elde edilmiştir.

Geliştirilen algoritmalar, HYSA 'nın eğitici öğrenmesinin bir amaç ölçütünün tasarım kısıtlamaları altında enazını bulma sorununa dönüştürülmesine dayanmaktadır. Enazlanmak istenen amaç ölçütü, istenen kararlı-durum çıkışları ile gerçek çıkışlar arasındaki Öklid uzaklığını veren hata işlevidir. Tasarım kısıtlamaları ise HYSA 'nın tam-kararlı olmasını, yani kararlı-durumda tüm yörüngelerinin denge noktalarından birine yerleşmesini sağlayan bağlantı ağırlıklarının simetrik olma koşulu ile kararlı-durum çıkışlarının iki-kutuplu (± 1) olmasını sağlayan çıkıştan öz-geribesleme bağlantı ağırlığının, durumdan öz-geribesleme katsayısından büyük olma koşulundan oluşmaktadır.

Geliştirilen algoritmalar RBLA, hata işlevinin kısıtlamalar altında yerel enaz noktalarından birinin yeterince küçük seçilen öğrenme oranları durumunda veren, izdüşüm türünden bir eğim-düşme enazlama algoritmasıdır. RBLA, Hopfield ağı için verilmiş olan Geriye-Yayılım Algoritmasının tam-kararlı HYSA için bir uygulamasıdır. Tam-bağılantılı bir ağ olan Hopfield ağında gerekli bağlantı ağırlık katsayılarının sayısı HYSA 'daki yerel ve düzgün bağlantıyı belirleyen şablon katsayılarının sayısına göre çok fazla olduğundan, önerilen RBLA, Hopfield ağı için önerilen algoritmaya göre, bu açıdan daha etkindir. HYSA 'ları için tezde önerilen RBLA 'nın, çıkış işlevinin doymaya yakın bölgelerde çok düşük türevleri olması yüzünden yavaş yakınsama sorunları vardır. Bu sorunlar tezde geliştirilen çeşitli teknikler ile yenilmeye çalışılmıştır.

RPLA ise, HYSA 'nın her bir hücresinin kararlı-durumda hücreye giren toplam girişe bağlı olarak çıkışında $+1$ ya da -1 veren Algılayıcı ("Perceptron") benzeri bir işlem elemanı olarak çalışması gözlemine dayanarak geliştirilmiştir. RPLA, tam-kararlı HYSA 'nın eğitici öğrenmesi için geliştirilmiş bilinen en etkin öğrenme algoritmasıdır.

SUMMARY

TWO LEARNING ALGORITHMS FOR CELLULAR NEURAL NETWORKS AND THEIR IMAGE PROCESSING APPLICATIONS

In this thesis, two supervised learning algorithms for obtaining the template coefficients in completely stable Cellular Neural Networks (CNNs) are presented. The Recurrent Backpropagation Learning Algorithm (RBLA) can be viewed as the CNN version of the well-known Recurrent Backpropagation Algorithm originally developed for a Hopfield type completely stable continuous-time dynamical neural network. The second algorithm presented is inspired by the analogy between the input-output relation of the well-known Perceptron and the steady-state behavior of the cells in CNNs. It is hence called as Recurrent Perceptron Learning Algorithm (RPLA) since applied to a dynamical network, CNN.

From the advent of the first useful computer (ENIAC) in 1946 until the late 1980s, essentially all information processing applications used a single basic approach: programmed computing. Solving a problem using programmed computing involves devising an algorithm and/or a set of rules for solving the problem and then correctly coding these in software and making necessary revisions and improvements.

Clearly, programmed computing can be used in only those cases where the processing to be accomplished can be described in terms of a known procedure or a known set of rules. If the required algorithmic procedure and/or set of rules are not known, then they must be developed an undertaking that, in general, has been found to be costly and time consuming. In fact, if the algorithm required is not simple (which is frequently the case with the most desirable capabilities), the development process may have to await a flash of insight. Obviously, such an innovation process cannot be accurately planned or controlled. Even when the required algorithm or rule set can be devised, the problem of software development still must be faced.

Because current computers operate on a totally logical basis, software must virtually perfect if it is to work. The exhaustive design, testing, and iterative improvement that software development demands makes it a lengthy and expensive process.

A new approach to information processing that does not require algorithm or rule development and that often significantly reduces the quantity of software that must be developed has recently become available. This approach, called neurocomputing, allows, for some types of problems (typically in areas such as sensor processing, image processing, pattern recognition, data analysis, and control), the development of information processing capabilities for which the algorithms or rules

are not known (or where they might be known, but where the software to implement them would be too expensive, time consuming, or inconvenient to develop). For those information processing operations amenable to neurocomputing implementation, the software that must be developed is typically for relatively straightforward operations such as data file input and output, peripheral device interface, preprocessing, and postprocessing. The Computer Aided Software Engineering (CASE) tools often used with neurocomputing systems can frequently be utilized to build these routine software modules in a few hours. These properties make neurocomputing an interesting alternative to programmed computing, at least in those areas where it is applicable.

Formally, neurocomputing is the technological discipline concerned with parallel, distributed, adaptive information processing systems that develop information environment. The primary information processing structures of interest in neurocomputing are artificial neural networks (although other classes of adaptive information processing structures are sometimes also considered, such as learning automata, genetic learning systems, data-adaptive content addressable memories, simulated annealing systems, associative memories, and fuzzy learning systems).

Artificial neural systems function as parallel distributed computing networks. Their most basic characteristics is their architecture. Only some of the networks provide instantaneous responses. Other networks need time to respond and are characterized by their time-domain behavior, which we often refer to as dynamics. Neural network also differ from each other in their learning modes. There are a variety of learning rules that establish when and how the connecting weights change. Finally, networks exhibit different speeds and efficiency of learning. As a result, they also differ in their ability to accurately respond to the cues presented at the input.

In contrast to conventional computers, which are programmed to perform specific tasks, most neural networks must be taught, or trained. Learning corresponds to parameter changes. Learning rules and algorithms used for experiential training of networks replace the programming required for conventional computation. Neural network users do not specify an algorithm to be executed by each computing node as would programmers of a more traditional machine. Instead, they select what in their view is the best architecture, specify the characteristics of the neurons and initial weights, and choose the training mode for the network. Appropriate inputs are then applied to the networks so that it can acquire knowledge from the environment. As a result of such exposure, the network assimilates the information that can later be recalled by the user.

In the past three decades a number of neural network architectures have been developed. The architectures have been inspired both by the principles governing biological neural systems and the well-established theories of engineering and fundamental sciences. Most of the widely applied neural networks fall into two main classes: 1) memoryless neural networks and 2) dynamical neural networks. From a circuit theoretical point of view, the memoryless neural networks are non-linear resistive circuits, while the dynamical neural networks are non-linear R-L-C circuits. A memoryless neural network defines a non-linear transformation from the space of input signals into the space of output signals. Such networks have been successfully used in pattern recognition and several problems which can be defined as a non-linear transformation between two spaces. As in the Hopfield network and Cellular Neural Network, dynamical neural networks have usually been designed as dynamical

systems where the inputs are set of some constant values and each trajectory approaches one of the stable equilibrium points depending upon the initial state. Some useful application of these networks includes image processing, pattern recognition and optimization.

Due to the grid topology of Cellular Neural Networks which is tailor made for image processing, this artificial neural network model is considered in this thesis.

A Cellular Neural Network is a 2-dimensional array of cells [1]. 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 defined by the following metric:

$$d(i, j; \hat{i}, \hat{j}) = \max\{|i - \hat{i}|, |j - \hat{j}|\}.$$

Where (i, j) is the vector of integers indexing the cell $C(i, j)$ in the i *th* row j *th* column of the 2-dimensional array. The system of equations describing a CNN with the neighborhood size of one is given in (1)-(2).

$$\dot{x}_{i,j} = -A x_{i,j} + \sum_{p,l \in \{-1,0,1\}} w_{p,l} y_{i+p,j+l}(\infty) + \sum_{p,l \in \{-1,0,1\}} z_{p,l} u_{i+p,j+l} + I. \quad (1)$$

$$y_{i,j} = f(x_{i,j}) = \frac{1}{2} \left\{ |x_{i,j} + 1| - |x_{i,j} - 1| \right\}. \quad (2)$$

Where, $A, I, w_{p,l}$ and $z_{p,l} \in \mathbb{R}$ are constants.

It is known in [1] that a CNN is completely stable if the feedback connection weights $w_{p,l}$ are symmetric. Throughout the thesis, the input connection weights $z_{p,l}$ are chosen symmetric for reducing computational costs while the feedback connection weights $w_{p,l}$ are chosen symmetric for ensuring the complete stability, i.e.,

$$w_{-1,-1} = w_{1,1} \stackrel{\text{def}}{=} a_1, \quad w_{-1,0} = w_{1,0} \stackrel{\text{def}}{=} a_2, \quad w_{-1,1} = w_{1,-1} \stackrel{\text{def}}{=} a_3,$$

$$w_{0,-1} = w_{0,1} \stackrel{\text{def}}{=} a_4, \quad w_{0,0} \stackrel{\text{def}}{=} a_5;$$

$$z_{-1,-1} = z_{1,1} \stackrel{\text{def}}{=} b_1, \quad z_{-1,0} = z_{1,0} \stackrel{\text{def}}{=} b_2, \quad z_{-1,1} = z_{1,-1} \stackrel{\text{def}}{=} b_3, \quad z_{0,-1} = z_{0,1} \stackrel{\text{def}}{=} b_4, \quad z_{0,0} \stackrel{\text{def}}{=} b_5.$$

In this thesis, the learning is accomplished through modification of the following weight vector $w \in \mathbb{R}^{11}$.

$$w \stackrel{\text{def}}{=} [a^T b^T I] \stackrel{\text{def}}{=} [a_1 a_2 a_3 a_4 a_5 b_1 b_2 b_3 b_4 b_5 I]^T. \quad (3)$$

Several design methods for determining the entries of w , i.e., the feedback template coefficients a_i 's, the input template coefficients b_i 's, and the threshold I are proposed in the literature [1], [3], [4]. The well-known relaxation methods for solving linear inequalities are used in [3]-[4] for finding one of the connection weights providing that the desired outputs can be obtained as the actual outputs for the given external inputs and for the properly chosen initial states. However, the methods in [3]-[4] do not specify which initial state vectors, except for the following trivial one, yield the desired output for the given external input and the weight vector found. The trivial solution in the determination of such a proper initial state vector is to take the desired output as the initial state; but this requires the knowledge of the desired output which is not available for the external inputs outside the training set. For this reason, it is still desired to develop new design methods and supervised learning algorithms for finding the connection weights yielding the desired output for the given external input and the chosen initial state. The supervised learning algorithms in this thesis are proposed for such a purpose.

Recently, a number of supervised learning algorithms for CNNs are given in the literature [2], [7]-[9], [11]-[14]. The well-known Backpropagation through Time Algorithm is applied in [7] for learning the desired trajectories in continuous-time CNNs. A modified Alternating Variable Method is used in [8] for learning the steady-state outputs in discrete-time CNNs. Both of these algorithms are proposed to be used for any kind of CNN and hence they do not take into account the constraints which are needed to be imposed on the connection weights for ensuring the complete stability and the bipolarity of the steady-state outputs.

It is shown in [9] that the supervised learning of the steady-state outputs in completely stable generalized CNNs [10] is a constrained optimization problem, where the objective function is the error function and the constraints are due to some desired qualitative and quantitative design requirements such as the bipolarity of the steady-state outputs and the complete stability. The algorithm given in [9] is a gradient-descent algorithm and indeed it is an extension of the recurrent backpropagation algorithm to the generalized CNNs. The recurrent backpropagation is applied also in [11]-[12] to a modified version of CNNs differing from the original CNN model in the following respects: i) The cells are fully-connected, ii) The output function is a differentiable sigmoidal one, and iii) The network is designed as a globally asymptotically stable network.

In a very recent paper [13], 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. It is assumed in [13] that the steady-state values of the states have the derivative with respect to the connection weights. However, this assumption is true only if the magnitudes of the states are all strictly greater than one. Therefore, the gradient calculated in [13] does

not describe the jumping of the steady-state values of the states from one saturation region to the other as a consequence of the changes in the connection weights.

The lack of the derivative of the error function prevents the use of the gradient-based methods for finding the templates minimizing the error. In order to overcome this problem, the output function can be changed [14] with a continuously differentiable one which is very close to the original piecewise-linear function. Whereas the gradient methods are now applicable, the error surfaces have almost flat portions resulting in extremely slow convergence [14]. An alternative solution to this problem is to use methods not requiring the derivative of the error function. Such a method is given in [2] by introducing genetic optimization algorithms for supervised learning of the optimal template coefficients.

The Recurrent Backpropagation Learning Algorithm (RPLA) given in this thesis is a special case of the one given in [30] and also can be viewed as the CNN version of the Recurrent Backpropagation Algorithm. The algorithm finds the optimal template coefficients and the threshold minimizing the total error function while satisfying the constraints i) $w_{0,0} = a_5 > A$, and ii) the symmetry conditions for the feedback template coefficients ; where the first constraint ensures the bipolarity of the steady-state outputs and the second the completely stability in CNNs.

The Recurrent Perceptron Learning Algorithm (RPLA) proposed in this thesis does not need the derivative of the error function. It is developed for finding the template coefficients of a CNN to realize an input-(steady-state)output map 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. The algorithm resembles the well-known Perceptron learning algorithm [16] and hence called as Recurrent Perceptron Learning Algorithm (RPLA) for CNNs. RPLA starts with an initial weight vector satisfying the constraint $w_{0,0} = a_5 > A$ ensuring the bipolarity of the steady-state output. The updated weight vector obtained in each step of the algorithm is projected onto the constraint set $A = \{ w \in \mathbb{R}^{11} \stackrel{\text{def}}{=} |a_5 > A \}$. It is shown in the thesis that if there is a weight vector which satisfies the bipolarity constraint and yields the zero Output Mismatching Error defined in (4), then RPLA finds this vector with a suitable time-varying learning-rate.

$$E(w) = \sum_{i,j} y_{i,j}(\infty)(y_{i,j}(\infty) - d_{i,j}) = \sum_{(i,j) \in D^+} y_{i,j}(\infty) - \sum_{(i,j) \in D^-} y_{i,j}(\infty) \quad (4)$$

$$\text{Where, } D^+ = \{(i, j) \mid y_{i,j}(\infty) = -d_{i,j} = 1\} \text{ and } D^- = \{(i, j) \mid y_{i,j}(\infty) = -d_{i,j} = -1\}$$

It is assumed here that the state vector is never chosen equal to one of the equilibrium points in the center region or partial regions in the state space and therefore any steady-state output is either +1 or -1.

The proposed algorithms RBLA and RPLA have been applied to several image processing problem. It has been observed that CNNs using the rule defined by RBLA and RPLA can learn edge detection, corner detection, and hole filling tasks very quickly.

The structure of the thesis is as follows. Chapter 2 gives an introduction to artificial neural networks. The general structure of CNNs is presented in Chapter 3. In order to put the work which is carried out by this thesis into a more understandable way a general information on learning and different learning algorithms for artificial neural networks are discussed in Chapter 4 and 5, respectively. In Chapter 6, the Recurrent Backpropagation Learning Algorithm (RBLA) and the Recurrent Perceptron Learning Algorithm (RPLA) are proposed. Chapter 7 is a guide for the simulation tool SLAT (Supervised Learning Algorithm Tool) which uses RBLA and RPLA. The simulation results of SLAT are presented in Chapter 8 .



BÖLÜM 1

GİRİŞ

1940'lı yılların başında McCulloch, Pits, Hebb, ve Rosenblatt 'un çabalarıyla, alışılagelmiş bilgi işleme sistemlerine bir seçenek olarak ilk yapay sinir ağı (YSA) modelleri geliştirilmeye başlanmıştır. Geçen 50 yıl boyunca birçok YSA modeli geliştirilmiştir [1],[2]. Bunların en tanınmışları; Grossberg 'in Uyarlanmalı Rezonans Kuramı (" Adaptive Resonance Theory : ART ") modelleri, Kohonen 'in Öz-Örgütlenmeli Haritası (" Self-Organizing Feature Map "), Hopfield Ağı, Çok-Katmanlı Algılayıcı (" Multilayer Perceptron ") ve Widrow 'un Uyarlanmalı Doğrusal Hücre (" ADaptive LİNeer Element : ADALİNE ") modelleridir. Geliştirilen YSA modelleri, hem biyolojik sinir sistemlerini yöneten ilkelerden hareketle, hem de mühendislik ve temel bilimlerin oturmuş kuramlarından esinlenerek elde edilmişlerdir.

1988 yılında Chua ve Yang, geniş çapta tümleşik devre olarak gerçeklemeye uygun ve işaret işlemede yeni olanaklar tanıyan Hücresel Yapay Sinir Ağı (HYSA) ("Cellular Neural Networks : CNN") modelini önermişlerdir [3],[4]. HYSA 'nın karmaşık bilgi işleme görevlerini gerçekleyebileceği [5] ve özellikle kenar, köşe saptama, gürültü süzme gibi alçak-düzeyle görüntü işleme işlevlerini yerine getirmede çok başarılı olduğu, yapılan araştırmalar ile gösterilmiştir [4], [6], [7], [8], [9]. HYSA uygulamalarında temel sorun, istenen görüntü işlemeyi yapacak olan hücreler arası bağlantı ağırlık katsayılarının tasarlanmasıdır. Bu sorunun çözümü için değişik yöntemler kullanılmıştır. Verilen ilk tasarım yönteminde [4], alışılagelmiş görüntü işleme yöntemlerinden esinlenilmiştir. İkinci türden tasarım yöntemleri [6]-[9] ise, bağlantı ağırlıklarının verilen giriş-çıkış işlevini yerine getirmek üzere analitik yollar ile elde edilmesine dayanmaktadır. Bağlantı ağırlık katsayılarının bulunması için kullanılan tüm bu yöntemler genelleştirilemediği ve çoğu durumda etkin olmadıklarından, HYSA 'ların tasarımına yönelik yeni yöntemlerin geliştirilmesi gereği doğmuştur [10]-[13]. Belirtilen tasarım gerekleri yanında, özel giriş-çıkış çiftlerine daha etkin sonuçlar elde edebilmek için HYSA 'nın verilen işlevi öğrenmesine yönelik algoritmaların geliştirilmesi önem kazanmıştır. Şu ana kadar geliştirilen öğrenme

algoritmalarının hepsi, ağı tam kararlı olması ve ikili-çıkış vermesi için gerekli olan kısıtlama koşullarını gözardı etmektedirler ve dayandıkları hata işlevinin enazını bulma yöntemlerinin doğal bir sonucu olarak hesaplama süresi açısından etkin değildirler.

Bu tezde, tam kararlı HYSA 'ları için istenen kararlı hal çıkışlarının eğitici öğretilmesi amacına yönelik olarak Dinamik Algılayıcı Öğrenme Algoritması ("Recurrent Perceptron Learning Algorithm : RPLA ") ve Dinamik Geriye-Yayılım Öğrenme Algoritması (" Recurrent Backpropagation Learning Algorithm : RBLA ") adında iki farklı eğitici öğrenme algoritması geliştirilmiştir [14],[15]. Bu algoritmaları kullanan, HYSA 'lar için şablon öğrenme programı olan SLAT (" Supervised Learning Algorithm Tool ") programı yazılarak, bir kişisel bilgisayar ortamında benzetim düzeni elde edilmiştir. Geliştirilen algoritmalar, HYSA 'nın eğitici öğrenmesinin bir amaç ölçütünün tasarım kısıtlamaları altında enazını bulma sorununa dönüştürülmesine dayanmaktadır. Enazlanmak istenen amaç ölçütü, istenen kararlı-durum çıkışları ile gerçek çıkışlar arasındaki Öklid uzaklığını veren hata işlevidir. Tasarım kısıtlamaları ise HYSA 'nın tam-kararlı olmasını, yani kararlı-durumda tüm yörüngelerinin denge noktalarından birine yerleşmesini sağlayan bağlantı ağırlıklarının simetrik olma koşulu ile kararlı-durum çıkışlarının iki-kutuplu (± 1) olmasını sağlayan çıkıştan öz-geribesleme bağlantı ağırlığının, durumdan öz-geribesleme katsayısından büyük olma koşulundan oluşmaktadır.

Geliştirilen algoritmalar RBLA, hata işlevinin kısıtlamalar altında yerel enaz noktalarından birinin yeterince küçük seçilen öğrenme oranları durumunda veren, izdüşüm türünden bir eğim-düşme enazlama algoritmasıdır. RBLA, Hopfield ağı için verilmiş olan [16] Geriye-Yayılım Algoritmasının tam-kararlı HYSA için bir uygulamasıdır. Tam-bağılantılı bir ağ olan Hopfield ağında gerekli bağlantı ağırlık katsayılarının sayısı HYSA 'daki yerel ve düzgün bağlantıyı belirleyen şablon katsayılarının sayısına göre çok fazla olduğundan, önerilen RBLA, Hopfield ağı için önerilen algoritmaya göre, bu açıdan daha etkindir. HYSA 'ları için tezde önerilen RBLA 'nın, çıkış işlevinin doymaya yakın bölgelerde çok düşük türevleri olması yüzünden yavaş yakınsama sorunları vardır. Bu sorunlar tezde geliştirilen çeşitli teknikler ile yenilmeye çalışılmıştır.

RPLA ise, HYSA 'nın herbir hücresinin kararlı-durumda hücreye giren toplam girişe bağlı olarak çıkışında +1 ya da -1 veren Algılayıcı ("Perceptron") benzeri bir işlem elemanı olarak çalışması gözlemine dayanarak geliştirilmiştir. RPLA, tam-kararlı HYSA 'nın eğitici öğrenmesi için geliştirilmiş bilinen en etkin öğrenme algoritmasıdır.

Bölüm 2, YSA için genel bir bakış vermektedir. Tezde temel alınan YSA modeli HYSA Bölüm 3 'te tanıtılmıştır. Bölüm 4 'te YSA 'nın öğrenmesi ve eğitici, eğitici öğrenme açıklanmıştır. Bölüm 5 'te YSA için geliştirilen, yaygın kullanım bulmuş öğrenme algoritmaları verilmiştir. Bölüm 6 'da HYSA için tezde geliştirilen Dinamik Geriye-Yayılım Öğrenme Algoritması (RBLA) ve Dinamik Algılayıcı Öğrenme Algoritması (RPLA) tanıtılmıştır. Bölüm 7 'de geliştirilen RBLA ve RPLA algoritmalarının kişisel bilgisayar ortamında gerçekleştirilmesi amacıyla C++ dilinde yazılan SLAT programının genel işleyişi ve nasıl kullanılacağı, temel giriş çıkış blokları ve eğitim çiftlerinin elde edilme yolu tanıtılarak açıklanmıştır. Bölüm 8 'de SLAT programı ile yapılan benzetimler sunulmuştur.

BÖLÜM 2

YAPAY SİNİR AĞLARI

1943 yılında, McCulloch ve Pitts biyolojik sinir hücresinin yapısından esinlenerek geliştirdikleri bir yapay sinir hücre modeli ile VE, VEYA gibi mantık işlemlerinin yapılabileceğini gösterdiler. Böylece, biyolojik sinir sistemlerinin incelenmesi ve onlara benzer şekilde çalışan Yapay Sinir Ağ (YSA) modellerinin geliştirilmesi çok farklı bilim dalından araştırmacıların ortak olarak çalıştığı bir bilim alanı oluşturdu.

YSA üzerinde araştırmalar 60'lı yılların ortalarından, 80'li yılların başına kadar bir durgunluk dönemi yaşadı. Durgunluğa sebep gösterilebilecek en önemli olgu, YSA 'nın bilgi işlemede alternatifi olan günümüzün sayısal bilgisayarlarının yarıiletken teknolojisi ile yoğun, büyük çapta, ucuz ve güvenilir gerçekleştirme olanağı bulmasıdır. Seri olarak çalışan hızlı birimlerden oluşmuş sayısal bilgisayarlar, aritmetik işlemlerde yüksek hız, kapasite ve güvenilirlik sağlamışlardır.

Biyolojik sistemler, sayısal bilgisayar karşısında aritmetik işlemlerde kıyaslanmayacak kadar güçsüz olmalarına karşın, görüntü ve ses işleme gibi bozulmuş, tam tanımlı olmayan yüksek veri yoğunluğuna sahip bilgilerin işlenmesinde sayısal bilgisayarlara göre çok hızlı ve yüksek doğrulukta çalışırlar. Bu gözlem ışığında, biyolojik sinir sistemlerinin yapısının daha iyi anlaşılması ve ana sebep olarak ta sayısal bilgisayarların görüntü ve ses gibi işaretlerin işlenmesinde yapısal olarak güçsüz olduğunun görülmesi, 1980 'lerde çeşitli bilim dalından büyük bir araştırmacı kitlesinin YSA'larına ilgisinin odaklanmasına yolaçtı.

YSA'lar konusunda yapılan çalışmalar üç ana grupta toplanabilir. İlk grup araştırmalar, varolan veya yeni geliştirilen ağ mimarilerinin matematiksel olarak modellenmeleri, bu modellerin analizi ve bu modellerin verilen bir işlevi yerine getirmek amacıyla sentez edilmesini kapsamaktadır. İkinci grup, yapay sinir ağlarının

fiziksel gerçekleşmesi üzerinedir. En geniş çalışma alanı ise, çok değişik bilim dalından binlerce kişinin etkinlik gösterdiği YSA 'nın tüm mühendislik alanlarındaki uygulamalarını içermektedir. Uygulama alanındaki çalışmaların kapsamlı olmasının en önemli nedenleri, uygulamanın YSA konusunda uzmanlık gerektirmeden gerçekleştirilebilmesi ve YSA 'nın uygun model, bağlantı ağırlığı seçimi ve eğitim yolu ile verilen herhangi bir işleve yeteri kadar yaklaşım sağlanabilmesidir [17] [18] .

Bu bölümde, YSA 'nın C.Güzeliş [19] tarafından önerilen devre kuramsal bir tanımı ve yaygın kullanım bulmuş, önemli YSA modellerinin kullanım alanları, işlem yetenekleri ve zayıf yanları bir tablo ile verilmiştir.

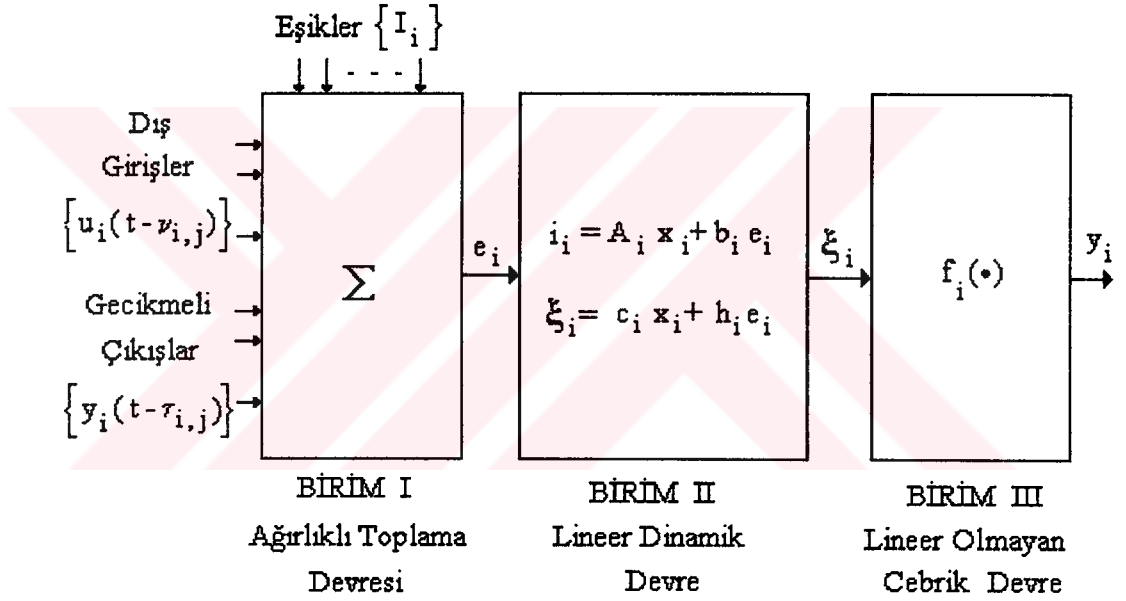
2.1. Yapay Sinir Ağlarının Bir Tanımı ve Özellikleri

Doğanın gözlenmesi sonucu, bazı olaylar ve sistemler taklit yolu ile insanların yararına sunulmaya çalışılmıştır. Günümüzdeki YSA modelleri, biyolojik sinir ağlarının çalışma biçiminin temel özelliklerini taklit etmekteyseler de, biyolojik ağlarının sadece çok ilkel modellerini oluşturmaktadırlar.

Bugüne kadar verilen çeşitli YSA modellerinin ortak özellikleri aşağıda sıralanmıştır [2].

- i) Ağ, ii)-viii) 'de verilen, hücre denilen yapı taşlarının birbirlerine ix)-xii) 'de belirtilen bir geometri ile bağlanmasından oluşan bir sistemdir.
- ii) Hücreler, genelde çok girişli ve tek çıkışlı, yüksek dereceden lineer olmayan dinamik alt devrelerdir.
- iii) Hücrelerin yapısı Şekil 2.1 'de gösterildiği biçimdedir. (Genelde lineer toplama birimi yerine lineer olmayan bir toplama birimi vardır.)
- iv) Tek olan hücre çıkış işareti, çoğullanarak dışarıya çıkış olarak alınabilir, veya diğer hücrelere giriş olacak şekilde (genelde gecikmeler ile) kullanılabilir.
- v) Bir hücre; dış girişleri (gecikme ile ulaşmış olabilir), komşu hücre çıkışlarını, ve bir eşik değerini giriş olarak kabul eder.
- vi) Hücrenin lineer dinamik birimi herhangi bir dereceden olabilir.
- vii) Hücre çıkışındaki lineer olmayan cebrik işlev 1-giriş 1-çıkışlıdır ve özel olarak lineer olabilen herhangi bir işlevdir.
- viii) Hücre parametreleri hücreden hücreye değişebilir.
- ix) Ağ, genelde çok boyutlu katmanların ileri veya geribeslemeli olacak şekilde bağlanmasından oluşmuştur.

- x) Ağdaki her bir katman, hücrelerin çok boyutlu bir dizisidir.
- xi) Katman içinde, her bir hücre komşularına (belirli bir metrik uyarınca) aynı şekilde bağlıdır. Böylece katman içi bağlantı geometrisi düzgündür.
- xii) Katmanlar arası bağlantı bir metrik ile tanımlı düzgün bir geometriye sahiptir.
- xiii) Ağda yalnızca bağlantı ağırlıkları değişebilir.
- xiv) Bağlantı ağırlıkları; ya eğitici bir öğrenme kuralı ile ya eğitici bir öğrenme kuralı ile değiştirilir ya da önceden tasarlanan değerlerde sabit tutulur.
- xv) Eğitici bir öğrenme kuralı (eğer varsa); giriş-(istenen) çıkış örnek çiftlerinin alındığı, bilinmeyen bir giriş işlevine bu örnekler yardımıyla yaklaşılmayı sağlayan bir bağlantı ağırlık değişim kuralıdır.
- xvi) Eğitici bir öğrenme kuralı (eğer varsa); giriş örneklerinin kümelendirilmesini sağlayan bir bağlantı ağırlık değişim kuralıdır.



Şekil 2.1 Hücrelerin Genel Yapısı

2.2. Yapay Sinir Ağlarının Üstünlükleri

YSA modelleri biyolojik sinir ağlarının çalışma biçimini açıklayan modeller olmaktan çok, biyolojik sinir ağlarından esinlenerek yaratılmıştır. YSA' lar, biyolojik olmayan yapı taşlarının düzgün bir mimari ile birbirine yoğun olarak bağlanmalarından oluşmaktadırlar. Sinir ağlarında bilgi işlemenin alternatif bilgi işleme yöntemlerine göre üstünlükleri maddeler halinde aşağıda verilmiştir [17],[18],[19].

Paralellik : Alışlagelmiş bilgi işlem yöntemlerinin çoğu seri işlemlerden oluşmaktadır. Bu da hız ve güvenilirlik sorunlarını beraberinde getirmektedir. Seri bir işlem gerçekleşirken herhangi bir birimin yavaş oluşu tüm sistemi doğrudan yavaşlatırken, paralel bir sistemde yavaş bir birimin etkisi çok azdır. Nitekim, seri bir bilgisayarın bir işlem elemanı beyine göre binlerce kez daha hızlı işlemesine rağmen, beynin toplam işlem hızı seri bir bilgisayara göre kıyaslanmayacak kadar yüksektir. Yapay sinir ağlarının paralel çalışması hız avantajıyla birlikte yüksek hata toleransı da sağlamaktadır. Seri bilgi işlem yapan bir sistemde herhangi bir birimin hatalı çalışması, hatta bozulmuş olması tüm sistemin hatalı çalışmasına veya bozulmasına sebep olacaktır. Halbuki paralel bilgi işleme yapan bir sistemin ayrı ayrı işlem elemanlarından meydana gelecek olan hatalı çalışma veya hasar, sistemin performansında keskin bir düşüşe yol açmadan, performansın sadece hatalı birimlerin bir oranınca düşmesine neden olur.

Yerel Bilgi İşleme: YSA'larda her bir işlem birimi, çözülecek problemin tümü ile ilgilenmek yerine, sadece problemin bir parçası işlemektedir. Hücrelerin çok basit işlemler yapmalarına rağmen, sağlanan görev paylaşımı sayesinde, çok karmaşık ve zor problemler çözülebilmektedir.

Gerçeklenme Kolaylığı: YSA'ların basit işlemler gerçekleyen türdeş hücrelerden oluşması ve bağlantıların düzgün olması, bu ağların fiziksel gerçeklenmesinde bu açıdan kolaylık olmasını sağlamaktadır.

Öğrenebilirlik: Alışlagelmiş veri işleme yöntemlerin çoğu programlama yolu ile hesaplamaya dayanmaktadır. Bu yöntemler ile tam tanımlı olmayan bir problem çözülemez. Ayrıca, herhangi bir problemin çözümü için probleme yönelik bir algoritma geliştirmek gerekir.

2.3. Önemli 15 Yapay Sinir Ağı Modeli

Tablo 2.1.a ve 2.1.b 'de birçok diğer modele taban teşkil etmiş önemli 15 YSA modeli işlem yetenekleri, sınırları belirtilerek verilmiştir [20].

Tablo 2.1.a Önemli 15 YSA Modeli

YSA 'nın Adı	Geliştiren	Yılı	Gerçekleştirilen	Sınırlar	Yorumlar
"Adaptive Resonance Theory"	Carpenter, Grossberg (Boston Univ.)	1978-1986	Örüntü tanıma (radar, sonar)	Geçişim, bozulmaya duyarlı	Karmaşık, kısıtlı bir alana uygulandı
"Avalanche"	Grossberg (Boston Univ.)	1967	Ses tanıma, robot kolu kontrolü	Hız ve devinimi değişikliğinin basit yolu yok	Bütün işleri yapan bir ağ yok
"Backpropagation"	Werbos, Parker, Rumelhart (Stanford Univ.)	1974-1985	Metinden sese dönüştürücü, robot kolu kontrolü	Sadece eğitici öğrenme uygulanabilir	En yaygın ağ, iyi çalışıyor, öğrenmesi basit
"Bidirectional Associative Memory"	Kosko (Southern California)	1985	İçerik sıralı bağlaşımsal bellek	Düşük saklama yoğunluğu, veri düzgün kodlanmalı	En kolay öğrenen ağ, parçalarından bütünü elde edebiliyor
"Boltzman and Chauchy Machine"	Hinton, Sejnovski, Szu	1985-1986	Görüntü, sonar ve radar için örüntü tanıma	Uzun eğitim zamanı, güdültü	Mutlak enaz noktayı bulabiliyor
"Brain state in a box"	Anderson (Brown Univ.)	1977	Veri tabanından bilgi çıkartılması	Tek-atış türü karar verme	Parçalardan bütünü elde edebilmektedir
"Cerebellatron"	Maar, Albus, Pellionez	1969-1982	Robot kolu ve motor hareketi kontrolü	Karmaşık kontrol girişi	"Avalanche " benzer
"Counterpropagation"	H.Nielsen (Neurocomputer Corp.)	1986	Görüntü sıkıştırma ve istatistiksel analiz	Çok sayıda hücre ve bağlantı	"Backpropagation" dan basit fakat güçsüz

Tablo 2.1.b Önemli 15 YSA Modeli

YSA 'nın Adı	Geliştiren	Yılı	Gerçekleştirilen	Sınırlar	Yorumlar
Hopfield	Hopfield (AT&T Bell Lab.)	1982	Parçalardan tüm görüntüyü veya veriyi elde etme	Öğrenmez	Geniş çapta gerçekleştirilebilir
"Madaline"	Widrow (Stanford Univ.)	1960-1962	Uyarlamalı kanal denkleyci	Giriş-çıkış arasında doğrusal ilişki varsayar	20 yıldır ticari kullanım, güçlü öğrenme kuralı
"Neocognitron"	Fukushima (HNK Lab.)	1978-1984	Elyazımı harflerin tanınması	Çok sayıda hücre ve bağlantı	Geliştirilen en karmaşık devre, dönüşüme duyarsız
"Perceptron"	Rosenblatt (Cornell Univ.)	1957	Daktilo yazımı harflerin tanınması	Karmaşık harfleri tanıyamaz, dönüşümlere duyarlı	En eski YSA, donanım olarak gerçekleştirildi
"Self-Organizing Feature Map"	Kohonen, (Helsinki Univ.)	1980	Bir geometrik bölgenin diğerine dönüştürülmesi	Uzun öğrenme zamanı	Sayısal hava devrim akış hesaplamalarında etkin
"Cellular Neural Network" Hücresel YSA	Chua , Yang (Berkeley Univ.)	1988	Görüntü işleme	İki-kutuplu çıkış	Geniş çapta tümleşik devre gerçekleştirilmeye uygun
"Generalized Cellular Neural Network"	C.Güzeliş (İ.T.Ü), Chua (Berkeley Univ.)	1991	Karmaşık, kaotik bilgi ve özellikle lineer olmayan işaret işleme	Hücre ve ağ yapısı karmaşık olmasından dolayı gerçekleştirme zorluğu	Bilinen en genel YSA modeli , evrensel YSA modeli olarak alınabilir

BÖLÜM 3

HÜCRESEL YAPAY SİNİR AĞI

Yazılım ve donanım benzetim gerçeklemeleri göstermiştir ki, her uygulama alanı için diğerlerine üstün gelen bir yapay sinir ağı modeli yoktur. Çözümü istenen problemin doğasına uygun bir model seçilmesi gerekmektedir. Diğer bir yandan, çeşitli bilim dallarından araştırmacılar tarafından önerilmiş bazı modellerin günümüzün yarı iletken teknolojisi ile, tümleşik devre olarak gerçekleştirilmeye uygun olmadıkları görülmüştür.

1988 yılında Chua ve Yang [3],[4] Hücresel Yapay Sinir Ağı (HYSA) adı verilen yeni bir model önerdiler. Büyük çaplı tümleşik devre gerçeklemesine uygun olan bu ağ modeli; görüntü işleme, örüntü tanıma, robot görmesi ve hareketli cisimlerin saptanması gibi birçok uygulamada başarılı sonuçlar vermiştir.

3.1 HYSA 'ın Yapısal Özellikleri

Aşağıda HYSA 'nın ağ-yapısı verilecek, tam-kararlı olması, iki-kutuplu olması için koşullar sunulacak ve görüntü işleme uygulamaları sunulacaktır.

Hücresel yapay sinir ağının temel yapı taşına hücre denir. Hücreler lineer ve lineer olamayan devre elemanlarından oluşmaktadır. Her bir hücre sadece yakın komşuluğunda bulunan hücrelere bağlıdır. Komşu olan hücreler birbirlerini doğrudan etkilerken, birbirine bağlı olmayan hücreler, yani komşu olmayan hücreler başlangıç durumdan kararlı duruma giderken zamanda yayılım ile birbirlerini etkilemektedirler. HYSA hücrelerinin 2-boyutlu bir ızgara oluşturacak bir şekilde birbirlerine bağlanmasından oluşmuş tek-katmanlı bir ağıdır.

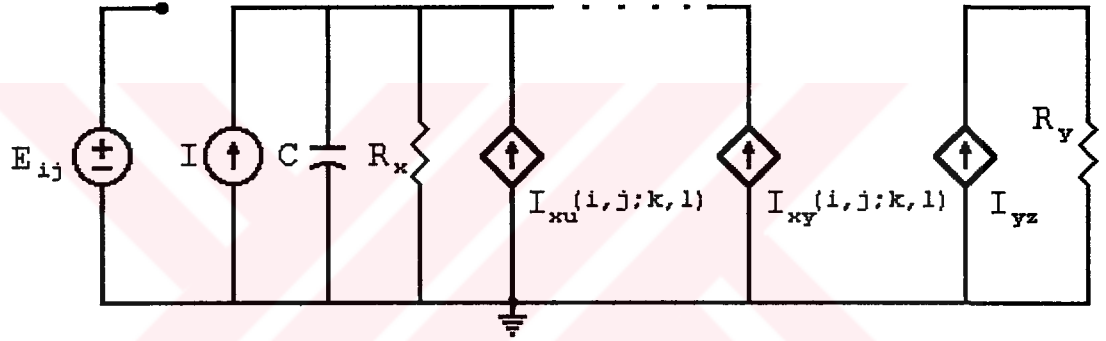
İki boyutlu M satır N sütunluk bir HYSA 'da , i 'inci satır j 'inci sütundaki bir hücre $C(i, j)$ ile gösterilir.

Tanım 1: r-komşuluğu

Bir hücresel sınır ağında bir $C(i, j)$ hücresinin r komşuluğu, r bir tamsayı olmak üzere ;

$$N_r(i, j) = \{ C(k, l) \mid \max\{|k-i|, |l-j|\} \leq r, 1 \leq k \leq M; 1 \leq l \leq N \} \quad (3.1.1)$$

şeklinde tanımlanır.

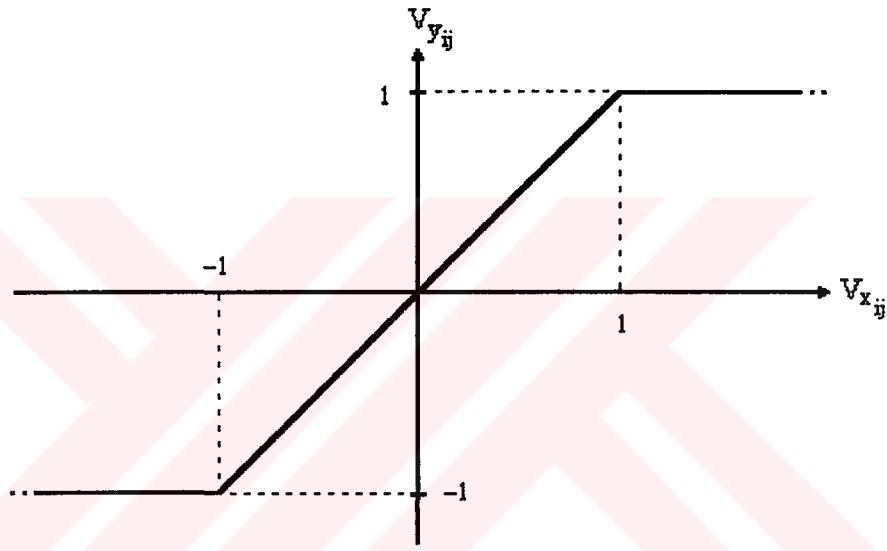


Şekil 3.1.1 Bir $C(i, j)$ Hücresinin İç Yapısı.

Genel olarak bir hücre üç temel blok ile modellenebilir [2]. Birinci blok, yani giriş bloğu, girişlerin ve diğer hücrelerden gelen çıkışların ağırlıklı toplamını oluşturan bölümdür. İkinci blok, lineer dinamik bir devreden oluşmaktadır. Hücrenin tek lineer olmayan bölümü üçüncü bloktur, yani çıkış bloğudur. Şekil 3.1.1 'de HYSA 'nın tipik bir hücresinin devre modeli verilmiştir. Burada u , x ve y sırasıyla; girişi, durumu ve çıkışı belirtmektedir.

Hücre, bağımsız bir gerilim kaynağı (E_{ij}), bağımsız bir akım kaynağı (I), bir kapasite (C), iki lineer direnç (R_x , R_y) ve diğer hücrelerin çıkışı ile kontrol edilen, m komşu sayısı olmak üzere, en fazla $2m$ tane lineer gerilim kontrollü akım kaynağından oluşmaktadır. Bu kaynakların karakteristikleri;

$C(k,l) \in N_r(i,j)$ için $I_{xy}(i,j;k,l) = A(i,j;k,l) \cdot V_{ykl}$ ve $I_{xu}(i,j;k,l) = B(i,j;k,l) \cdot V_{ukl}$
 Burada $A(i,j;k,l)$ ve $B(i,j;k,l)$ katsayıları $C(i,j)$ ve $C(k,l)$ hücreleri arasındaki bağlantı ağırlık katsayılarıdır. $A(i,j;k,l)$ katsayılarına geribesleme ağırlık katsayıları ve $B(i,j;k,l)$ 'lere de giriş ağırlık katsayıları denir. Hücrenin çıkış katında ise, parça parça doğrusal olan gerilim kontrollü bir akım kaynağı vardır. $I_{yz} = (1/R_y) f(V_{xij})$ olmak üzere $f(\bullet)$ karakteristiği Şekil 3.1.2 'de gösterilmiştir



Şekil 3.1.2 $f(\bullet)$ Karakteristiği.

Bir $C(i,j)$ hücrelerini tanımlayan devre denklemleri aşağıda verilmiştir;

Durum denklemi:

$$C \frac{dV_{xij}(t)}{dt} = -\frac{1}{R_x} V_{xij}(t) + \sum_{C(k,l) \in N_r(i,j)} A(i,j;k,l) V_{ykl}(t) + \sum_{C(k,l) \in N_r(i,j)} B(i,j;k,l) V_{ukl}(t) + I$$

$$, 1 \leq i \leq M ; 1 \leq j \leq N \quad (3.1.2)$$

Çıkış denklemi:

$$V_{yij}(t) = \frac{1}{2} \left(|V_{xij}(t)+1| - |V_{xij}(t)-1| \right), \quad 1 \leq i \leq M; 1 \leq j \leq N \quad (3.1.3)$$

Giriş denklemi:

$$V_{uij} = E_{ij}, \quad 1 \leq i \leq M; 1 \leq j \leq N \quad (3.1.4)$$

Sınır şartları:

$$|V_{xij}(0)| \leq 1, \quad |V_{uij}| \leq 1, \quad 1 \leq i \leq M; 1 \leq j \leq N \quad (3.1.5)$$

Simetri varsayımı:

$$A(i,j,k,l) = A(k,l,i,j), \quad 1 \leq i \leq M; 1 \leq j \leq N \quad (3.1.6)$$

Parametre varsayımları:

$$C > 0, \quad R_x > 0 \quad (3.1.7)$$

Özellik 1: (3.1.5) ile verilen sınır şartları ve (3.1.6) ile verilen simetri varsayımı hücrenin tam kararlı olması için yeter koşullardır. Yani, zaman sonsuza giderken bütün yörüngeler tek bir denge noktasına giderler [3].

Özellik 2: Çıkıştan öz-geribesleme katsayısı $A(i,j;i,j)$, durumdan öz-geribesleme katsayısı olan $1/CR_x$ 'den büyük seçilirse, mutlak değerce 1'den büyük iki tane kararlı denge noktası elde edilir, dolayısıyla kararlı hal durum değişkenleri mutlak değerce birden büyük olur [3].

$$\lim_{t \rightarrow \infty} |V_{xij}(t)| > 1, \quad 1 \leq i \leq M, \quad 1 \leq j \leq N \quad (3.1.8)$$

Bu durumda Şekil 3.1.2 'deki çıkış karakteristiğinden görülebileceği gibi çıkışlar mutlaka -1 veya +1 değerlerinden birini alacaktır, yani;

$$\lim_{t \rightarrow \infty} V_{yij}(t) = \mp 1, \quad 1 \leq i \leq M, \quad 1 \leq j \leq N \quad (3.1.9)$$

Sonuç olarak HYSA, dış-girişlerden ($V_{uij} \in [-1, +1]^{N \times M}$) ve başlangıç-durumlardan ($V_{xij}(0) \in [-1, +1]^{N \times M}$), kararlı-durum çıkışına cebrik bir işlev sağlamaktadır.

$$[-1, +1]^{N \times M} \times [-1, +1]^{N \times M} \xrightarrow{f} \{-1, +1\}^{N \times M} \quad (3.1.10)$$

HYSA 'ların önemli bir alt-sınıfı, konumsal-değişmez HYSA 'lardır. Konumsal-değişmez HYSA'lar bir şablon ile karakterize edilebilir. Yani, bağlantı ağırlık katsayıları

$$A(i, j; k, l) = \tilde{A}(k-i, j-l) \quad \text{ve} \quad B(i, j; k, l) = \tilde{B}(k-i, j-l) \quad (3.1.11)$$

biçiminde bir A ve B katsayı takımına bağlıdır.

Böylece bağlantı ağırlık katsayıları matrisel biçimde gösterilebilir. 1-komşuluklu bir HYSA 'nda geribesleme bağlantı ağırlıkları ve giriş bağlantı ağırlıkları, sırasıyla, (3.1.12) 'deki A ve B matrisleri gösterebilirler.

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \quad \text{ve} \quad \mathbf{B} = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} \quad (3.1.12)$$

şeklinde ifade edilebilir.

$a_1 = \tilde{A}(-1, -1)$, $a_2 = \tilde{A}(-1, 0)$, $a_3 = \tilde{A}(-1, 1)$, $a_4 = \tilde{A}(0, -1)$, $a_5 = \tilde{A}(0, 0)$ ve diğer a_i ve b_i 'ler benzer şekilde elde edilir. A matrisine geribesleme şablonu, B

matrisine giriş şablonu ve I skalerine eşik şablonu denir. HYSA 'yı tanımlayan durum denklemler $\mathbf{A}$, $\mathbf{B}$, $\mathbf{I}$ şablonları cinsinden (3.1.13) 'daki biçimde yazılabilir.

$$C \frac{dV_{xij}(t)}{dt} = - \frac{1}{R_x} V_{xij}(t) + \mathbf{A} * V_{yij}(t) + \mathbf{B} * V_{uij} + \mathbf{I} \quad (3.1.13)$$

Burada görülen $*$ sembolü, iki boyutlu konvolüsyon operatörüdür ve

$$\mathbf{T} * V_{ij} = \sum_{c(k,l) \in N_r(i,j)} T(k-i, l-j) V_{kl} \quad (3.1.14)$$

biçiminde tanımlıdır.

(3.1.6) 'da ki simetri varsayımı ve (3.1.11) 'daki konumsal-değişmezlik varsayımı altında (3.1.15) elde edilir.

$$\begin{aligned} \tilde{A}(k-i, l-j) &= \tilde{A}(i-k, j-l) = \tilde{A}(|i-k|, |j-l|) \\ \tilde{B}(k-i, l-j) &= \tilde{B}(i-k, j-l) = \tilde{B}(|i-k|, |j-l|) \end{aligned} \quad (3.1.15)$$

Dolayısıyla, şablon katsayıları arasındaki aşağıdaki eşitlikler elde edilir.

$$a_1 = a_9, a_2 = a_8, a_3 = a_7, a_4 = a_6, b_1 = b_9, b_2 = b_8, b_3 = b_7, b_4 = b_6 \quad (3.1.16)$$

3.2. HYSA 'nın Görüntü İşleme Uygulamaları

2-boyutlu ızgara yapısı ile, herbir görüntü elemanına (piksel) bir hücre karşı düşürülerek, HYSA 'nın 2-boyutlu görüntüleri işleme için doğal bir aday olacağı açıktır. Bu gözlem, HYSA uygulamalarının büyük bir kısmının görüntü işlemeye ayrılması gerçeği ile doğrulanmaktadır.

HYSA 'nın (3.1.10) uyarınca sağladığı cebrik işlev sebebiyle, giriş görüntüsü ağa ya dış-girişlerden ya başlangıç-durumlardan yada her ikisinden uygulanabilir. -1 beyaz, +1 siyah ve aradeğerler de gri seviyeler ile temsil edilirse, çıkış görüntüsü sadece siyah-beyaz bir görüntü olabilmektedir. Bu da 2 'den fazla gri seviyeli çıkış gerektiren işlevlerinin tek katmanlı bir HYSA ile yapılamayacağını göstermektedir.

Bu uygulamalar arasında; kenar saptama, köşe saptama, görüntü inceltme, boşluk doldurma, gürültü süzme, birleşik elemanların saptanması, örüntü tanıma, gölgelendirme ve gölge saptama, Japon karakterlerinin tanınması, cisimlerin sayılması, hareketli ve duran cisimlerin saptanması, baskılı devrede hatalı yolların saptanması, otomatik pilotlu araçların kontrolü sayılabilir [4][5][21][22][23][24].



BÖLÜM 4

ÖĞRENME

Bu bölümde YSA'nın öğrenmesinden ne anladığımız açıklanacaktır.

Simon (1983) tarafından öğrenme, bir sistemin bir görevi daha etkili ve verimli yapabilme yeteneğinin kazanması şeklinde tanımlanmıştır. Michalski (1986) ise, öğrenmenin deneyimin bir göstergesi olduğunu belirtmiştir [26].

Genel olarak, öğrenme bir bilginin çözümü demektir. Bir sistem aldığı bilgiyi kendi bünyesinde çözerek öğrenmektedir, bunu da öğrenme süreci boyunca yapısını değiştirerek yerine getirmektedir.

Matematiksel tabanda olaya bakıldığında, bir sistem uygulanan x_i işaretine karşılık istenen bir y_i işaretini çıkışta üretiyorsa, bu sistem (x_i, y_i) uyarıcı-yanıt çiftini öğrenmiştir denir. (x_i, y_i) giriş-çıkış çifti $f(\bullet) : R^n \rightarrow R^p$ giriş-çıkış işlevinin bir örneğini temsil etsin. Bu durumda, tüm $x \in R^n$ için, sistem $y = f(x)$ yanıtı üretiyorsa, sistem $f(\bullet)$ işlevini öğrenmiştir [27].

Sonuçta öğrenme; sistemin, parametrelerini değiştirerek verilen giriş işaretlerine karşılık istenen çıkış işaretlerini üretmesini sağlayacak bir kendini uyarılama sürecidir.

4.1. YSA 'larda Öğrenme

Yapay Sinir Ağlar'ında, değişebilen sistem parametreleri, hücreler arası bağlantıları sağlayan sinapsları temsil eden bağlantıların ağırlık katsayılarıdır. Öğrenme sırasında oluşabilecek değişimler aşağıda verilmiştir [20] :

- Yeni bağlantıların oluşması
- Varolan bağlantıların kaybolması
- Varolan bağlantıların ağırlıklarının değişimi

Öğrenme olayı genelde bir süreçtir, yani adım adım yapılan bir işlemdir. Öğrenme sırasında bir eğitim kümesine gereksinim vardır. Eğitim kümesi, YSA 'ya öğrenme için uygulanacak olan giriş verilerinden ve varsa istenen çıkışlardan oluşmaktadır. Her adımda eğitim kümesinden sırayla veya rastgele örnekler alınmakta ve ağa uygulanmaktadır. Adım sonunda belirli bir amaç ölçütü uyarınca ağırlık bağlantı ağırlık katsayıları değiştirilmektedir.

Adım adım eğitim dışında, kümesel eğitim denilen öğrenme kuralları da vardır. Bunlarda eğitim tek adımda gerçekleşmektedir. Bağlantı ağırlık katsayılarının son değerleri eğitim kümesindeki tüm örnekler göz önüne alınarak tek bir adımda bulunmaktadır.

4.2. Eğitici ve Eğitici-siz Öğrenme

YSA 'larda öğrenme ilk aşamada eğitici ve eğitici-siz öğrenme diye ikiye ayrılabilir.

Eğitici öğrenme basit anlamda öğrenciler ile öğretmenin bir sınıftaki dersine benzetilebilir. Öğrenciler ile öğretmen ders boyunca karşılıklı etkileşim içindedirler. Öğrencilerin sorulan sorulara verdikleri cevaplar öğretmen tarafından denetlenmektedir ve gerekirse, doğru cevap öğretmen tarafından öğrencilere iletilmektedir.

YSA 'larda eğitici öğrenmede, ağ çevreden aldığı girişlere karşı bir eğiticinin belirttiği doğru çıkışı üretmek hedefiyle kendisini uyarlamaktadır. Eğiticiden gelen girişler iki türlü olabilir. İlk türde eğitici sadece cevabın doğru veya yanlış olduğunu belirtmektedir. Buna ödül-ceza türü eğitici öğrenme denmektedir. İkinci türde ise, eğitici tarafından mevcut girişe karşılık istenen çıkış ağı uygulanmaktadır. Ağ tarafından üretilen gerçek çıkış ile istenen çıkış arasındaki fark, bağlantı ağırlık katsayılarının nasıl ve ne kadar değişeceğini belirlemektedir. Söz konusu fark belirli bir metriğe göre tanımlanmaktadır ve yapılan çıkış hatasının bir göstergesidir. Bağlantı ağırlık katsayılarıdaki değişim ise, geribesleme olarak gelen hatayı enazlayacak ve hatta mümkünse sıfırlayacak şekilde olmaktadır.

Eğitici öğrenme veya kendi kendine öğrenmede, herhangi bir eğiticiden ağın ürettiği yanıtın doğruluğuna dair herhangi bir geribesleme gelmemektedir. YSA'nın bağlantı ağırlık katsayıları, oluşan hatayı göz önüne almaksızın daha önce belirlenen bir amaç ölçütü uyarınca sadece girişlere bağlı olarak değişmektedirler. Bu tür bir öğrenme her YSA için yoktur ve her probleme uygulanamaz.

BÖLÜM 5

YSA 'LARDA ÖĞRENME ALGORİTMALARI

Bu bölümde, yapay sinir ağlarında öğrenmeyi daha iyi açıklamak için, Bölüm 6 ve Bölüm 7 'de geliştirilen öğrenme algoritmaları ile ilişki kurmak amacıyla, hem eğitici, hem de eğitici olmayan öğrenmeye yönelik yedi önemli öğrenme algoritması verilecektir [18].

Aşağıdaki algoritmalarda, $w_i = [w_{i1} w_{i2} \dots w_{in}]^T$ vektörü ağdaki i 'inci hücreye gelen n tane girişin bağlantı ağırlık katsayılarını, $x = [x_1 x_2 \dots x_n]^T$ vektörü ağa gelen n tane girişi, y_i ve d_i de sırasıyla i 'inci hücrenin çıkışını ve istenen çıkışını temsil etmektedirler.

5.1. Hebb Öğrenme Algoritması

Hebb'in öğrenme algoritmasında temel fikir, hücrenin darbe üretmesine sebep olan bağlantıların ağırlık katsayılarını büyütürken etkilerini arttırmaktır. Buna göre değişim (5.1.a) ve (5.1.b) ile tanımlanmıştır.

$$w_i(k+1) = w_i(k) + c f(w_i^T(k) x(k)) x(k) \quad (5.1.a)$$

yani,

$$w_i(k+1) = w_i(k) + c y_i(k) x(k) \quad (5.1.b)$$

Burada c 'ye öğrenme katsayısı veya öğrenme oranı denilmektedir. Dışardan istenilen bir çıkış uygulanmadığından eğitici olmayan bir öğrenme algoritmasıdır.

5.2. Algılayıcı ("Perceptron") Öğrenme Algoritması

Rosenblatt (1958) [28] tarafından önerilen Algılayıcı Öğrenme Algoritması 'nda w değişimi çıkış ile istenen çıkışın farkı ile orantılıdır, dolayısıyla eğitici bir öğrenme algoritmasıdır.

$$w_i(k+1) = w_i(k) + c [d_i(k) - \text{sgn}(w_i^T(k) x(k))] x(k) \quad (5.2)$$

Devrenini çıkışı iki-kutuplu ve w 'ların oluşturduğu bir hiperyüzey tarafından belirlenmektedir. Ağın ürettiği çıkış istenen çıkıştan farklı ise, hiperyüzey hatayı azaltacak yönde hareket etmektedir. Hata sıfır olduğunda w değişimi durmaktadır.

5.3. Eğimdüşme Öğrenme Algoritması

Eğimdüşme ("Gradient Descent") Öğrenme Algoritması veya "Delta kuralı" öğrenme algoritması sadece türetilabilir fonksiyonlara sahip olan ağlara uygulanabilmektedir. Eğitici bir öğrenme algoritmasıdır. w değişimi hatanın eğiminin ters yönündedir. Böylece hata fonksiyonunun oluşturduğu çanağın dibine, yani hatanın minimum olduğu yere doğru hareket edilmektedir.

$$w_i(k+1) = w_i(k) - c \frac{\partial E(k)}{\partial w(k)} \quad (5.2)$$

$E(k)$ hata fonksiyonunu temsil etmektedir ve (5.4.a) ve (5.4.b) 'de tanımlanmıştır.

$$E(k) = \frac{1}{2} (d_i(k) - y_i(k))^2 \quad (5.4.a)$$

$$E(k) = \frac{1}{2} (d_i(k) - f(w_i^T(k) x(k)))^2 \quad (5.4.b)$$

$$\frac{\partial E(k)}{\partial w(k)} = -(d_i(k) - f(w_i^T(k) x(k))) \frac{\partial f(w_i^T(k) x(k))}{\partial (w_i^T(k) x(k))} x(k) \quad (5.5)$$

5.4. Widrow-Hoff Öğrenme Algoritması

Widrow-Hoff Öğrenme Algoritması eğitici bir öğrenme algoritmasıdır. w değişimi ağı çıkışından bağımsızdır.

$$w_i(k+1) = w_i(k) + c (d_i(k) - w_i(k) x(k)) x(k) \quad (5.6)$$

Aslında bu algoritma Eğim Düşüm Algoritması'nın özel bir halidir.

$$f(w_i^T(k) x(k)) = w_i^T(k) x(k) \quad (5.7)$$

alınırsa (5.6) denklemi, yani Widrow-Hoff algoritması elde edilmektedir.

5.5. İnti Öğrenme Algoritması

Hebb Öğrenme Algoritması'nın eğitici uyarlamasıdır. $f(w_i^T(k) x(k))$ yerine $d_i(k)$ istenen çıkış alındığında, (5.8) denkelemi elde edilmektedir.

$$w_i(k+1) = w_i(k) + c d_i(k) x(k) \quad (5.8)$$

5.6. Kazanan-Herşeyi-Alır Öğrenme Algoritması

Yarışma türü öğrenme algoritmalarına örnek teşkil etmekte olan bu algorithmada esas, giriş işaretine en yakın olan bağlantı ağırlık katsayılarını bulmaktır. Bu w 'lara ilişkin hücre kazanan hücre olarak adlandırılmaktadır. Sadece kazanan hücre bir çıkış üretmekte ve bu hücreye ilişkin w 'lar değişime uğramaktadır. Eğiticişiz öğrenmenin tipik bir örneği olan bu algoritma sonuçta ağa gelen girişleri sınıflandırmaktadır.

$$w(k+1) = w(k) + c(x(k) - w_m(k)) \quad (5.9)$$

Burada w_m kazanan hücrenin bağlantı ağırlık katsayısı ve (5.10) 'da tanımlanmıştır.

$$w_m^T x = \max_{i=1,2,\dots,p} (w_i^T x) \quad (5.10)$$

5.7. "Outstar" Öğrenme Algoritması

Eğiticişiz öğrenme algoritma türü olan bu öğrenme algoritmasında amaç, w 'ları istenen çıkışa benzetmektir.

$$w(k+1) = w(k) + c(d_i(k) - w(k)) \quad (5.11)$$

BÖLÜM 6

HYSA 'LAR İÇİN DİNAMİK ÖĞRENME ALGORİTMALARI

Dinamik bir ağ olan Hücrel Yapay Sinir Ağı için değişik öğrenme algoritmaları geliştirilmiştir [11], [12]. [11] 'de verilen algoritma sürekli zaman HYSA'larının istenen yörüngelerini öğrenmeye yöneliktir. [12] 'de verilen algoritma ise, ayrık-zaman HYSA 'larının kararlı-durum çıkışlarını öğrenmek amacı ile geliştirilmiştir. İki algoritma da tam kararlılık ve iki kutupluluk için gerekli olan kısıtlamaları göz ardı etmektedir. Bu kısıtlamalar da göz önüne alındığında HYSA'larının kararlı-durum çıkışlarını öğrenmeye yönelik bir öğrenme algoritması aslında kısıtlamalı bir enazlama probleminden farksız olduğu ortaya çıkmaktadır.

HYSA 'lar için geliştirilen öğrenme algoritmalarında kullanılan notasyon aşağıda verilmiştir. $v = [v_u^T v_x^T]^T$ giriş vektörünü ifade etmektedir. Burada $v_u = [..., u_{i,j}, ...]^T$ dış girişleri, $v_x = [..., x_{i,j}(0), ...]^T$ başlangıç durumlarını temsil etmektedir. Belirli bir v giriş vektörü ve bağlantı ağırlık katsayıları vektörü w için, tam kararlı bir HYSA 'da çıkış vektörü $y(t) = [..., y_{i,j}(t), ...]^T$ kararlı-durum çıkışı olan $y(\infty)$ 'a dönüşmektedir.

HYSA 'larda kararlı-durum çıkışlarının eğitici öğretilmesi için, ağı kararlı-durum çıkışları ve istenen kararlı-durum çıkışlarının (d) farkı ile orantılı bir hata fonksiyonu minimize edilmesi gerekmektedir. Bu amaç doğrultusunda ağ, kararlı-durum çıkışlar ve bunlara karşı düşen istenen çıkışlardan oluşmakta olan çiftlerle eğitilmesi gerekmektedir.

Minimize edilecek hata fonksiyonu $E(w)$, kararlı-durum çıkışlarının ve istenen çıkışlarının farkı ile orantılıdır. Metrik olarak Öklid mesafesi alındığında hata fonksiyonu bir eğitim çifti için (6.1) 'de gibi tanımlanmaktadır.

$$E^k(w) = \frac{1}{2} \sum_{i,j} (y_{i,j}^k(\infty) - d_{i,j}^k)^2 \quad (6.1)$$

Tam kararlı bir HYSA 'nın elde edilebilmesi için bağlantı ağırlık katsayıları simetrik alınması yeterlidir . Böylece w vektörü (6.2) 'deki gibi olur.

$$w = [a^T b^T I]^T = [a_1 a_2 a_3 a_4 a_5 b_1 b_2 b_3 b_4 b_5 I]^T \quad (6.2)$$

Bölüm 6.1 'de geliştirilen algoritmalarından Dinamik Geriye-Yayılım Öğrenme Algoritması (" Recurrent Backpropagation Learning Algorithm : RBLA "), Hopfield ağı için verilmiş olan [16] Geriye-Yayılım Algoritması 'nın tam-kararlı HYSA için bir uygulamasıdır. Tam-bağlantılı bir ağ olan Hopfield ağında gerekli bağlantı ağırlık katsayılarının sayısı HYSA 'daki yerel ve düzgün bağlantıyı belirleyen şablon katsayılarının sayısına göre çok fazla olduğundan, önerilen RBLA, Hopfield ağı için önerilen algoritmaya göre, bu açıdan daha etkindir.

Bölüm 6.2 'de, HYSA 'nın herbir hücresinin kararlı-durumda hücreye giren toplam girişe bağlı olarak çıkışında +1 ya da -1 veren Algılayıcı ("Perceptron") benzeri bir işlem elemanı olarak çalışması gözlemine dayanarak Dinamik Algılayıcı Öğrenme Algoritması (" Recurrent Perceptron Learning Algorithm ") geliştirilmiştir.

6.1. Dinamik Geriye-Yayılım Öğrenme Algoritması

HYSA 'nın parça parça doğrusal çıkış işlevinin yerine buna çok yakın olan sürekli türetilbilir bir çıkış işlevi alınarak, türev gerektiren bir öğrenme algoritması kullanılmasına olanak sağlanmıştır. Geliştirilen algoritmalarından Dinamik Geriye-Yayılım Öğrenme Algoritması (" Recurrent Backpropagation Learning Algorithm : RBLA "), hata işlevinin kısıtlamalar altında yerel enaz noktalarından birinin yeterince küçük seçilen öğrenme oranları durumunda veren, izdüşüm türünden bir eğim-düşme enazlama algoritmasıdır.

İki-kutuplu çıkışların elde edilebilmesi için, (3.1.2) ile verilen durum denklemin normalize edilmiş hali, yani durumdan öz-geribesleme katsayısı $\frac{1}{CR_x} = 1$ için, $a_s \geq 1$ olması gerekmektedir. Bu sebepten bağlantı ağırlıklarının (6.1.1) ile verilen A kümesine izdüşümü alınmıştır.

$$w(n) \in A = \{w(n) \in R^{11} \mid a_s \geq 1\} \quad (6.1.1)$$

$w(n) \notin A$ olduğu durumlarda $w(n) = K_n w(n)$ alınmaktadır.

$$K_n = \mu \frac{1}{a_s(n)} \quad (6.1.2)$$

Burada $\mu > 0$ olmak üzere sabit bir sayıdır.

Yeni oluşan hata fonksiyonunun minimizasyonu için eğim düşüm enazlama yöntemi kullanılmıştır. Bu durumda bağlantı ağırlık katsayıları iteratif olarak (6.1.3) 'deki gibi değişmektedir.

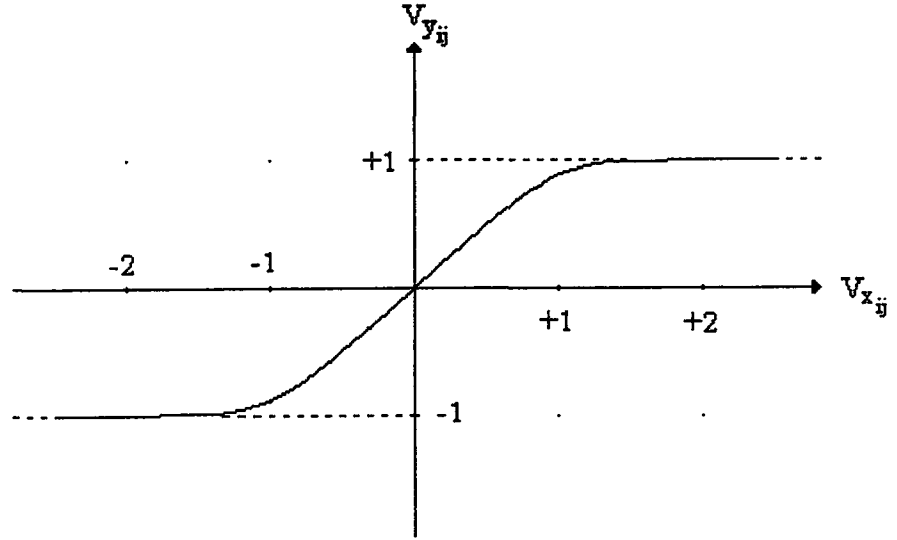
$$w(k+1) = \left[w(k) - \varepsilon(k) \left(\nabla \hat{E}[w(k)] \right)^T \right] \quad (6.1.3)$$

$$\nabla_w \hat{E}[w(k)] = \sum_{i,j} \left(y_{i,j}^k(\infty) - d_{i,j}^k \right) \nabla_w y_{i,j}^k(\infty) \quad (6.1.4)$$

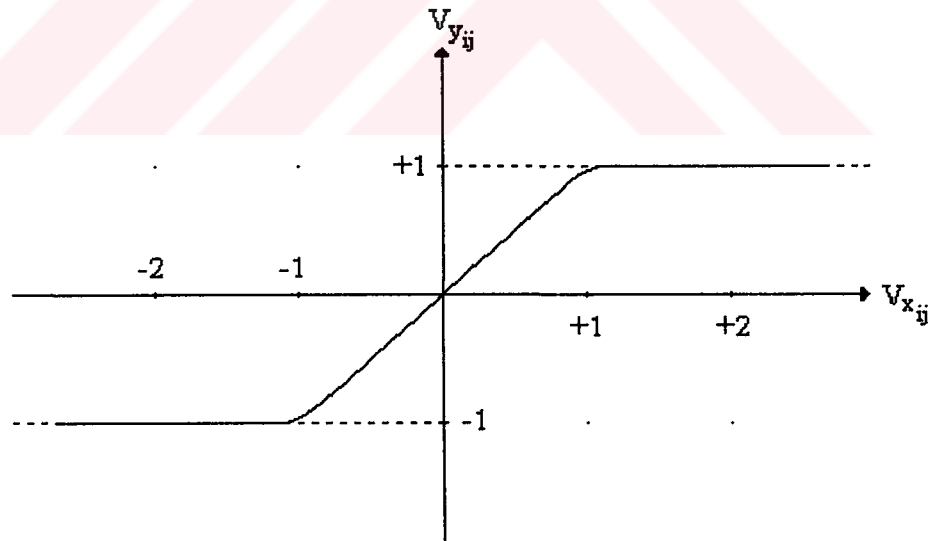
(6.1.4) 'da görüldüğü gibi ağırlık kararlı-durum çıkışları türetilebilir olmaları gerekmektedir. Şekil 3.1.2 'de verilmiş olan kısmi lineer çıkış fonksiyonu bu şartı sağlamadığı için, (3.1.3) yerine (6.1.5) kullanılmıştır.

$$y_{i,j} = F_\beta(x_{i,j}) = \frac{1}{2\beta} \left\{ \ln \left[\cosh \beta (x_{i,j} + 1) \right] - \ln \left[\cosh \beta (x_{i,j} - 1) \right] \right\} \quad (6.1.5)$$

Büyük β değerleri için (6.1.5) çıkış fonksiyonu (3.1.3) 'ya yakın olmaktadır.



Şekil 6.1.a $\beta = 3$ için (6.1.5) çıkış fonksiyonu



Şekil 6.1.b $\beta = 10$ için (6.1.5) çıkış fonksiyonu

Zincir kuralı kullanıldığında, kararlı-durum çıkışlarının bağlantı ağırlık katsayılarına göre türevler hesaplanabilmektedir.

$$\nabla_w y_{i,j}^k(\infty) = \frac{\partial y_{i,j}^k(\infty)}{\partial w^k} = \frac{\partial F_\beta(x_{i,j}^k(\infty))}{\partial x_{i,j}^k(\infty)} \frac{\partial x_{i,j}^k(\infty)}{\partial w^k} \quad (6.1.6)$$

$$\frac{\partial F_\beta(x_{i,j})}{\partial x_{i,j}} = \frac{1}{2} (\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)) \quad (6.1.7)$$

HYSA kararlı duruma ulaştığında $\dot{x}_{i,j}(\infty) = 0$ olacağından (3.2) ile verilen durum denkleminde $x_{i,j}(\infty)$ hesaplanabilmektedir. $x_{i,j}(\infty)$ 'un w vektörüne göre türevi aşağıda verilmiştir.

$$\begin{aligned} \frac{\partial x_{i,j}^k(\infty)}{\partial a_1} &= [y_{i-1,j-1}^k(\infty) + y_{i+1,j+1}^k(\infty)] & \frac{\partial x_{i,j}^k(\infty)}{\partial a_2} &= [y_{i-1,j}^k(\infty) + y_{i+1,j}^k(\infty)] \\ \frac{\partial x_{i,j}^k(\infty)}{\partial a_3} &= [y_{i-1,j+1}^k(\infty) + y_{i+1,j-1}^k(\infty)] & \frac{\partial x_{i,j}^k(\infty)}{\partial a_4} &= [y_{i,j-1}^k(\infty) + y_{i,j+1}^k(\infty)] \\ \frac{\partial x_{i,j}^k(\infty)}{\partial a_5} &= y_{i,j}^k(\infty) \\ \frac{\partial x_{i,j}^k(\infty)}{\partial b_1} &= [u_{i-1,j-1}^k + u_{i+1,j+1}^k] & \frac{\partial x_{i,j}^k(\infty)}{\partial b_2} &= [u_{i-1,j}^k + u_{i+1,j}^k] \\ \frac{\partial x_{i,j}^k(\infty)}{\partial b_3} &= [u_{i-1,j+1}^k + u_{i+1,j-1}^k] & \frac{\partial x_{i,j}^k(\infty)}{\partial b_4} &= [u_{i,j-1}^k + u_{i,j+1}^k] \\ \frac{\partial x_{i,j}^k(\infty)}{\partial b_5} &= u_{i,j}^k & \frac{\partial x_{i,j}^k(\infty)}{\partial I} &= 1 \end{aligned} \quad (6.1.8)$$

Bağlantı ağırlık katsayıların değişimi (6.1.9) 'de verilmiştir.

$$a_1(k+1) = a_1(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [y_{i-1,j-1}^k(\infty) + y_{i+1,j+1}^k(\infty)]$$

$$a_2(k+1) = a_2(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [y_{i-1,j}^k(\infty) + y_{i+1,j}^k(\infty)]$$

$$a_3(k+1) = a_3(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [y_{i-1,j-1}^k(\infty) + y_{i+1,j+1}^k(\infty)]$$

$$a_4(k+1) = a_4(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [y_{i,j-1}^k(\infty) + y_{i,j+1}^k(\infty)]$$

$$a_5(k+1) = a_5(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [y_{i,j}^k(\infty)]$$

$$b_1(k+1) = b_1(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [u_{i-1,j-1}^k + u_{i+1,j+1}^k]$$

$$b_2(k+1) = b_2(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [u_{i-1,j}^k + u_{i+1,j}^k]$$

$$b_3(k+1) = b_3(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [u_{i-1,j+1}^k + u_{i+1,j-1}^k]$$

$$b_4(k+1) = b_4(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [u_{i,j-1}^k + u_{i,j+1}^k]$$

$$b_5(k+1) = b_5(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] [u_{i,j}^k]$$

$$I(k+1) = I(k) - \varepsilon(k) \frac{1}{2} [\tanh \beta(x_{i,j}(\infty) + 1) - \tanh \beta(x_{i,j}(\infty) - 1)] \quad (6.1.9)$$

Öğrenme algoritmasının deterministik uyarlaması için eğitim setinde bulunan tüm eğitim çiftleri ağı uygulandıktan sonra toplam hataya göre w 'lar değiştirilmektedir. (6.1) 'deki hata fonksiyonu N tane eğitim örneği için (6.12) 'de verilmiştir.

$$E(w) = \frac{1}{2} \sum_N \sum_{i,j} (y_{i,j}^k(\infty) - d_{i,j}^k)^2 \quad (6.1.10)$$

$\varepsilon(k)$ öğrenme oranı da, sabit pozitif bir sayı olmaktadır.

Öğrenme algoritmasının stokastik (rastlantısal) uyarlamasında ise, eğitim setindeki örnekler rastgele seçilmektedir ve her örnekten sonra w değiştirilmektedir.

$\hat{E}(w)$ ile verilen hata fonksiyonunun yakınsaması için $\varepsilon(k)$ öğrenme oranı (6.1.11) ve (6.1.12) ile verilen iki koşulu sağlaması gerekmektedir.

$$\sum_{i=1}^{\infty} \varepsilon(k) = \infty \quad (6.1.11)$$

$$\sum_{i=1}^{\infty} \varepsilon^2(k) < \infty \quad (6.1.12)$$

6.2. Dinamik Algılayıcı Öğrenme Algoritması

Bölüm 6.2 'de, HYSA 'lar için geliştirilmiş olan öğrenme algoritmasında hücrelerin çıkış fonksiyonu parça parça doğrusal bir fonksiyon yerine sürekli bir fonksiyon kullanılmıştır. Böylece eğitim düşün algoritmasından yararlanılabilmektedir.

Bu bölümde tanıtılacak olan öğrenme algoritmasında hata fonksiyonunun türevine ihtiyaç yoktur. Geliştirilmiş olan bu algoritma, bilinen Algılayıcı Öğrenme Algoritması'nın (" Perceptron Learning Algorithm") [29] dinamik bir ağ olan HYSA'ya uyarlaması olduğundan Dinamik Algılayıcı Öğrenme Algoritması (Recurrent Perceptron Learnig Algorithm : RPLA ") diye adlandırılmıştır.

İlk geliştirilen öğrenme algoritmasında da olduğu gibi, Dinamik Algılayıcı Öğrenme Algoritması ($a_s > 1$) koşulunu sağlayan bir bağlantı ağırlık katsayı vektörü (şablon) ile başlayıp, giriş-(istenen) kararlı-durum çıkış çiftlerinden oluşan eğitim seti ile eğitmektedir. Burada girişler, hem dış ortamdan gelen girişler hem de başlangıç durumlarıdır.

RPLA ana fikir olarak Hebb öğrenme algoritmasına benzer: Kararlı-durum çıkışı, istenen kararlı-durum çıkışından farklı olan hücrenin, komşu hücrelerinden gelen bağlantılarının ağırlık katsayılarını arttırmaktadır (azaltmaktadır).

İki-kutupluluk ve kararlılık şartlarının sağlanabilmesi için, her adım sonunda değiştirilen bağlantı ağırlık katsayıları (w), $A = \{w \in R^{11} \mid a_s > 1\}$ kümesi ile kısıtlandırılmışlardır. Görüldüğü gibi w simetrik alınarak, bağlantı ağırlık katsayıları 19'dan 11'e indirilmiştir. Böylece hem kararlılık şartı sağlanmıştır, hem de w 'nin güncelleştirilmesi için gereken hesaplar azalmıştır.

RPLA ile enazlanacak olan hata işlevi bir önceki bölümde verilen notasyonlar uyarınca aşağıda verilmiştir.

$$E(w) = \sum_{i,j} y_{i,j}(\infty) (y_{i,j}(\infty) - d_{i,j}) = \sum_{(i,j) \in D^+} y_{i,j}(\infty) - \sum_{(i,j) \in D^-} y_{i,j}(\infty) \quad (6.2.1)$$

Burada, $D^+ = \{(i,j) \mid y_{i,j}(\infty) = -d_{i,j} = 1\}$ yani, istenen çıkış -1 olmasına rağmen +1 kararlı-durum çıkışı veren hücrelerin oluşturduğu küme. $D^- = \{(i,j) \mid y_{i,j}(\infty) = -d_{i,j} = -1\}$ yani, istenen çıkış +1 olmasına rağmen -1

kararlı-durum çıkışı veren hücrelerin oluşturduğu küme. Buna göre $E(w)$ hata fonksiyonu, kararlı-durum çıkışlar ile istenen kararlı-durum çıkışları çakışmayan hücrelerin toplamıdır.

Hata fonksiyonunun değerinin bulunabilmesi için, istenen çıkışlarla, kararlı-durum çıkışları çakışmayan hücrelerin hangileri olduğu bilinmesi gerekmektedir. Kararlı durumda $\dot{x}_{i,j}(\infty) = 0$ olacağı için (3.2) bağıntısından $x_{i,j}(\infty)$ sol tarafa alınarak bulunabilmektedir ve normalize halde (6.2.2) olmaktadır.

$$x_{i,j}(\infty) = \sum_{p,l \in \{-1,0,1\}} w_{p,l} y_{i+p,j+l}(\infty) + \sum_{p,l \in \{-1,0,1\}} z_{p,l} u_{i+p,j+l} + I = [Y_{i,j}]^T w \quad (6.2.2)$$

Burada $Y_{i,j}$ toplam girişi temsil etmektedir.

$$Y_{i,j} = \left[[y_{i,j}]^T [u_{i,j}]^T 1 \right] \quad (6.2.3)$$

$$y_{i,j} = \begin{bmatrix} y_{i-1,j-1}(\infty) + y_{i+1,j+1}(\infty) & y_{i-1,j}(\infty) + y_{i+1,j}(\infty) \\ y_{i-1,j+1}(\infty) + y_{i+1,j-1}(\infty) & y_{i,j+1}(\infty) + y_{i,j-1}(\infty) & y_{i,j}(\infty) \end{bmatrix}^T \quad (6.2.4)$$

$$u_{i,j} = \begin{bmatrix} u_{i-1,j-1} + u_{i+1,j+1} & u_{i-1,j} + u_{i+1,j} & u_{i-1,j+1} + u_{i+1,j-1} & u_{i,j+1} + u_{i,j-1} & u_{i,j} \end{bmatrix}^T \quad (6.2.5)$$

Çıkış fonksiyonu (6.2.6) olduğuna göre, $E(w)$ kolayca hesaplanabilmektedir.

$$y_{i,j}(\infty) = \frac{1}{2} \left[|x_{i,j}(\infty) + 1| - |x_{i,j}(\infty) - 1| \right] \quad (6.2.6)$$

Bağlantı ağırlık katsayıların değişimi (6.2.7) ile verilmiştir.

$$w(n+1) = [w(n) - \varepsilon(n)Y(n)] \quad (6.2.7)$$

Burada;

$$Y(n) = \left(\sum_{(i,j) \in D^+} Y_{i,j}(n) - \sum_{(i,j) \in D^-} Y_{i,j}(n) \right) \quad (6.2.8)$$

ve

$$w(n) \in A = \{w(n) \in \mathbb{R}^{11} \mid a_5 > 1\} \quad (6.2.9)$$

$w(n) \notin A$ olduğu durumlarda $w(n) = K_n w(n)$ alınmaktadır.

$$K_n = \mu \frac{1}{a_5(n)} \quad (6.2.10)$$

$\mu > 1$ olmalıdır ve genelde $\mu = 1.5$ alınmıştır.

Aşağıda verilmiş olan özellikler RPLA 'nın davranışını daha iyi açıklamaktadır.

Özellik 1: $Y(w)$ 'nin beşinci elemanı, yani a_5 , $E(w)$ 'ya eşittir. Ayrıca pozitif öğrenme oranları için $(\varepsilon(n) > 0)$ ve $a_5 - \varepsilon(n)E(n) \leq 1$ olamadığı sürece a_5 artmamaktadır.

Özellik 2: $Y(w)$ 'nin onbirinci elemanı, yani I , $\#(D^+) - \#(D^-)$ farkına eşittir. Burada $\#(D^+)$ ve $\#(D^-)$ sırasıyla +1 ve -1 uyumsuz olan hücrelerden oluşan kümelerin eleman sayısını ifade etmektedir.

BÖLÜM 7

SLAT PROGRAMI KULLANIM KILAVUZU

7.1. Giriş

SLAT (" Supervised Learning Algorithm Tool "), Hücresel Yapay Sinir Ağları için tasarlanmış eğitici öğrenme algoritmalarını gerçekleyen programdır. Program, belirli giriş görüntülerine ve başlangıç görüntülerine karşılık, istenen kararlı-durum çıkış görüntülerini veren şablonları üretmektedir. Böylece, HYSA 'lar için değişik görüntü işlemleri yapabilen şablonlardan oluşan bir kütüphane üretilmektedir.

Program esas olarak Dinamik Algılayıcı Öğrenme Algoritması için tasarlanmış olmasına rağmen, Dinamik Geriye yayılım Öğrenme Algoritması 'nı da uygulayabilmektedir. İki öğrenme algoritmasının da deterministik ve rastlantısal (stokastik) uyarlamaları programda bulunmaktadır.

Programda, HYSA benzetimi için üç değişik analiz yöntemi kullanılmaktadır. Bunlar " Euler ", " Heun " ve " Dördüncü Dereceden Runge-Kutta " yöntemleridir.

7.2. Gerekli Bilgisayar Donanımı

SLAT programı Turbo C++ programlama dilinde yazılmıştır ve bellekte 161 Kbyte yer kaplamaktadır. Program IBM uyumlu bilgisayarlarda çalışabilmektedir. Ayrıca kullanılacak bilgisayar en azından 640 Kbyte RAM belleğe ve VGA grafik kartına sahip olmalıdır. Programın yeterince hızlı çalışabilmesi için, matematiksel işlemcisi olan bir 80386 veya daha gelişmiş bir bilgisayar tavsiye edilmektedir.

7.3. Programın İşleyişi

Program çalıştırıldıktan sonra eğitim sürecinin başlatılabilmesi için, yükleme (LOAD) menüsünden eğitim kümesi ve başlangıç şablonu seçilmesi gerekmektedir. Öğrenme algoritmasının seçimi, ayarlamalar (SETUP) menüsünün altında bulunan algoritma (ALGORITHM) alt menüsünden yapılmaktadır. Yine ayarlamalar menüsünün metod (METHOD) alt menüsünden, deterministik veya rastlantısal algoritma seçimi yapılabilmektedir. Tüm ayarlamalar tamamlandıktan sonra eğitim süreci başlatılması için ana menüde bulunan " RUN " tuşu aktif hale getirilir.

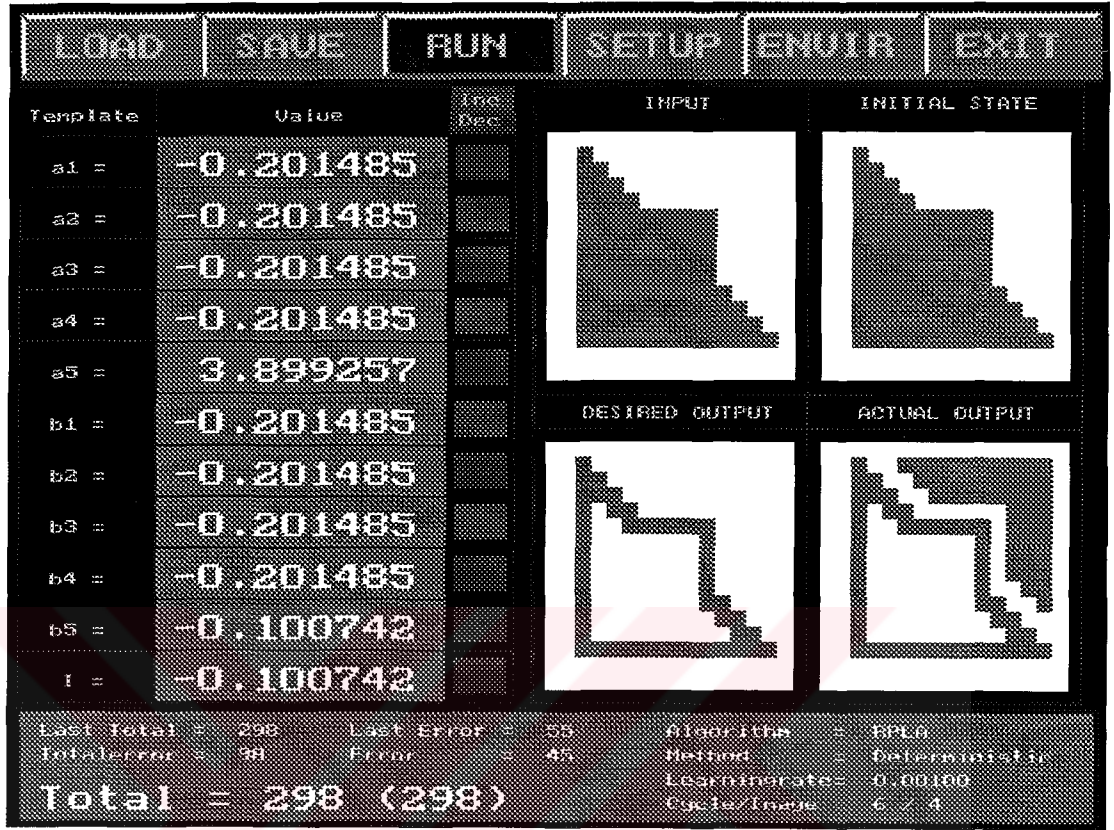
Öğrenme algoritmalarının deterministik ayarlamaları, toplam hata sıfır olduğunda eğitim sürecini durdurmaktadır. Rastlantısal ayarlamalar ise, çevrim sayısı ayarlamalar menüsünde bulunan " Cycle " sayısına ulaştığında durmaktadır. " RUN " tuşu ile başlatılan eğitim süreci sırasında, bu tuş " STOP " tuşuna dönüşmektedir. Kullanıcı böylece " STOP " tuşunu aktif hale getirerek eğitim sürecini durdurabilmektedir. Eğitim süreci bittiğinde elde edilmiş olan şablonun kaydedilmesi gerekmektedir.

7.4. Ekran Görüntüsü

SLAT programının ekranı Şekil 7-1 'de görüldüğü gibi dört bölümden oluşmaktadır. En üstteki bölüm menü, soldaki bölüm şablon, sağdaki bölüm görüntü ve en alttaki bölüm de veri bloğudur.

7.4.1. Menü Bloğu

Menü bloğu, altı tuştan oluşmaktadır. Lacivert tuş, klavyede "ENTER" tuşuna basılarak aktif hale getirilebilmektedir. Klavyede bulunan sağ ve sol ok tuşları ile tuş seçimi yapılabilmektedir.



Şekil 7.1 SLAT Programının Ekran Görüntüsü

Menü bloğunun ilk tuşu " LOAD " tuşudur. Bu tuş aktif hale getirildiğinde, " Training Set " ve " Template " alt menüleri ekranda görünecektir. " Training Set " menüsünden eğitim kümesi ve " Template " menüsünden de başlangıç şablonu yüklenebilmektedir. Her iki alt menüde de klavyedeki tüm yön tuşları ile " HOME ", " END ", " PAGE UP " ve " PAGE DOWN " tuşları kullanılabilir.

" SAVE " tuşu veri saklama menüsünü aktif hale getirmektedir. Böylece eğitim süreci esnasında veya sürecin sonunda önemli veriler bilgisayarda saklanabilmektedir. " Outputimage " alt menüsü kullanılarak mevcut kararlı-durum çıkışı ve " Template " alt menüsü kullanılarak da mevcut şablon saklanabilir. " Setup " alt menüsünden de, eğitim süreci için yapılmış olan ayarlamalar kaydedilebilmektedir. Herbir alt menünün aktif hale gelmesi ile kullanıcıdan saklanacak dosyanın ismi istenmektedir. Dosya isminin uzunluğu en fazla sekiz karakter, uzantısı ise üç

karakter uzunluğunda olabilmektedir. Eğitim süreci esnasında yapılması istenen kayıtlar için, ilk önce " STOP " tuşu ile eğitim sürecine ara verilmesi gerekmektedir. Gerekli kayıtlar yapıldıktan sonra, " RUN " tuşu ile eğitim sürecine kalındığı yerden devam edilebilmektedir.

Ana menünün üçüncü tuşu " RUN " tuşudur. Eğitim süreci bu tuş ile başlatılır. Eğitim süreci başladıktan sonra aynı tuş, sürecin durdurulmasına yarayan " STOP " tuşuna dönüşecektir.

" SETUP " tuşu, eğitim süreci için gerekli tüm ayarlamalarının yapıldığı alt menüyü aktif hale getirmektedir. " Algorithm " menüsünden RPLA veya RBLA öğrenme algoritması seçilebilmektedir. " Method " menüsünden, deterministik veya rastlantısal uyarlamaların seçimi yapılabilmektedir. " Simulation " menüsünden de kararlı-durum çıkışlarının hesaplanması için kullanılması istenen analiz yöntemleri seçilebilmektedir. Bu yöntemler " Euler " , " Heun " ve "Dördüncü Dereceden Runge-Kutta " yöntemleridir. Bu üç alt menüde yeşil renkli seçenek aktif haldedir. Öğrenme algoritması " Incosh " algoritması ise " Beta " menüsü ile β değeri ayarlanabilmektedir. Eğitim sürecinin $\beta = 3$ değeri için koşturulması ve sürecin ileri çevrimlerinde veya sonunda β değerinin birer birer arttırılarak sürecin devam ettirilmesi tavsiye edilmektedir. " Learningrate " alt menüsünden öğrenme oranı ayarlanabilmektedir. Eğitim süreci esnasında, şablonda büyük değişimlere izin verilmediği için, öğrenme oranı azalabilmektedir. " Cycle " menüsü ise sadece rastlantısal algoritmaların uyarlamaları için gereklidir. Buradan eğitim sürecinin kaç çevrim süreceği ayarlanabilmektedir.

Ana menünün son tuşu ise " EXIT " tuşudur. Bu menüde iki seçenek bulunmaktadır. İlki programdan çıkmaya yarayan " Quit ", ikincisi ise programı başlangıç konumuna getiren " RESET " tuşudur. " RESET " tuşu ile tüm başlangıç ayarlamaları yüklenmekte ve program yeni bir eğitim sürecini başlatmaya hazır durumuna gelmektedir.

7.4.2. Şablon Bloğu

Şablon bloğunun sol tarafı eğitim süreci boyunca şablonu, sağ tarafı ise şablonda oluşan değişimi göstermektedir. Şablon değeri azaldıysa karşı düşen hücre kırmızı, arttıysa yeşil olmaktadır. Değişim yoksa, hücre taban rengi olan griye dönüşmektedir.

7.4.3. Görüntü Bloğu

Görüntü bloğu dört parçadan oluşmaktadır. Sol üst tarafta giriş görüntüsü, sağ üst tarafta başlangıç görüntüsü, sol alt tarafta istenen kararlı-durum çıkış görüntüsü ve sağ alt tarafta da benzetim süreci boyunca oluşan gerçek çıkış görüntüsü gösterilmektedir. Çıkış kararlı duruma ulaştığında, görüntüsü siyah-beyaz duruma dönüşmektedir.

7.4.4. Veri Bloğu

Veri bloğu eğitim sürecinde değişen verileri göstermektedir. Sağ tarafında sırasıyla, kullanılmakta olan algoritma uyarlaması (deterministik veya rastlantısal), öğrenme oranı, çevrim sayısı ve o anda işlenen görüntünün numarası gösterilmektedir. Sol tarafında ise hataların gösterildiği bölüm bulunmaktadır. Algoritmaların rastlantısal uyarlamaları için, altta o anki görüntüde oluşan hata, üstte ise aynı görüntünün bir önceki simülasyonunda oluşan hata gösterilmektedir. Deterministik uyarlamalarda ise bunlara ek olarak, o anki toplam hata ve bir önceki çevrimde oluşmuş olan toplam hata ekrana yansımaktadır. Hatada azalma varsa, rakamlar kırmızı olmaktadır.

7.5. Başlangıç Ayarları

SLAT programı, " SLAT.cfg " dosyasından başlangıç ayarlarını okumaktadır. Başlangıç ayarlarında sırasıyla; RPLA öğrenme algoritması, deterministik uyarlama, " Heun " analiz yöntemi kullanılmış ve $\beta = 3$, öğrenme oranı 0.0001, çevrim sayısı 500 olarak alınmıştır. Bu dosyadaki değerler herhangi bir editör ile değiştirilebilir.

P-RPLA Deterministic Heun 3.0 0.0001 500

Şekil 7.2 SLAT.cfg Dosyası

7.6. Kütüphaneler

SLAT programı eğitim kümesi kütüphanesini, şablon kütüphanesini ve görüntü kütüphanesini kullanmaktadır.

7.6.1. Eğitim Kümesi Kütüphanesi

Eğitim kümesi kütüphanesinde eğitim kümeleri bulunmaktadır. Bu kütüphane "TSE" alt dizininde bulunmaktadır. Eğitim kümeleri kullanıcı tarafından da yaratılabilmektedir. Bunun için, SLAT programı ile birlikte sunulan " E " editörü veya dosyaları text modda oluşturan herhangi bir editör kullanılabilir. Uyulması gereken format Şekil 7.3 'de verilmiştir. İlk olarak eğitim kümesinde bulunan örneklerin sayısının verilmesi gerekmektedir. Her bir örnek sırasıyla, başlangıç görüntüsünün

dosya ismi, giriş görüntüsünün dosya ismi ve istenen çıkış görüntüsünün dosya isminden oluşmaktadır. Dosya isimleri, varsa, uzantıları ile birlikte verilmelidir. Eğitim kümesinin dosya ismi ise mutlaka " TSE " uzantılı olmalıdır. Şekil 7.3 'de verilmiş olan eğitim kümesi örneğinde dört tane eğitim örneği bulunmaktadır. Kümenin ilk örneğinde, "image1" dosyası başlangıç görüntüsü ve giriş görüntüsü olarak, "dimage1" dosyası da istenen çıkış görüntüsü olarak tanımlanmıştır. Belirtilen görüntü dosyaları mutlaka " İMAGES " alt dizininde bulunmalıdır.

Deterministik uyarlamada, seçilmiş olan eğitim kümesinde bulunan örnekler sırasıyla işlenmektedir. Her bir çevrim için eğitim kümesinin başından başlanılmaktadır. Rastlantısal uyarlamada ise örnekler rastgele seçilmekte ve her bir çevrim tek bir örnekten oluşmaktadır.

4
image1 image1 dimage1
image2 image2 dimage2
image3 image3 dimage3
image4 image4 dimage4

Şekil 7.3 Eğitim Kümesi Örneği

7.6.2. Şablon Kütüphanesi

Şablon kütüphanesinde eğitim sürecinin ilk çevriminde kullanılması gereken başlangıç şablonları ve " SAVE " menüsünden kullanıcı tarafından kaydedilen şablonlar bulunmaktadır. Ayrıca SLAT programı, eğitim süreci boyunca en az hatayı veren şablonu " MİN.TEM " adıyla bu kütüphaneye yüklemektedir. Bu kütüphane " TEMPLATE " alt dizininde bulunmaktadır.

Şekil 7.4 ile bir şablon örneği verilmiştir. " Neighborhood: " ile HYSA 'nın yakın komşuluk boyutu belirtilmektedir. SLAT sadece 1 yakın komşuluk boyutu için çalışmaktadır. " Feedback: " ile geribesleme şablonu , " Control: " ile kontrol şablonu

ve " Current: " ile eşik şablonu belirtilmektedir. Buralarda " : " 'dan sonra mutlaka bir boşluk bulunmalıdır. Kullanıcı aynı formatta kendi şablonunu oluşturabilir. Bu amaç için yine " E " editörü veya eşdeğer bir editör kullanılabilir. Şablon dosyalarının isimleri mutlaka " TEM " uzantılı olmalıdır. Ayrıca geribesleme ve kontrol şablonları simetrik olmalıdır.

```

Neighborhood: 1

Feedback:  0 0 0
           0 4 0
           0 0 0

Current: -1

Control:   1 1 1
           1 3 1
           1 1 1

```

Şekil 7.4 Şablon Dosyası Örneği

7.6.3. Görüntü Kütüphanesi

Görüntü kütüphanesi " İMAGES " alt dizininde bulunmaktadır. Görüntü dosyaların formatı Şekil 7.5 ile verilmiştir. Görüntünün boyutları başta belirtilmiştir. Daha sonra [-1,1] aralığında bulunan değerler ile görüntü sol üst köşeden başlayarak satır satır tanımlanmaktadır. Görüntü oluşturma işlemini kolaylaştırmak amacıyla, SLAT programı ile birlikte " EDCNİ " görüntü çizim programı da sunulmuştur. Bu program ile 44 X 44 boyutlarına kadar görüntüler fare (mouse) yardımı ile oluşturulabilmektedir. Ayrıca var olan görüntüler bu program yardımı ile renkli olarak izlenebilmektedir. " EDCNİ " programı, " CNİ " uzantılı görüntü dosyaları oluşturmaktadır.

BÖLÜM 8

RBLA VE RPLA İLE GÖRÜNTÜ İŞLEME

HYSA 'ların 2 boyutlu ızgara yapısı, görüntü işleme uygulamaları için oldukça elverişlidir. Bu bölümde, SLAT programı ile yapılan benzetimler sonucu değişik görüntü işlemleri yapabilen bağlantı ağırlık katsayıları (şablonlar) sunulacaktır.

İşlenecek görüntü, HYSA 'nın ya dış girişlerine veya başlangıç durumlarına ya da her ikisine birden uygulanabilir. SLAT ile yapılan benzetimlerde 5 tane 16x16 'lık görüntü kullanılmıştır ve işlenecek görüntüler HYSA 'nın dış girişlerine uygulanmıştır. Bulunan şablonlar aynı görüntü işlemeyi yapmak üzere eğitim kümesinin içinde bulunmayan 10 tane şekil üzerinde denenmiştir.

Öğrenme oranı; eğitim süreci boyunca şablon değerlerinin aşırı değişimlerini engelleyecek şekilde azalmakta, eğitim sürecini hızlandırmak için de artmaktadır. Şablon değerlerden herhangi biri 0.1 'den büyük bir değişim gösterdiğinde öğrenme oranı düşmektedir. Öğrenme algoritmalarının deterministik uyarlamalarında, toplam hata uzun süre aynı kaldığında öğrenme oranı artmaktadır.

8.1. RBLA ile Görüntü İşleme

Bu alt bölümde RBLA eğitici öğrenme algoritması ile yapılan benzetimlerin sonuçları sunulacaktır. Bulunan şablonlar kenar ve köşe saptama görüntü işlemlerini yerine getirebilmektedir. (6.7) ile verilen çıkış işlevinde eğitim sürecinin başında β değeri 3 seçilmiştir. Bu değer, istenen görüntü işlemeyi eğitim kümesi için sıfır hata ile yapabilen şablonlar bulunduktan sonra, birer birer 20 'ye kadar arttırılmaktadır ve her yeni β değeri için eğitim tekrarlanmaktadır.

Kenar Saptama :

RBLA ile yapılan kenar saptamaya yönelik benzetimler sonucu birçok uygun şablon bulunmuştur. Bulunan şablonlardan biri (8.1.1) 'de verilmiştir. Bu şablonun bulunmasında RBLA 'nın deterministik uyarlaması kullanılmıştır. Öğrenme oranı başlangıçta $\varepsilon(0) = 1000$ seçilmiş ve eğitim süreci boyunca $[10^{-2}, 10^{12}]$ aralığında değişimler göstermiştir.

$$A = \begin{bmatrix} -0.003336 & -0.099059 & -0.038978 \\ -0.058992 & 3.690972 & -0.058992 \\ -0.03897 & -0.099059 & -0.003336 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.247333 & -0.247333 & -0.298196 \\ -0.247333 & -0.123666 & -0.247333 \\ -0.298196 & -0.247333 & -0.247333 \end{bmatrix}, \quad I = -0.319689 \quad (8.1.1)$$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad I=0 \quad (8.1.2)$$

Eğitim süreci (8.1.2) ile verilen başlangıç şablonu ile başlamıştır ve 79 çevrim sonunda bitmiştir. Şekil 8.1 ile benzetimin değişik çevrimlerinden örnekler verilmiştir. Şekil 8.1.a 'da eğitim süreci için HYSA 'nın dış girişlerine uygulanan 5 görüntü gösterilmiştir. Siyah bir hücre +1 değerine, beyaz bir hücre ise -1 değerine karşı düşmektedir. Şekil 8.1.b 'de başlangıç durum görüntüleri ve Şekil 8.1.g 'de de istenen çıkış görüntüleri verilmiştir. Altı eğitim çevrimi boyunca başlangıç durum görüntüleri değişmemektedir. 5 şekil için toplam hata bu durumda 298 olmaktadır. ondördüncü çevrim Şekil 8.1.c ile verilmiştir. Her bir şekil için hatalar sırasıyla 36,0,13, 31,31 olmakta, yani toplam hata 111 olmaktadır. Şekil 8.1.d 'de yirmibeşinci çevrim verilmiştir. Her bir şekil için hatalar sırasıyla 28, 20, 13, 37, 29 olmakta, yani toplam hata 127 olmaktadır. Şekil 8.1.e ile verilen otuzüçüncü çevrimdeki toplam hata 115 olduktan sonra, otuzsekizinci çevrimde hatalar sırasıyla 0, 4, 0, 2, 3 olmakta, yani toplam hata 9 olmaktadır. otuzdokuzuncü çevrimden sonra, toplam hata 0 olmakta ve çıkış görüntüleri istenen çıkış görüntülerine eşit olmaktadır (Şekil 8.1.g). Daha sonra β , başlangıçtaki 3 değerinden birer birer artırılarak 20 'ye çıkartılmıştır. Toplam 79

çevrim sonunda (8.1.1) 'de verilen şablon bulunmuştur. Kullanılan eğitim kümesi (Şekil 8.1.a) için kenar saptama yapan bu şablon, Şekil 8.2 'de verilen 10 görüntü için denenmiştir. Elde edilen görüntüler sıfır hata ile bu görüntülerin kenarları olmuştur (Şekil 8.3).

Köşe Saptama :

RBLA ile yapılan köşe saptamaya yönelik benzetimler sonucu bulunan şablonlardan biri (8.1.3) 'de verilmiştir. Bu şablonun bulunmasında RBLA 'nın deterministik uyarlaması kullanılmıştır. Öğrenme oranı başlangıçta $\varepsilon(0) = 1000$ seçilmiş ve eğitim süreci boyunca $[10^{-2}, 10^9]$ aralığında değişimler göstermiştir.

$$A = \begin{bmatrix} 0.223622 & 0.153554 & 0.165183 \\ 0.129503 & 3.690972 & 0.129503 \\ 0.165183 & 0.153554 & 0.223622 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.363606 & -0.401270 & -0.285965 \\ -0.471105 & -0.555354 & -0.471105 \\ -0.285965 & -0.401270 & -0.363606 \end{bmatrix}, \quad I = -0.649645 \quad (8.1.3)$$

Eğitim süreci (8.1.2) ile verilen başlangıç şablonu ile başlamıştır ve 127 çevrim sonunda sona ermiştir.

8.2. RPLA ile Görüntü İşleme

Bu alt bölümde RPLA eğiticili öğrenme algoritması ile yapılan benzetimlerin sonuçları sunulacaktır. Bulunan şablonlar kenar ve köşe saptama ile boşluk doldurma görüntü işlemlerini yerine getirebilmektedir.

Kenar Saptama :

RPLA ile yapılan kenar saptamaya yönelik benzetimler sonucu birçok uygun şablon bulunmuştur. Bulunan şablonlardan biri (8.2.1) 'de verilmiştir. Bu şablonun

bulunmasında RPLA 'nın deterministik uyarlaması kullanılmıştır. Öğrenme oranı başlangıçta $\varepsilon(0) = 0.0001$ seçilmiş ve eğitim süreci boyunca $[10^{-4}, 10^{-5}]$ aralığında değişimler göstermiştir.

$$A = \begin{bmatrix} -0.096934 & -0.192281 & -0.088861 \\ -0.179305 & 3.806794 & -0.179305 \\ -0.088861 & -0.192281 & -0.09693 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.203938 & -0.203938 & -0.203938 \\ -0.203938 & -0.101969 & -0.203938 \\ -0.203938 & -0.203938 & -0.203938 \end{bmatrix}, \quad I = -0.193207 \quad (8.2.1)$$

Eğitim süreci (8.1.2) ile verilen başlangıç şablonu ile başlamıştır ve 17 çevrim sonunda bitmiştir. Şekil 8.5 ile benzetimin değişik çevrimlerinden örnekler verilmiştir. Şekil 8.5.a 'da eğitim süreci için HYSA 'nın dış girişlerine uygulanan 5 görüntü gösterilmiştir. Şekil 8.5.b 'de başlangıç durum görüntüleri ve Şekil 8.5.g 'de de istenen çıkış görüntüleri verilmiştir. Eğitim sürecin ilk 5 çevrimi boyunca HYSA 'nın çıkışları Şekil 8.5.b 'de gösterildiği gibi kalmıştır. Toplam hata 298 olmuştur. Altıncı çevrim sonunda toplam hata 131 'e inmiştir. Görüntülerin hataları sırasıyla 40, 0, 13, 45, 33 (Şekil 8.5.c) olmuştur. Sekizinci çevrim sonunda toplam hata 118 olmak üzere, görüntüleri hataları sırasıyla 36, 4, 13, 34, 31 olmuştur (Şekil 8.5.d). Dokuzuncu çevrim sonunda toplam hata büyük bir düşüş gösterip 33 'e düşmüştür. Görüntüleri hataları sırasıyla 0, 20, 0, 9, 4 olmuştur (Şekil 8.5.e). Onüçüncü çevrimden onaltıncı çevrime kadar toplam hata 9 'da kalmıştır. Görüntülerin hataları sırasıyla 0, 4, 0, 2, 3 olmuştur (Şekil 8.5.f). Eğitim sürecin onyedinci çevriminden sonra Şekil 8.5.g ile verilen istenen görüntüler elde edilmiştir.

Elde edilen şablon Şekil 8.2 ile verilen test görüntüler üzerinde denenmiştir. Eğitim kümesinin dışında bulunan bu 10 görüntü için kenar saptama işlemi hatasız gerçekleşmiştir. Şekil 8.3 ile test görüntülerinin kenarları verilmiştir.

RPLA 'nın rastlantısal uyarlaması ile yapılan benzetimler sonucu, (8.2.3) ile verilen şablon elde edilmiştir. Kenar saptama yapabilen bu şablon, 172 çevrim sonunda bulunmuştur. Başlangıç şablonu olarak yine, (8.1.2) ile verilen şablon

alınmıştır. Öğrenme oranı başlangıçta $\varepsilon(0) = 0.0001$ seçilmiş ve eğitim süreci boyunca değişmemiştir. Bulunan şablon 10 tane test görüntüsü için hatasız çalışmıştır.

$$A = \begin{bmatrix} -0.174187 & -0.174187 & -0.174187 \\ -0.174187 & -3.905828 & -0.174187 \\ -0.174187 & -0.174187 & -0.174187 \end{bmatrix},$$

$$B = \begin{bmatrix} -1.883433 & -1.883433 & -1.883433 \\ -1.883433 & -0.094172 & -1.883433 \\ -1.883433 & -1.883433 & -1.883433 \end{bmatrix}, \quad I = -0.094172 \quad (8.2.3)$$

Köşe Saptama :

RPLA ile yapılan köşe saptamaya yönelik benzetimler sonucu bulunan şablonlardan biri (8.2.4) 'de verilmiştir. Bu şablonun bulunmasında RPLA 'nın deterministik uyarlaması kullanılmıştır. Öğrenme oranı başlangıçta $\varepsilon(0) = 0.0001$ seçilmiş ve eğitim süreci boyunca $[10^{-3}, 10^{-4}]$ aralığında değişimler göstermiştir.

$$A = \begin{bmatrix} 0.173943 & 0.193183 & 0.175125 \\ 0.134455 & 3.252625 & 0.134455 \\ 0.175125 & 0.193183 & 0.173943 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.387946 & -0.459971 & -0.381612 \\ -0.511623 & -0.612406 & -0.511623 \\ -0.381612 & -0.459971 & -0.387946 \end{bmatrix}, \quad I = -0.649522 \quad (8.2.4)$$

Eğitim süreci (8.1.2) ile verilen başlangıç şablonu ile başlamıştır ve 42 çevrim sonunda sona ermiştir. Şekil 8.6 ile benzetimin değişik çevrimlerinden örnekler verilmiştir. Şekil 8.6 'deki görüntüler ile sırasıyla, HYSA 'nın dış girişlerine uygulanan 5 görüntü, başlangıç durum görüntüleri, beşinci çevrime ait görüntüler, onuncu çevrime ait görüntüler, onsekizinci çevrime ait görüntüler, otuzdördüncü çevrime ait görüntüler ve son olarak kırkikinci çevrime ait görüntüler, yani istenen çıkış görüntüler gösterilmiştir. Toplam hata ilk üç çevrim için 544 ve görüntüler Şekil 8.6.a 'daki gibidir. Şekil 8.6.c 'den itibaren toplam hatalar sırasıyla 246, 201, 138, 33 ve 0 olmuştur.

Elde edilen şablon Şekil 8.2 ile verilen test görüntüler üzerinde denenmiştir. Eğitim kümesinin dışında bulunan bu 10 görüntü için köşe saptama işlemi toplam altı hata ile gerçekleşmiştir. Şekil 8.4 ile test görüntülerinin köşeleri verilmiştir.

RPLA 'nın deterministik uyarlaması ile yapılan benzetimlerden birinde de başlangıç durum görüntüleri olarak beyaz görüntüler alınmıştır (-1 değerlerinden oluşan görüntüler). Bulunan şablon (8.2.5) ile verilmiştir. Eğitim süreci (8.1.2) ile verilen başlangıç şablonu ile başlamıştır. Öğrenme oranı başlangıçta $\varepsilon (0) = 0.0001$ seçilmiş ve eğitim süreci boyunca $[10^{-4}, 10^{-5}]$ aralığında değişimler göstermiştir.

$$A = \begin{bmatrix} -0.265233 & -0.343507 & -0.264432 \\ -0.346131 & 3.734485 & -0.346131 \\ -0.264432 & -0.343507 & -0.265233 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.048049 & -0.025063 & -0.048049 \\ -0.027559 & 3.734485 & -0.027559 \\ -0.048049 & -0.025063 & -0.048049 \end{bmatrix}, \quad I = -0.129907 \quad (8.2.5)$$

Elde edilen şablon Şekil 8.2 ile verilen test görüntüler üzerinde denenmiştir. Eğitim kümesinin dışında bulunan bu 10 görüntü için köşe saptama işlemi hatasız gerçekleşmiştir. Şekil 8.4 ile test görüntülerinin köşeleri verilmiştir

Boşluk Doldurma :

RPLA ile yapılan boşluk doldurmaya yönelik benzetimler sonucu bulunan şablonlardan biri (8.2.6) 'da verilmiştir. Bu şablonun bulunmasında RPLA 'nın deterministik uyarlaması kullanılmıştır. Eğitim süreci (8.2.7) ile verilen başlangıç şablonu ile başlamıştır. Öğrenme oranı başlangıçta $\varepsilon (0) = 0.0001$ seçilmiş ve eğitim süreci boyunca $[10^{-3}, 10^{-4}]$ aralığında değişimler göstermiştir.

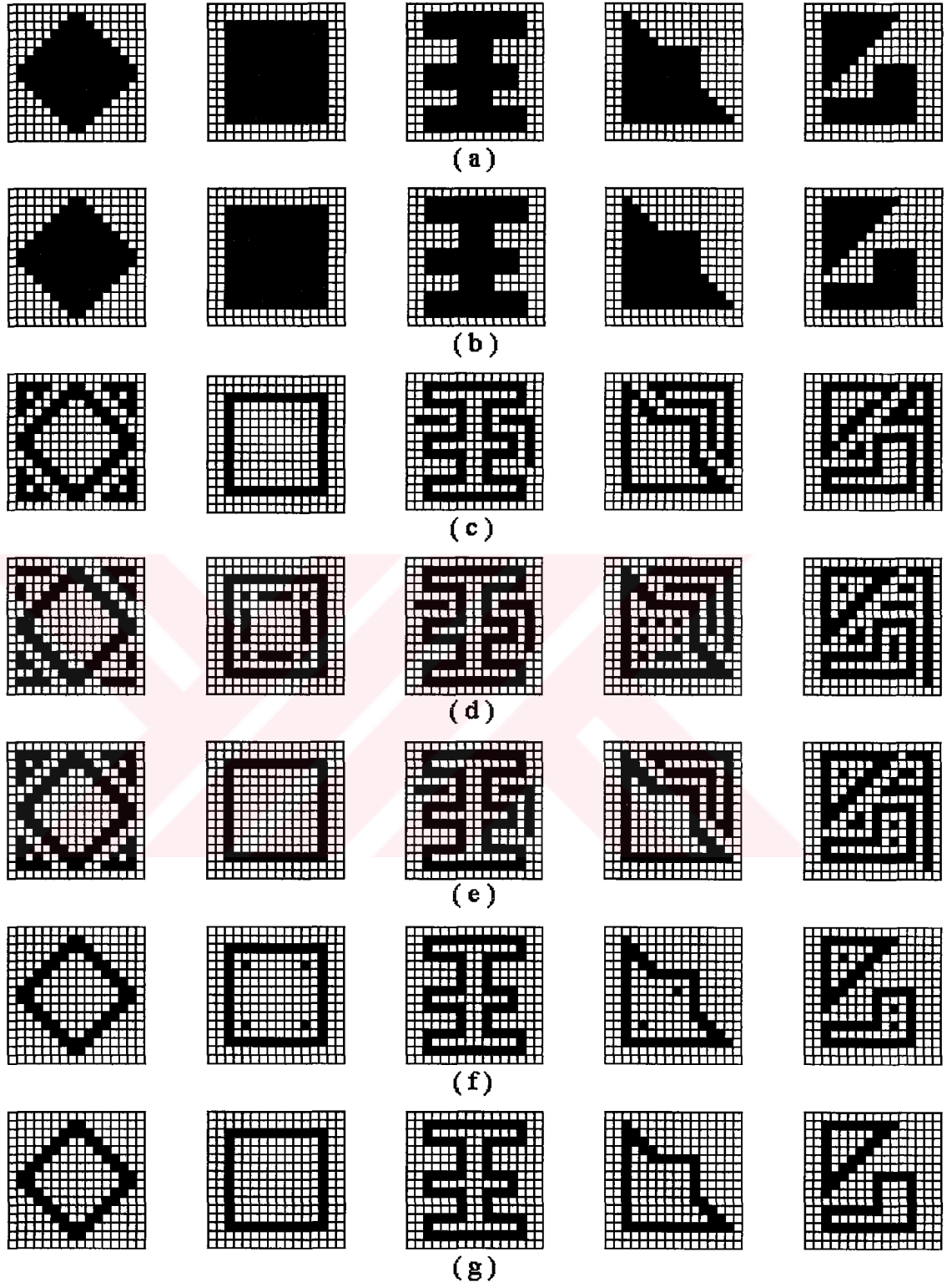
$$A = \begin{bmatrix} 0.422643 & 0.276928 & 0.383840 \\ 0.437143 & 3.512904 & 0.437143 \\ 0.383840 & 0.276928 & 0.422643 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.006490 & 0.154091 & -0.017128 \\ 0.293609 & 4.328963 & 0.293609 \\ -0.017128 & 0.154091 & -0.006490 \end{bmatrix}, \quad I = -0.328994 \quad (8.2.6)$$

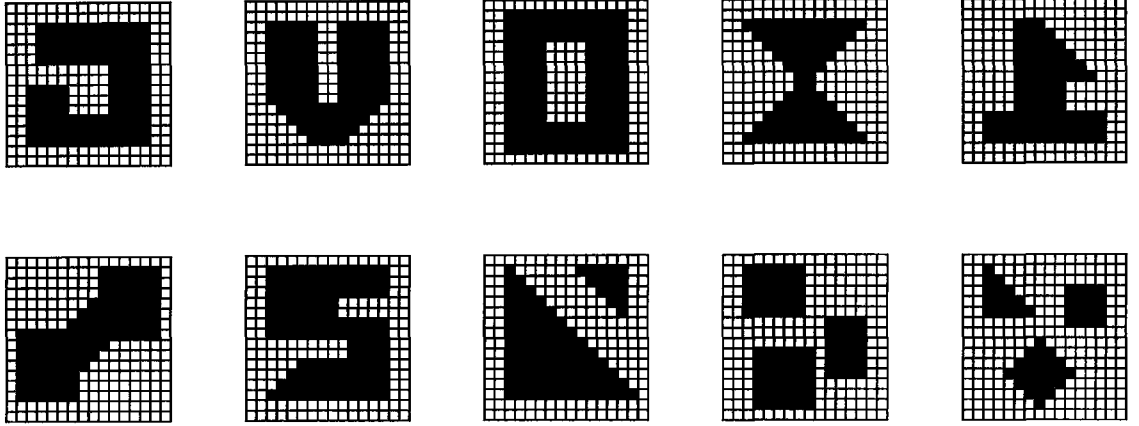
$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad I=0 \quad (8.2.7)$$

Şekil 8.7 ile boşluk doldurma eğitim sürecinden örnekler verilmiştir. Başlangıç görüntüleri siyah seçilmiştir (Şekil 8.7.a). HYSA 'nın dış girişlerine Şekil 8.7.b 'deki görüntüler uygulanmıştır. Diğer şekiller sırasıyla yirmiyedinci, otuzdokuzuncu, elliikinci, elliyedinci ve altmışbirinci, yani son çevrime aittir. Toplam hatalar sırasıyla 61, 39, 22, 4 ve 0 olmuştur.

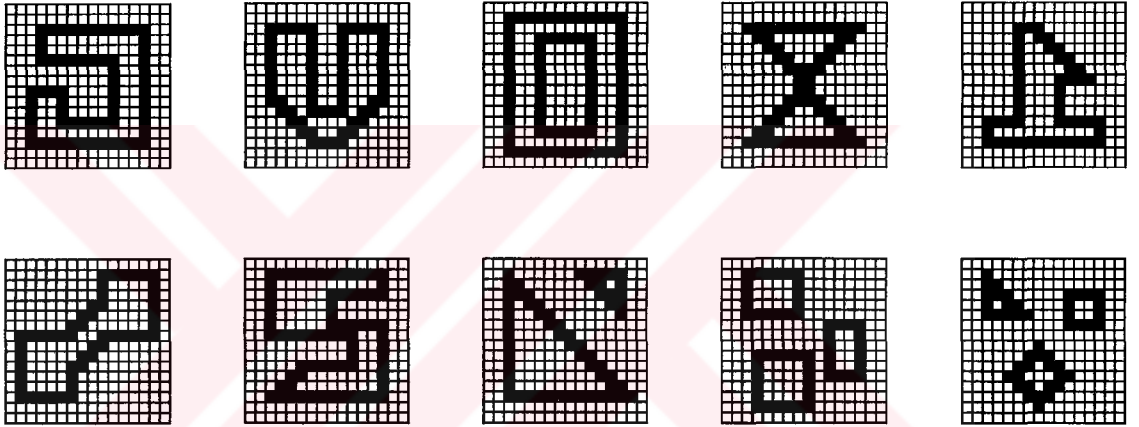
Elde edilen şablon test görüntüleri üzerinde denenmiştir. İşlenecek görüntüler olarak Şekil 8.2 'deki test görüntülerinin kenarları alınmıştır (Şekil 8.3). Başlangıç görüntüleri, eğitim sürecinde alındığı gibi, zorunlu olarak siyah görüntüler şeklinde alınmıştır. Elde edilen görüntüler, üçüncü görüntü hariç Şekil 8.2 'deki görüntüler gibi olmuştur. Üçüncü görüntünün en iç bölümü de doğal olarak siyaha dönüşmüştür.



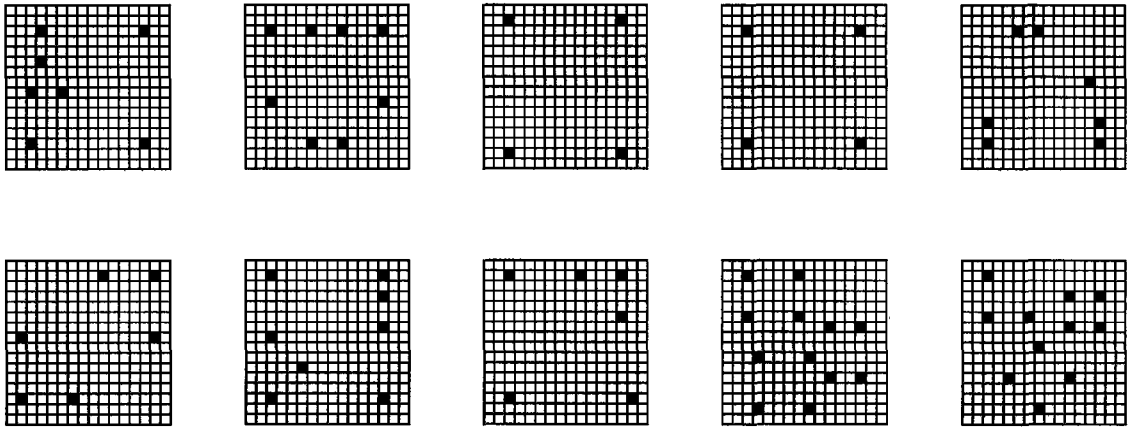
Şekil 8.1 RBLA ile Kenar Saptama



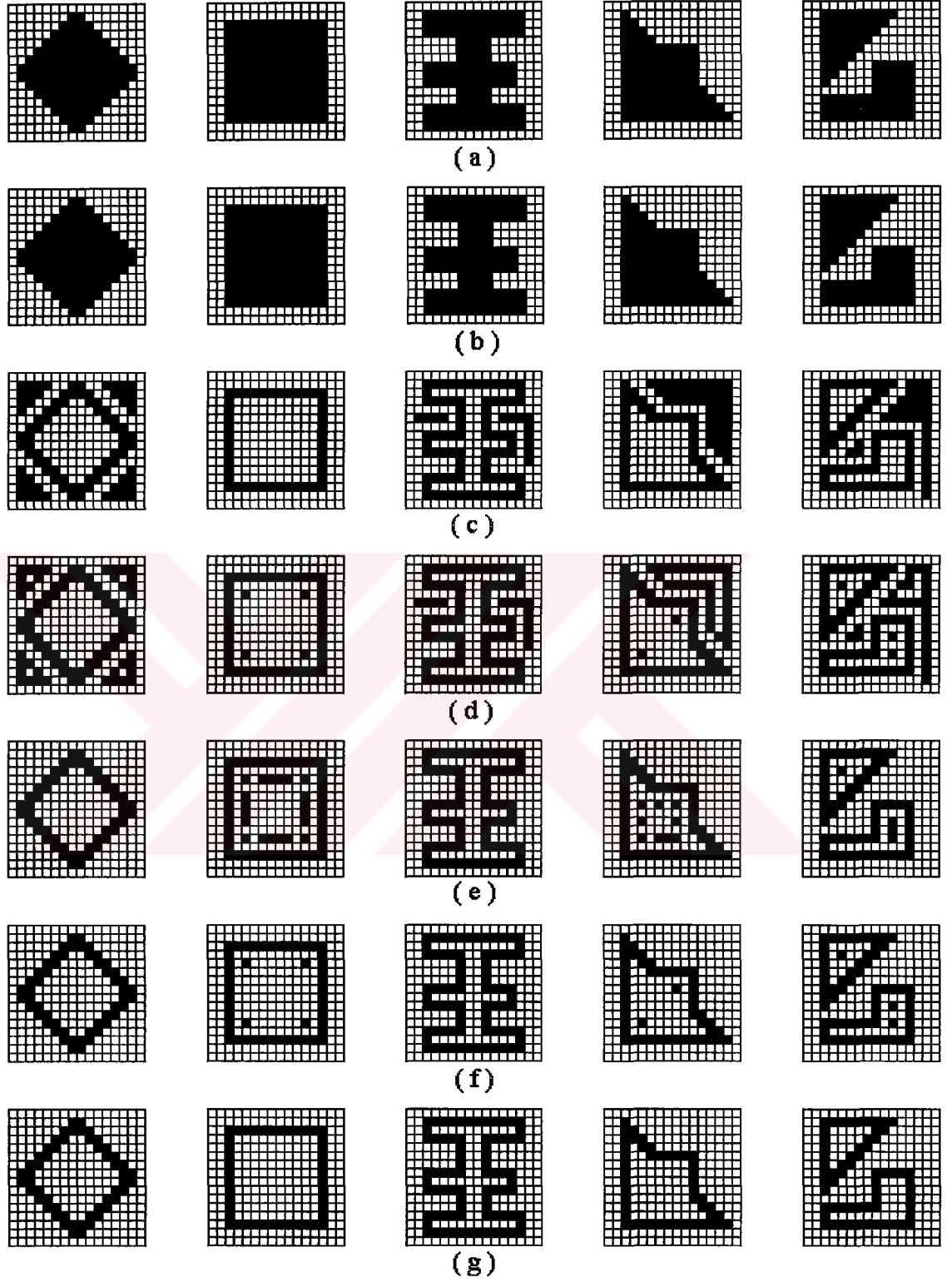
Şekil 8.2 Test Görüntüleri



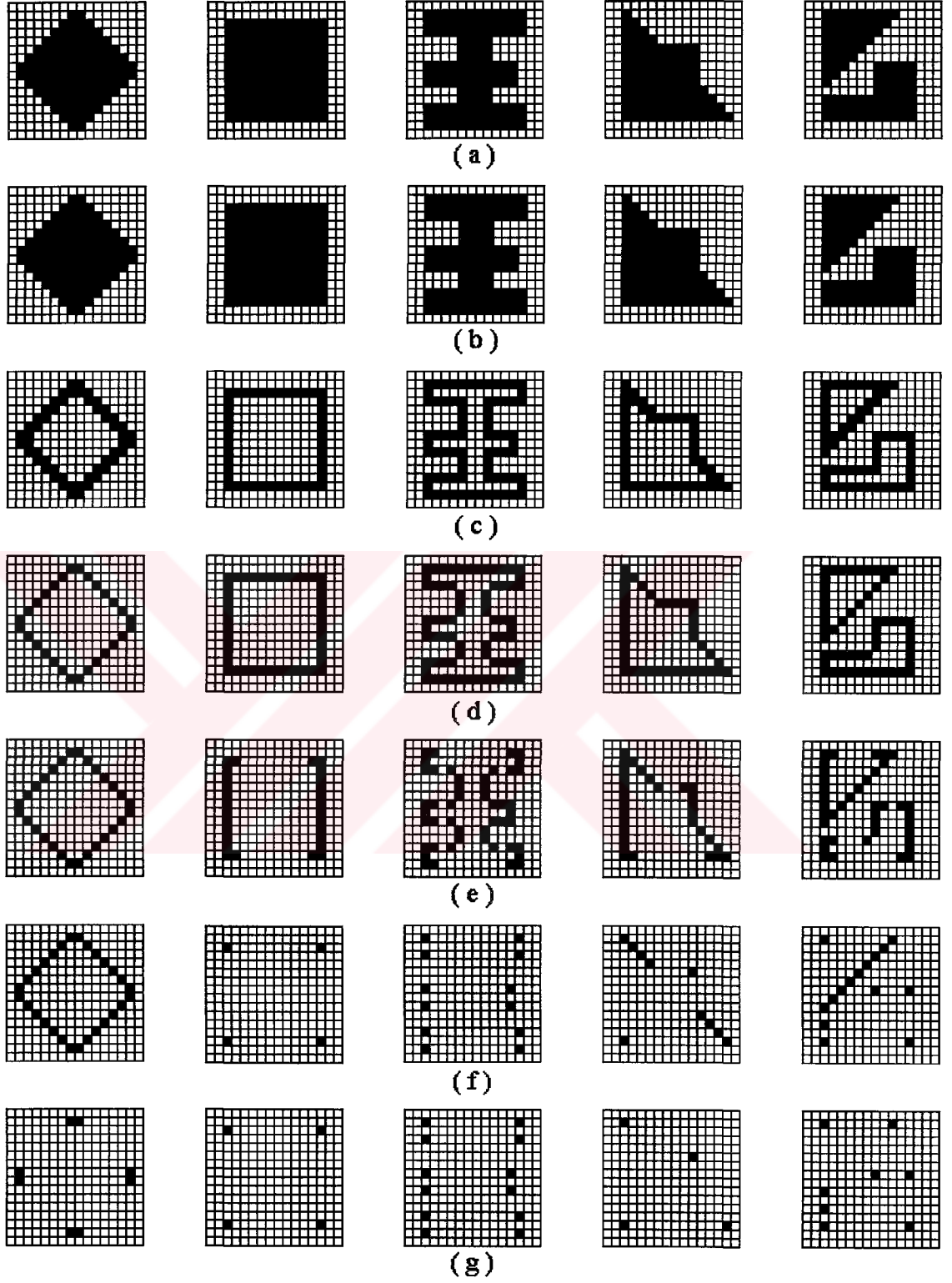
Şekil 8.3 Test Görüntülerinin Kenarları



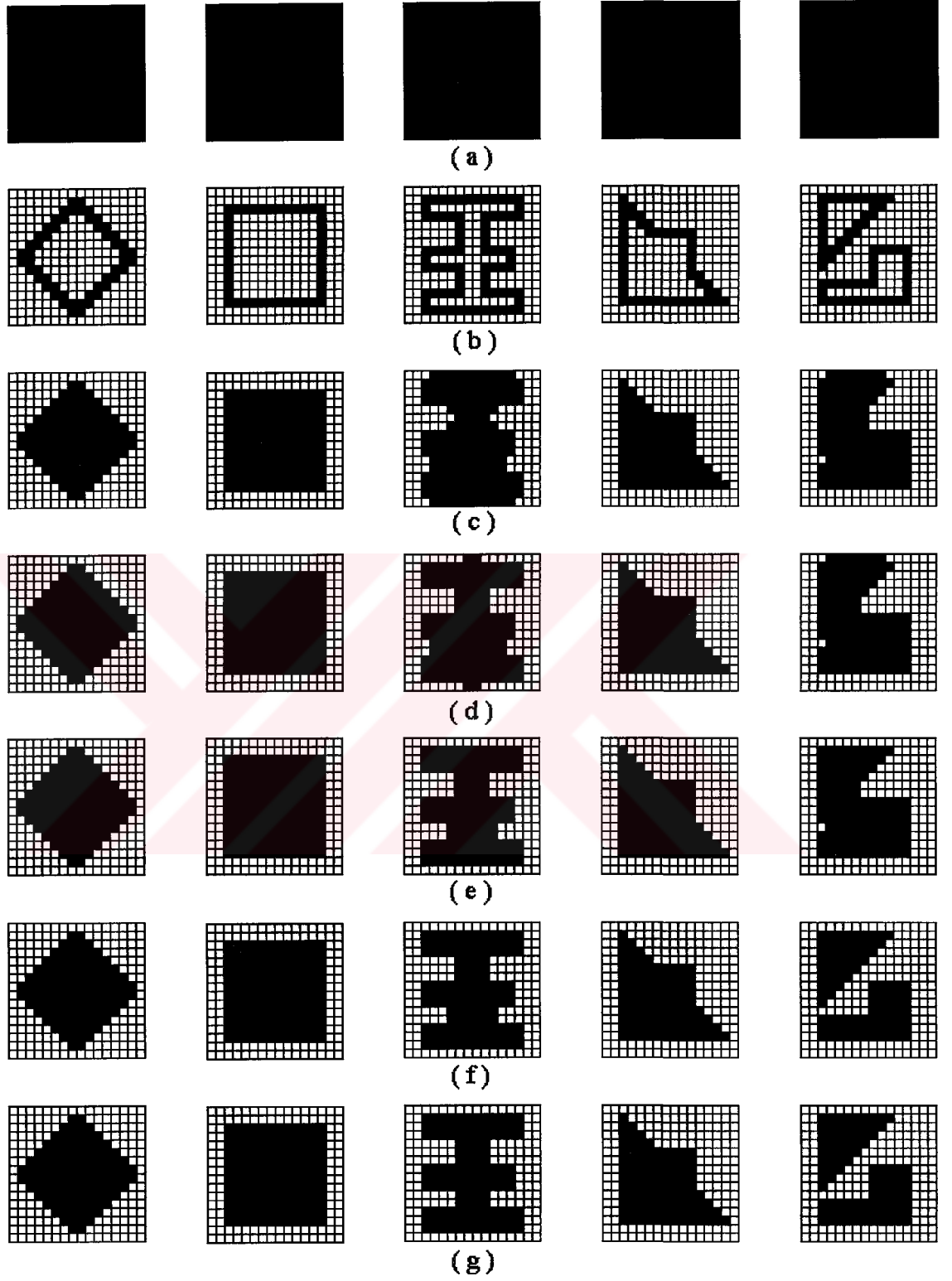
Şekil 8.4 Test Görüntülerinin Köşeleri



Şekil 8.5 RPLA ile Kenar Saptama



Şekil 8.6 RPLA ile Köşe Saptama



Şekil 8.7 RPLA ile Boşluk Doldurma

SONUÇ VE ÖNERİLER

Bu tezde, tam kararlı Hücrel Yapay Sinir Ağları (HYSA) için istenen kararlı hal çıkışlarının eğitici öğretilmesi amacıyla yönelik olarak Dinamik Algılayıcı Öğrenme Algoritması ("Recurrent Perceptron Learning Algorithm : RPLA ") ve Dinamik Geriye-Yayılım Öğrenme Algoritması (" Recurrent Backpropagation Learning Algorithm : RBLA ") adında iki farklı eğitici öğrenme algoritması geliştirilmiştir. Bu algoritmaları kullanan, HYSA 'lar için şablon öğrenme programı olan SLAT (" Supervised Learning Algorithm Tool ") programı yazılarak, bir kişisel bilgisayar ortamında benzetim düzeni elde edilmiştir.

Geliştirilen algoritmalar ile, HYSA 'ları kenar ve köşe saptama ile boşluk doldurma gibi temel görüntü işleme problemlerini gerçeklemek üzere eğitilmişlerdir. Bu algoritmalar, $[-1, +1]^m$ kümesinden $\{-1, +1\}^m$ kümesine cebrik bir işlev ile tanımlanabilen herhangi bir görüntü işleme problemine uygulanabilir.

Daha karmaşık görüntü işleme işlerini gerçekleyen şablonları (bağlantı ağırlık katsayıları) da içeren bir şablon kütüphanesi oluşturmak, çok geniş çapta tümleşik devre tasarımına uygun ve gürbüz (" robust ") şablonlar elde etmek, bu alandaki yapılabilecek bir sonraki araştırmaların konusunu oluşturmaktadır.

Tezde sunulan RBLA ve RPLA algoritmaları, uygulamaya özgü tasarıma yönelik veya HYSA 'ların uyarlamalı (" adaptif ") eğitilmesinde kullanılabilir.

KAYNAKLAR

- [1] DARPA Neural Network Study. Fairfax, VA : AFCEA International Press, 1988.
- [2] GÜZELİŞ, C., Genelleştirilmiş Hücresel Yapay Sinir Ağları, Elektrik Mühendisliği 5. Ulusal Kongresi, Trabzon, pp.7-12, 1993.
- [3] CHUA, L.O. ,and YANG, L., Cellular Neural Networks : Theory, IEEE Transaction on Circuits and Systems, vol.35. pp.1257-1272,October 1988.
- [4] CHUA, L.O.,and YANG, L., Cellular Neural Networks : Application, IEEE Transaction on Circuits and Systems, vol.35. pp.1273-1290, October 1988.
- [5] CHUA, L.O.,ROSKA,T., VENETIANER,and P.L.,ZARANDY, A.," Some Novel Capabilities of CNN : Game of Life and Examples of Multipath Algorithms ", In Proc.IEEE 2 nd int. Workshop on Cellular Neural Networks, pp.276-281, 1992.
- [6] MATSUMATO,T., CHUA, L.O.,and SUZUKI, H., " CNN Cloning Template :Connected Component Detector ", IEEE Transaction on Circuits and Systems, vol 37, No. 5, pp. 633-635, May 1990.
- [7] MATSUMATO,T., CHUA, L.O.,and FURUKAWA, R., " CNN Cloning Template : Hole Filler ", IEEE Transaction on Circuits and Systems, vol 37, No. 5, pp.635-638, May 1990.
- [8] MATSUMATO, T., CHUA, L.O. ,and YOKOHAMA,T., " Image Thinning in Cellular Neural Network ", IEEE Transaction on Circuits and Systems, vol 37, No. 5, pp. 638-642, May 1990.

- [9] CHUA, L.O.,and THIRAN, R., " An Analytic Method for Designing Simple Cellular Neural Networks " ,IEEE Transaction on Circuits and Systems, vol 38, No.11, pp. 1332-1340, November 1991.
- [10] ZOU, F., SCHWARZ,S.,and NOSSEK, J.A., " Cellular Neural Network Design Using a Learning Algorithm ", In Proc. IEEE Int. Workshop on Cellular Neural Networks and their Applications, pp.73-81, 1990.
- [11] SCHULER, A.J., NACHBAR, P., NOSSEK, and J.A., CHUA, L.O., "Learning State Space Trajectories in Cellular Neural Networks ", In Proc. IEEE 2 nd int. Workshop on Cellular Neural Networks, pp. 68-73, 1992.
- [12] MAGNUSSEN, H.,and NOSSEK, J.A., " Towards a Learning algorithm for Discrete-time Cellular Neural Networks ", In Proc. IEEE 2 nd int. Workshop on Cellular Neural Networks, pp. 80-85, 1992.
- [13] M. BALSI, " Generalized CNN : Potentials of an CNN with Non-uniform Weights " , In Proc. IEEE 2 nd int. Workshop on Cellular Neural Networks, pp. 129-134, 1992.
- [14] KARMAHMUT, S.,and GÜZELİŞ, C., " Recurrent Backpropagation Algorithm for Completely Stable Cellular Neural Networks ", Turkish Symposium on Artificial Intelligence and Neural Networks TAINN '94, Ankara, 1994.
- [15] GÜZELİŞ, C.,and KARMAHMUT, S., " Recurrent Perceptron Learning Algorithm for Completely Stable Cellular Neural Networks ", Submitted to IEEE Transaction on Circuits and Systems, Part I: Fundamental Theory and Applications.
- [16] ALMEDA, L., Backpropagation in Perceptrons with Feedback, in Eckmiller, R. and von der Malsberg, Neural Computers, NATO ASI series, F(41) Springer-Verlag, Berlin, 1988
- [17] HECHT-NIELSEN, R., Neurocomputing ,Hnc, Inc. and University of California, San Diego,1989.
- [18] ZURADA, J.M., Introduction to Artificial Neural Systems , West Publishing Company, St. Paul,1991.

- [19] SEVEN, A., "Yapay Sinir Ağları ile Doku Sınıflandırma ", Yüksek Lisans Tezi, İTÜ, Temmuz 1993.
- [20] GÜZELİŞ, C., "Yapay Sinir Ağları, Yüksek Lisans Ders Notları ", İTÜ, 1991.
- [21] Proc. of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA-90), Budapest, 1990.
- [22] Proc. of the 2nd IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA-92), Budapest, 1992.
- [23] Proc. of Special Session on Cellular Neural Networks, European Conference on Circuit Theory and Design (ECCTD-91), October 1991.
- [24] Int. J. Circuit Theory and Application, Special Issue on Cellular Neural Networks, October-November 1992.
- [25] IEEE Transaction on Circuits and Systems, Special Issue on Cellular Neural Networks, March 1992.
- [26] BAKLAVACI, S., "Yapay Sinir Ağlarında Öğrenme Algoritmalarının Analizi", Yüksek Lisans Tezi, İTÜ, Ocak 1994.
- [27] KOSKO, B., Neural Networks and Fuzzy Systems, University of Southern California, Prentice Hall, Englewood Cliffs, New Jersey, 1992.
- [28] ROSENBLATT, F. , " The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain ", Psych. Rev., pp.386-408, 1958.
- [29] PINEDA, F.J., Generalization of Backpropagation to Recurrent and Higher Order Neural Networks, In Neural Information Processing Systems, Anderson Ed. New York : American Institute of Physics, 1988.
- [30] GÜZELİŞ, C., " Supervised Learning of the Steady-state Outputs in Generalized Cellular Neural Networks ", In Proc. IEEE 2 nd Int. Workshop on Cellular Neural Networks and their Applications, pp.74-79, 1992.

EK
SLAT PROGRAMI

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<dir.h>
#include<alloc.h>
#include<sys\stat.h>
#include<io.h>
#include<string.h>
#include<time.h>

#include<c:\slat\def.inc>

void main_loop();

void stepx();           /* Simulation Tools */
void stepxk();
void stepxk4();
void stepy();
void stepya();
void unstable_x();
void exchange();

void init_template();
int update_template(); /* Learning Tools */
void last_template();
void multiply_template();
void multiply_learningrate();
void decrease_learningrate();
float multiplicator();
void finderror();

void read_configuration(); /* Reading Tools */
void read_trainingset(char trainingset[13]);
void read_template(char temdos[13]);
int fileinputu();
int fileinputx();
```

```

int fileinpute();
void utransfer(int which);
void xtransfer(int which);
void etransfer(int which);

void write_output();           /* Writing Tools */
void write_error();
void auto_tempsave();
void write_template();
void write_setup();

void message();               /* Show Tools (Monitor) */
void show_template();
void clear_status();
void show_status1();
void show_status2();

int c_break(void);
void bound();

void graphinst();             /* Graphical Tools */
void editor();
void menutext(int r);
void cell(char type);
int find_color(float cell_value);
void celltype(char type);
void pixel(int dim);
int stroke();
void cursor_bar(int X1,int Y1,int X2,int Y2,int color);
void window1(int X1,int Y1,int X2,int Y2);
void window1_text(int X1,int Y1,int which,int quantity,char *text[]);
void save_screen(int X1,int Y1,int X2,int Y2,int dim);
void restore_screen(int X1,int Y1,int dim);
void file_menu(int X1,int Y1,int X2,int Y2,char files[30]);
char *filelist(char *head,char *dir);
void print_filenames(char *posit,char *tail,int X1,int Y1);
void load_up(char file[13]);
void save_name(int X1,int Y1,int X2,int Y2);
void save_up();
void algorithm();
void algorithm_text();
void method();
void method_text();
void formula();
void formula_text();
void beta_change();
void learningrate_change();
void cycle_change();
void reset_func();
void step1();

```

```

void step2();
void step3(int X1,int Y1,int X2,int Y2, int entry, char *text[]);
int step4();
void step45();

char *Loadm_text[]={"","Initial Template","Trainingset","Quit"};
char *Savem_text[]={"","Outputimage","Template","Setup","Quit"};
char *Setupm_text[]={"","Algorithm","Method","Formula","Beta","Learningrate",
                    "Cycles","Quit"};
char *Envirm_text[]={"","Report","Path","Extensions","Printer","Quit"};
char *Exit_text[]={"","Reset","Quit"};

char *inputfiles="c:\\slat\\images\\*.";
char *tsefiles="c:\\slat\\tse\\*.tse";
char *temfiles="c:\\slat\\template\\*.tem";

char save_file_name[30];
char output_file_name[30];
char message_text1[35],message_text2[35];

int POS1=1,POS2=1,POS3=1,RUN=0;
char buffer[100];
void *imagebuffer[4];

int dim,UNI,BOS;
int CELLX,CELLY;
int MAXX,MAXY;
int KEY=0;

float time_step=0.4,MAXDIF=0.01,kenar=0.0;
float Treshold=1.5;
float learningrate;
float beta;
int TEM_OK=0,TSE_OK=0;
int N=0,M=0,steady_state;
float minX;
float K,First_K=1;
int image_no;
int Error[100],OLD_Error[100],Totalerror,OLD_Totalerror,Minerror=1000;
int Error_counter=0;
char algorithm_type[20],method_type[20],x_type[20],y_type[20];
float U[MAXDIM][MAXDIM];
float Xe[MAXDIM][MAXDIM],Xy[MAXDIM][MAXDIM];
float Y[MAXDIM][MAXDIM],E[MAXDIM][MAXDIM];
float Udepo[MAXDIM][MAXDIM][TSSIZE];
float Xdepo[MAXDIM][MAXDIM][TSSIZE];
float Edepo[MAXDIM][MAXDIM][TSSIZE];
float a[6],b[6],current,t_a[6],t_b[6],t_current;
int color_a[6],color_b[6],color_current;
char temdos[25],trainingset[25];

```

```

struct str { char ixdosya[25],iudosya[25],edosya[25];} training[100];
int ts_entry_quantity;
int cycle;

```

```

time_t first, second;

```

```

void main()
{
    int Ecl;
    clrscr();

    read_configuration();
    bound();

    graphinst();
    setbkcolor(LIGHTGRAY);
    editor();

```

```

while(1)
{

```

```

    TOP:
    RUN=0;
    menutext(0); /* RUN appears */
    step1();
    menutext(1); /* STOP appears */
    clear_status();
    show_template();

```

```

    first = time(NULL);

```

```

    fileinputu();
    fileinputx();
    fileinpute();

```

```

    if(method_type[0]=='S')
    {
        randomize();
        for (K=First_K;K<=cycle;K++)
        {
            if(kbhit()) { if( stroke()==13 )
                { First_K=K;
                  RUN=0;
                  menutext(0); /* RUN appears */
                  step1();
                  menutext(1); /* STOP appears */
                }
            }
        }
        main_loop();
    }

```

```

init_template();
if(update_template()) {
    init_template();
    decrease_learningrate();
    continue;
}

last_template();
show_template();
}
}
else
{
    for (K=First_K;K<=cycle;K++)
    {
        if ( (OLD_Totalerror==Totalerror)&&(Totalerror) )
        { Error_counter=Error_counter+1 ;      }
        else Error_counter=0;

        if(Totalerror<=Minerror) { Minerror=Totalerror; auto_tempsave();}
        OLD_Totalerror=Totalerror;
        Totalerror=0;

        if( algorithm_type[0]=='B') Ecl=2;
        else Ecl=29;

        if( Error_counter==Ecl )
        {
            multiply_learningrate();
            Error_counter=0;
        }
        init_template();

        for (image_no=1,image_no<=ts_entry_quantity;image_no++)
        {
            if(kbhit()) { if( stroke()==13 )
                {
                    First_K=K;
                    RUN=0;
                    menutext(0); /* RUN appears */
                    step1();
                    menutext(1); /* STOP appears */
                }
            }
        }
        main_loop();
        if(update_template()) {
            init_template();
            image_no=0;
            decrease_learningrate();
            Totalerror=0;
        }
    }
}

```

```

    }
    last_template();
    show_template();
    write_error();

    if(Totalerror==0)
    { if( (algorithm_type[0]=='B')&& ( beta<=20) )
      {
        beta=beta+1;
        strcpy(message_text1,"  Beta increased ");
        sprintf(message_text2,"    %d",(int)beta);
        message();
      }
      else
      {
        if( time_step > 0.01 )
        {
          time_step=0.01;
          strcpy(message_text1,"  Test Mode ");
          strcpy(message_text2,"  Time-step = 0.01  ");
          message();
        }
        else
        {
          second = time(NULL);
          strcpy(message_text1,"  Simulation time ");
          sprintf(message_text2,"    %d seconds", (int)diffime(second,first));
          message();
          goto TOP;
        }
      }
    }
  }
}

}

}

}

/*----- FUNCTIONS -----*/

```

```

void main_loop()
{
    if(method_type[0]=='S') image_no=random(ts_entry_quantity)+1;

    utransfer(image_no);
    xtransfer(image_no);
    etransfer(image_no);
}

```

```

dim=N; if(N<M) dim=M;
pixel(dim);
BOS=((CALT-CUST)/2-20-UNI*dim)/2;

celltype('s'); cell('s');      /* Cell --> Monitor */
celltype('i'); cell('i');
celltype('e'); cell('e');

setcbrk(1);
ctrlbrk(c_break);

show_status1();

switch(x_type[0])
{
case 'e': if( algorithm_type[0]=='B') stepy();
          else stepya();
          do { ctrlbrk(c_break);
              stepx();
              celltype('o'); cell('o');
              exchange();
              if( algorithm_type[0]=='B') stepy();
              else stepya();
          } while ( (steady_state==0)|| (minX<1) );
          break;
case 'h': if( algorithm_type[0]=='B') stepy();
          else stepya();
          do { ctrlbrk(c_break);
              stepxk();
              celltype('o'); cell('o');
              exchange();
              if( algorithm_type[0]=='B') stepy();
              else stepya();
          } while ( (steady_state==0)|| (minX<1) );
          break;
case 'r': if( algorithm_type[0]=='B') stepy();
          else stepya();
          do { ctrlbrk(c_break);
              stepxk4();
              celltype('o'); cell('o');
              exchange();
              if( algorithm_type[0]=='B') stepy();
              else stepya();
          } while ( (steady_state==0)|| (minX<1) );
          break;
}

unstable_x();
celltype('o'); cell('o');

```



```

+ a[2]*(Y[i][j-1]+Y[i][j+1])
+ a[3]*(Y[i-1][j-1]+Y[i+1][j+1])
+ a[4]*(Y[i-1][j+1]+Y[i+1][j-1])
+ b[0]*(U[i][j])
+ b[1]*(U[i-1][j]+U[i+1][j])
+ b[2]*(U[i][j-1]+U[i][j+1])
+ b[3]*(U[i-1][j-1]+U[i+1][j+1])
+ b[4]*(U[i-1][j+1]+U[i+1][j-1])
+ current
);

dif=fabs(Xy[i][j]-Xe[i][j]);
if ( dif>(maxdif/fabs(Xe[i][j]))) maxdif=dif/fabs(Xe[i][j]);
if( (fabs(Xy[i][j])-minX)<0 ) minX=fabs(Xy[i][j]);
    }
}
if (maxdif>MAXDIF) steady_state=0;
}

void stepxk4()
{
    float dif,maxdif=0;
    int i,j;
    steady_state=1;
    minX=1;

    for (i=1;i<=N;i++)
    {
        for (j=1;j<=M;j++)
        {
            Xy[i][j] = (1-time_step+pow(time_step,2)/2-
            pow(time_step,3)/6+pow(time_step,4)/24)*Xe[i][j]+
            (time_step-pow(time_step,2)/2+pow(time_step,3)/6-pow(time_step,4)/24)
            *( a[0]*Y[i][j]
            + a[1]*(Y[i-1][j]+Y[i+1][j])
            + a[2]*(Y[i][j-1]+Y[i][j+1])
            + a[3]*(Y[i-1][j-1]+Y[i+1][j+1])
            + a[4]*(Y[i-1][j+1]+Y[i+1][j-1])
            + b[0]*(U[i][j])
            + b[1]*(U[i-1][j]+U[i+1][j])
            + b[2]*(U[i][j-1]+U[i][j+1])
            + b[3]*(U[i-1][j-1]+U[i+1][j+1])
            + b[4]*(U[i-1][j+1]+U[i+1][j-1])
            + current
            );

            dif=fabs(Xy[i][j]-Xe[i][j]);
            if ( dif>(maxdif/fabs(Xe[i][j])) ) maxdif=dif/fabs(Xe[i][j]);
            if( (fabs(Xy[i][j])-minX)<0 ) minX=fabs(Xy[i][j]);
        }
    }
    if (maxdif>MAXDIF) steady_state=0;
}

```

```

    }

void stepy()
{
    int i,j;

    for (j=1;j<=M;j++)
        { for (i=1;i<=N;i++)
            {
                if (Xe[i][j]>=6) { Y[i][j]=1; } /* Overflow Protection */
                if (Xe[i][j]<=-6) { Y[i][j]=-1; }
                if ((Xe[i][j]>-6)&&(Xe[i][j]<6))
                    { Y[i][j]= 1/(2*beta)*log(cosh(beta*(Xe[i][j]+1)))
                      -1/(2*beta)*log(cosh(beta*(Xe[i][j]-1))) );
                }
            }
        }
}

```

```

void stepya()
{
    int i,j;

    for (j=1;j<=M;j++)
        { for (i=1;i<=N;i++)
            {
                Y[i][j]=0.5*( fabs(Xe[i][j]+1)-fabs(Xe[i][j]-1) );
            }
        }
}

```

```

void unstable_x()
{
    int i,j;
    float unstableX=1.0;

    for (j=1;j<=M;j++)
        { for (i=1;i<=N;i++)
            {
                if( (fabs(Xe[i][j]))<unstableX ) unstableX=fabs(Xe[i][j]);
            }
        }
    if(unstableX<1.0)
        {
            sprintf(buffer," %f", unstableX);
            strcpy(message_text1," Unstable X ");
            strcpy(message_text2,buffer);
            message();
        }
}

```

```

}
```

```

void exchange()
{

```

```

    int i,j;
    for (j=1;j<=M;j++)
        { for (i=1;i<=N;i++) Xe[i][j]=Xy[i][j]; }
}

```

```

void init_template()
{

```

```

    t_a[0]=a[0]; t_a[1]=a[1]; t_a[2]=a[2]; t_a[3]=a[3]; t_a[4]=a[4];
    t_b[0]=b[0]; t_b[1]=b[1]; t_b[2]=b[2]; t_b[3]=b[3]; t_b[4]=b[4];
    t_current=current;
}

```

```

int update_template()
{

```

```

    float y, Norm, m, normrate=1, maxvar=0.1;
    float productor_a[5], productor_b[5], productor_current;
    double ust, dom, nom;
    int i,j;

```

```

    productor_a[0]=0; productor_b[0]=0;
    productor_a[1]=0; productor_b[1]=0;
    productor_a[2]=0; productor_b[2]=0;
    productor_a[3]=0; productor_b[3]=0;
    productor_a[4]=0; productor_b[4]=0;
    productor_current=0;

```

```

    for (j=1;j<=M;j++)
        { for (i=1;i<=N;i++)
            {
                /*rounding error protection */
                y=Y[i][j]; if(y>0.9) y=1.0; if(y<-0.9) y=-1.0;
                nom= (y-E[i][j]);
                if(!nom) continue;

                /*normalization*/
                Norm= sqrt( pow(Y[i][j],2)
                    +pow((Y[i][j]-1) + Y[i][j+1]),2)
                    +pow((Y[i-1][j] + Y[i+1][j]),2)
                    +pow((Y[i-1][j]-1) + Y[i+1][j+1]),2)
                    +pow((Y[i+1][j]-1) + Y[i-1][j+1]),2)
                    +pow(U[i][j],2)
                    +pow((U[i][j]-1) + U[i][j+1]),2)
                    +pow((U[i-1][j] + U[i+1][j]),2)
                    +pow((U[i-1][j]-1) + U[i+1][j+1]),2)
                    +pow((U[i+1][j]-1) + U[i-1][j+1]),2)
                );
            }
        }
}

```

```

        +1
    );
    if(algorithm_type[0]=='B')
    { dom= 0.5*(tanh(beta*(Xy[i][j]+1)) - tanh(beta*(Xy[i][j]-1)) ); }
    else
    { dom=1; }
    ust=nom*dom/Norm;

    productor_a[0]=productor_a[0]+ust*Y[i][j];
    productor_a[1]=productor_a[1]+ust*(Y[i-1][j]+Y[i+1][j]);
    productor_a[2]=productor_a[2]+ust*(Y[i][j-1]+Y[i][j+1]);
    productor_a[3]=productor_a[3]+ust*(Y[i-1][j-1]+Y[i+1][j+1]);
    productor_a[4]=productor_a[4]+ust*(Y[i+1][j-1]+Y[i-1][j+1]);
    productor_b[0]=productor_b[0]+ust*U[i][j];
    productor_b[1]=productor_b[1]+ust*(U[i-1][j]+U[i+1][j]);
    productor_b[2]=productor_b[2]+ust*(U[i][j-1]+U[i][j+1]);
    productor_b[3]=productor_b[3]+ust*(U[i-1][j-1]+U[i+1][j+1]);
    productor_b[4]=productor_b[4]+ust*(U[i+1][j-1]+U[i-1][j+1]);
    productor_current=productor_current+ust;
}

}
m=multiplicator();
t_a[0]=t_a[0]+( productor_a[0]*(-2)*m );
if( t_a[0]<Treshold ) { normrate=Treshold/t_a[0]; t_a[0]=Treshold; }
if( fabs(t_a[0]-a[0]) > maxvar) return(1);

t_a[1]=( t_a[1]+( productor_a[1]*(-2)*m ) )*normrate;
if( fabs(t_a[1]-a[1]) > maxvar) return(1);

t_a[2]=( t_a[2]+( productor_a[2]*(-2)*m ) )*normrate;
if( fabs(t_a[2]-a[2]) > maxvar) return(1);

t_a[3]=( t_a[3]+( productor_a[3]*(-2)*m ) )*normrate;
if( fabs(t_a[3]-a[3]) > maxvar) return(1);

t_a[4]=( t_a[4]+( productor_a[4]*(-2)*m ) )*normrate;
if( fabs(t_a[4]-a[4]) > maxvar) return(1);

t_b[0]=( t_b[0]+( productor_b[0]*(-2)*m ) )*normrate;
if( fabs(t_b[0]-b[0]) > maxvar) return(1);

t_b[1]=( t_b[1]+( productor_b[1]*(-2)*m ) )*normrate;
if( fabs(t_b[1]-b[1]) > maxvar) return(1);

t_b[2]=( t_b[2]+( productor_b[2]*(-2)*m ) )*normrate;
if( fabs(t_b[2]-b[2]) > maxvar) return(1);

t_b[3]=( t_b[3]+( productor_b[3]*(-2)*m ) )*normrate;
if( fabs(t_b[3]-b[3]) > maxvar) return(1);

```

```

    t_b[4]=( t_b[4]+( productor_b[4]*(-2)*m ) ) *normrate;
    if( fabs(t_b[4]-b[4]) > maxvar) return(1);

    t_current=( t_current+( productor_current*(-2)*m ) ) *normrate;
    if( fabs(t_current-current) > maxvar) return(1);

    return(0);
}

void last_template()
{
    int I;
    float dif;

    dif=0;
    for (I=0;I<=4;I++)
        { if( (fabs(t_a[I]-a[I])) > dif ) dif=fabs(t_a[I]-a[I]);}
    for (I=0;I<=4;I++)
        { if( fabs((t_b[I]-b[I])) > dif ) dif=fabs(t_b[I]-b[I]);}
        if( fabs((t_current-current)) > dif ) dif=fabs(t_current-current);

    if(method_type[0]=='S') Totalelerror=0;
    if ( (dif < 0.0001) && (Totalelerror!=0) ) multiply_learningrate();

    color_a[0]=LIGHTGRAY;
    if((t_a[0]-a[0])>0) color_a[0]=GREEN;
    if((t_a[0]-a[0])<0) color_a[0]=RED;
    a[0]=t_a[0];

    color_a[1]=LIGHTGRAY;
    if((t_a[1]-a[1])>0) color_a[1]=GREEN;
    if((t_a[1]-a[1])<0) color_a[1]=RED;
    a[1]=t_a[1];

    color_a[2]=LIGHTGRAY;
    if((t_a[2]-a[2])>0) color_a[2]=GREEN;
    if((t_a[2]-a[2])<0) color_a[2]=RED;
    a[2]=t_a[2];

    color_a[3]=LIGHTGRAY;
    if((t_a[3]-a[3])>0) color_a[3]=GREEN;
    if((t_a[3]-a[3])<0) color_a[3]=RED;
    a[3]=t_a[3];

    color_a[4]=LIGHTGRAY;
    if((t_a[4]-a[4])>0) color_a[4]=GREEN;
    if((t_a[4]-a[4])<0) color_a[4]=RED;
    a[4]=t_a[4];

```

```

    color_b[0]=LIGHTGRAY;
    if((t_b[0]-b[0])>0) color_b[0]=GREEN;
    if((t_b[0]-b[0])<0) color_b[0]=RED;
    b[0]=t_b[0];

    color_b[1]=LIGHTGRAY;
    if((t_b[1]-b[1])>0) color_b[1]=GREEN;
    if((t_b[1]-b[1])<0) color_b[1]=RED;
    b[1]=t_b[1];

    color_b[2]=LIGHTGRAY;
    if((t_b[2]-b[2])>0) color_b[2]=GREEN;
    if((t_b[2]-b[2])<0) color_b[2]=RED;
    b[2]=t_b[2];

    color_b[3]=LIGHTGRAY;
    if((t_b[3]-b[3])>0) color_b[3]=GREEN;
    if((t_b[3]-b[3])<0) color_b[3]=RED;
    b[3]=t_b[3];

    color_b[4]=LIGHTGRAY;
    if((t_b[4]-b[4])>0) color_b[4]=GREEN;
    if((t_b[4]-b[4])<0) color_b[4]=RED;
    b[4]=t_b[4];

    color_current=LIGHTGRAY;
    if((t_current-current)>0) color_current=GREEN;
    if((t_current-current)<0) color_current=RED;
    current=t_current;

}

void multiply_template()
{
    int z=1.2;

    a[0]=a[0]*z; b[0]=b[0]*z;
    a[1]=a[1]*z; b[1]=b[1]*z;
    a[2]=a[2]*z; b[2]=b[2]*z;
    a[3]=a[3]*z; b[3]=b[3]*z;
    a[4]=a[4]*z; b[4]=b[4]*z; current=current*z;

    strcpy(message_text1,"Template multiplication ");
    strcpy(message_text2,"    occurred    ");
    message();

}

void multiply_learningrate()
{

```

```

    strcpy(message_text1," Learningrate ");
    strcpy(message_text2,"multiplication occurred ");
    message();
    learningrate=learningrate*10;
    if (learningrate > 1 ) learningrate=ceil(learningrate);
}

void decrease_learningrate()
{
    strcpy(message_text1," Template variation ");
    strcpy(message_text2," overflow ");
    message();
    learningrate=learningrate*0.1;
}

float multiplicator()
{
    float m;

    m=learningrate;
    if(method_type[0]=='S') m=learningrate*(1-K/cycle);
    return(m);
}

void finderror()
{
    int i,j;
    float y,error;

    error=0;

    for (j=1;j<=M;j++)
        { for (i=1;i<=N;i++)
            {
                if(Y[i][j]>0.9) y=1; if(Y[i][j]<-0.9) y=-1;
                error=error+((y-E[i][j])*(y-E[i][j])/4);
            }
        }
    OLD_Error[image_no]=Error[image_no];
    Error[image_no]=error;
    Totalerror=Totalerror+error;
}

void read_configuration()
{
    FILE *image;
    int i;

    if ((image = fopen("slat.cfg","r")) == NULL)
    {

```

```

        strcpy(message_text1, "    Cannot open ");
        strcpy(message_text2, " configurationfile.");
        message();
    }

    fscanf(image, "%s %s %s", algorithm_type, method_type, x_type);
    fscanf(image, "%f %f %d", &beta, &learningrate, &cycle);

    fclose(image);
}

void read_trainingset(char trainingset[13])
{
    FILE *image;
    int i;

    strcpy(buffer, "c:\\slat\\tse\\");
    strcat(buffer, trainingset);
    if ((image = fopen(buffer, "r")) == NULL)
    {
        strcpy(message_text1, "Cannot open trainingsetfile.");
        strcpy(message_text2, "");
        message();
    }

    fscanf(image, "%d", &ts_entry_quantity);

    for(i=1; i<=ts_entry_quantity; i++)
    {
        fscanf(image, "%s", &training[i].ixdosya);
        fscanf(image, "%s", &training[i].iudosya);
        fscanf(image, "%s", &training[i].edosya);
    }
    fclose(image);
}

void read_template(char temdos[13])
{
    FILE *image;
    char word[20];
    float bos;

    strcpy(buffer, "c:\\slat\\template\\");
    strcat(buffer, temdos);
    if ((image = fopen(buffer, "r")) == NULL)

    if ((image = fopen(temdos, "r")) == NULL)
    {
        strcpy(message_text1, "Cannot open templatefile.");
        strcpy(message_text2, "");
    }
}

```



```

        message();
    }

    fscanf(image,"%s",word);    fscanf(image,"%f",&bos);    /* a[3] a[2] a[4] */
    fscanf(image,"%s",word);    fscanf(image,"%f",&a[3]);    /* a[1] a[0] a[1] */
    fscanf(image,"%f",&a[2]);    fscanf(image,"%f",&a[4]);    /* a[4] a[2] a[3] */
    fscanf(image,"%f",&a[1]);    fscanf(image,"%f",&a[0]);    /* a1 a2 a3 */
    fscanf(image,"%f",&bos);    fscanf(image,"%f",&bos);    /* a4 a5 a4 */
    fscanf(image,"%f",&bos);    fscanf(image,"%f",&bos);    /* a3 a2 a1 */
    fscanf(image,"%s",word);    fscanf(image,"%f",&current);
    fscanf(image,"%s",word);    fscanf(image,"%f",&b[3]);
    fscanf(image,"%f",&b[2]);    fscanf(image,"%f",&b[4]);
    fscanf(image,"%f",&b[1]);    fscanf(image,"%f",&b[0]);

    fclose(image);
}

int fileinputu()
{
    FILE *image;
    int l,i,j;
    int n,m;

    for (l=1;l<=ts_entry_quantity;l++)
    {
        strcpy(buffer,"c:\\slat\\images\\");
        strcat(buffer,training[l].iudosya);
        if ((image = fopen(buffer,"r")) == NULL)
        {
            strcpy(message_text1,"Cannot open inputfile.");
            strcpy(message_text2,"");
            message();
            return(0);
        }

        if ( N==0 && M==0 )
        {
            fscanf(image,"%d %d",&N,&M);
        }
        else
        {
            fscanf(image,"%d %d",&n,&m);
            if ( n!=N || m!=M )
            {
                strcpy(message_text1," Input and state image");
                strcpy(message_text2," dimensions don't match");
                message();
                return(0);
            }
        }
    }
}

```

```

        for ( j=1 ; j<=M ; j++ )
        {
            for ( i=1 ; i<=N ; i++ )
            {
                fscanf(image, "%f", &Udepo[i][j][1]);
            }
        }
        fclose(image);
    }
    return(1);
}

int fileinputx()
{
    FILE *image;
    int l,i,j;
    int n,m,X;

    for (l=1;l<=ts_entry_quantity;l++)
    {
        strcpy(buffer, "c:\\slat\\images\\");
        strcat(buffer, training[l].ixdosya);
        if ((image = fopen(buffer, "r")) == NULL)
        {
            strcpy(message_text1, "Cannot open statefile.");
            strcpy(message_text2, "");
            message();
            return(0);
        }

        if ( N==0 && M==0 )
        {
            fscanf(image, "%d %d", &N, &M);
        }
        else
        {
            fscanf(image, "%d %d", &n, &m);
            if ( n!=N || m!=M )
            {
                strcpy(message_text1, " Input and state image ");
                strcpy(message_text2, "dimensions don't match !");
                message();
                exit(0);
            }
        }

        for ( j=1 ; j<=M ; j++ )
        {
            for ( i=1 ; i<=N ; i++ )
            {

```

```

        fscanf(image,"%f",&Xdepo[i][j][l]);
    }
}
fclose(image);
}
return(1);
}

```

```

int fileinpute()
{
    FILE *image;
    int l,i,j;
    int n,m;

    for (l=1;l<=ts_entry_quantity;l++)
    {
        strcpy(buffer,"c:\\slat\\images\\");
        strcat(buffer,training[l].edosya);
        if ((image = fopen(buffer,"r")) == NULL)
        {
            strcpy(message_text1,"Cannot open expectedfile.");
            strcpy(message_text2,"");
            message();
            return(0);
        }
        if ( N==0 && M==0 )
        {
            fscanf(image,"%d %d",&N,&M);
        }
        else
        {
            fscanf(image,"%d %d",&n,&m);
            if ( n!=N || m!=M )
            {
                strcpy(message_text1,"Image dimensions don't");
                strcpy(message_text2,"      match");
                message();
                exit(0);
            }
        }
        for ( j=1 ; j<=M ; j++ )
        {
            for ( i=1 ; i<=N ; i++ )
            {
                fscanf(image,"%f",&Edepo[i][j][l]);
            }
        }
        fclose(image);
    }
}

```

```

    return(1);
}

void utransfer(int which)
{
    int i,j;

    for(j=1;j<MAXDIM;j++)
    {
        for(i=1;i<MAXDIM;i++)
        {
            U[i][j]=Udepo[i][j][which];
        }
    }
}

void xtransfer(int which)
{
    int i,j;

    for(j=1;j<MAXDIM;j++)
    {
        for(i=1;i<MAXDIM;i++)
        {
            Xe[i][j]=Xdepo[i][j][which];
        }
    }
}

void etransfer(int which)
{
    int i,j;

    for(j=1;j<MAXDIM;j++)
    {
        for(i=1;i<MAXDIM;i++)
        {
            E[i][j]=Edepo[i][j][which];
        }
    }
}

void write_output()
{
    FILE *image;
    int i,j;

    strcpy(buffer,"c:\\slat\\images\\");
    strcat(buffer,save_file_name);
    if ((image = fopen(buffer,"w")) == NULL)
    {

```

```

        strcpy(message_text1,"Cannot open outputfile.");
        strcpy(message_text2,"");
        message();
    }
    fprintf(image,"%d \n %d \n",N,M);
    for ( j=1 ; j<=M ; j++ )
    {
        for ( i=1 ; i<=N ; i++ )
        {
            if ( Y[i][j]<0 )
            {
                fprintf(image,"%3.2f ",Y[i][j]);
            }
            else
            {
                fprintf(image," %3.2f ",Y[i][j]);
            }
        }
        fprintf(image,"\n");
    }
    fclose(image);
}

void write_error()
{
    FILE *image;
    int i,j;

    strcpy(buffer,"c:\\slat\\error");
    if (K==1) (image = fopen(buffer,"w"));
    else      (image = fopen(buffer,"a"));

    fprintf(image,"\n %3.0f. cycle %d  %f",K,Totalelerror,learningrate);

    fclose(image);
}

void auto_tempsave()
{
    strcpy(save_file_name,"min.tem");
    write_template();
}

void write_template()
{
    FILE *image;

    strcpy(buffer,"c:\\slat\\template\\");
    strcat(buffer,save_file_name);

```

```

if ((image = fopen(buffer, "w")) == NULL)
{
    strcpy(message_text1, "Cannot open save_file.");
    strcpy(message_text2, "");
    message();
}

fprintf(image, "Neighborhood: 1\n\n");

fprintf(image, "Feedback: \n");
fprintf(image, "%8.6f %8.6f %8.6f\n", a[3], a[2], a[4]);
fprintf(image, "%8.6f %8.6f %8.6f\n", a[1], a[0], a[1]);
fprintf(image, "%8.6f %8.6f %8.6f\n", a[4], a[2], a[3]);

fprintf(image, "Current: %8.6f\n\n", current);

fprintf(image, "Control: \n");
fprintf(image, "%8.6f %8.6f %8.6f\n", b[3], b[2], b[4]);
fprintf(image, "%8.6f %8.6f %8.6f\n", b[1], b[0], b[1]);
fprintf(image, "%8.6f %8.6f %8.6f\n", b[4], b[2], b[3]);

second = time(NULL);
fprintf(image, " \n\n Simulation time: %d seconds", (int)difftime(second, first));

fclose(image);
}

void write_setup()
{
    FILE *image;

    if ((image = fopen(save_file_name, "w")) == NULL)
    {
        strcpy(message_text1, "Cannot open save_file.");
        strcpy(message_text2, "");
        message();
    }

    switch(algorithm_type[0])
    {
        case 'B': fprintf(image, "\nB-RBLA\n"); break;
        case 'P': fprintf(image, "\nP-RPLA\n"); break;
    }

    switch(method_type[0])
    {
        case 'D': fprintf(image, "\nDeterministic\n"); break;
        case 'S': fprintf(image, "\nStochastic\n"); break;
    }
}

```

```

switch(x_type[0])
{
    case 'e': fprintf(image, "\neuler\n");    break;
    case 'r': fprintf(image, "\nrunge_kutta\n"); break;
}

fprintf(image, "%4.2f\n%7.2f\n%d\n\n", beta, learningrate, cycle);

fprintf(image, "#1 (P-RPLA) or (B-RBLA)\n");
fprintf(image, "#2 (Deterministic) or (Stochastic)\n");
fprintf(image, "#3 (Euler) or (Runge-Kutta)\n");
fprintf(image, "#4 beta\n");
fprintf(image, "#5 learningrate\n");
fprintf(image, "#6 cycle\n");

fclose(image);

}

void message()
{
    save_screen(190,200,450,280,3);
    window1(190,200,450,280);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
    outtextxy(245,215,"ATTENTION");
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
    outtextxy(230,240,message_text1);
    outtextxy(230,258,message_text2);
    delay(2000);
    restore_screen(190,200,3);
}

void show_template()
{
    settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
    setcolor(WHITE);
    setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+ 31,TSAG-41,TUST+ 59);
    sprintf(buffer, " %f", a[3]); if(a[3]<0) sprintf(buffer,"%f", a[3]);
    outtextxy(TSOL+90,TUST+ 40, buffer);
    setfillstyle( SOLID_FILL,color_a[3]); bar(TSAG-35,TUST+35,TSAG-5,TUST+55);
    setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+ 61,TSAG-41,TUST+ 89);
    sprintf(buffer, " %f", a[2]); if(a[2]<0) sprintf(buffer,"%f", a[2]);
    outtextxy(TSOL+90,TUST+ 70, buffer);
    setfillstyle( SOLID_FILL,color_a[2]); bar(TSAG-35,TUST+65,TSAG-5,TUST+85);
    setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+ 91,TSAG-41,TUST+119);
    sprintf(buffer, " %f", a[4]); if(a[4]<0) sprintf(buffer,"%f", a[4]);
    outtextxy(TSOL+90,TUST+100, buffer);
    setfillstyle( SOLID_FILL,color_a[4]); bar(TSAG-35,TUST+95,TSAG-5,TUST+115);
    setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+121,TSAG-41,TUST+149);
    sprintf(buffer, " %f", a[1]); if(a[1]<0) sprintf(buffer,"%f", a[1]);

```

```

outtextxy(TSOL+90,TUST+130,buffer);
setfillstyle( SOLID_FILL,color_a[1]); bar(TSAG-35,TUST+125,TSAG,TUST+145);
setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+151,TSAG-41,TUST+179);
sprintf(buffer," %f", a[0]); if(a[0]<0) sprintf(buffer,"%f", a[0]);
outtextxy(TSOL+90,TUST+160,buffer);
setfillstyle( SOLID_FILL,color_a[0]); bar(TSAG-35,TUST+155,TSAG,TUST+175);
setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+181,TSAG-41,TUST+209);
sprintf(buffer," %f", b[3]); if(b[3]<0) sprintf(buffer,"%f", b[3]);
outtextxy(TSOL+90,TUST+190,buffer);
setfillstyle( SOLID_FILL,color_b[3]); bar(TSAG-35,TUST+185,TSAG,TUST+205);
setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+211,TSAG-41,TUST+239);
sprintf(buffer," %f", b[2]); if(b[2]<0) sprintf(buffer,"%f", b[2]);
outtextxy(TSOL+90,TUST+220,buffer);
setfillstyle( SOLID_FILL,color_b[2]); bar(TSAG-35,TUST+215,TSAG,TUST+235);
setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+241,TSAG-41,TUST+269);
sprintf(buffer," %f", b[4]); if(b[4]<0) sprintf(buffer,"%f", b[4]);
outtextxy(TSOL+90,TUST+250,buffer);
setfillstyle( SOLID_FILL,color_b[4]); bar(TSAG-35,TUST+245,TSAG,TUST+265);
setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+271,TSAG-41,TUST+299);
sprintf(buffer," %f", b[1]); if(b[1]<0) sprintf(buffer,"%f", b[1]);
outtextxy(TSOL+90,TUST+280,buffer);
setfillstyle( SOLID_FILL,color_b[1]); bar(TSAG-35,TUST+275,TSAG,TUST+295);
setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+301,TSAG-41,TUST+329);
sprintf(buffer," %f", b[0]); if(b[0]<0) sprintf(buffer,"%f", b[0]);
outtextxy(TSOL+90,TUST+310,buffer);
setfillstyle( SOLID_FILL,color_b[0]); bar(TSAG-35,TUST+305,TSAG,TUST+325);
setfillstyle( SOLID_FILL,CYAN); bar(TSOL+81,TUST+331,TSAG-41,TUST+359);
sprintf(buffer," %f",current);if(current<0) sprintf(buffer,"%f",current);
outtextxy(TSOL+90,TUST+340,buffer);
setfillstyle( SOLID_FILL,color_current); bar(TSAG,TUST+335,TSAG,TUST+355);
}

```

```

void clear_status()
{
setfillstyle(SOLID_FILL,LIGHTGRAY); bar(SSOL+3,SUST+3,SSAG-3,SALT-3);
}

```

```

void show_status1()
{
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
setcolor(WHITE);
setfillstyle(SOLID_FILL,LIGHTGRAY);
bar(SSAG-260,SUST+3,SSAG-3,SALT-3);

outtextxy(SSAG-250,SUST+10,"Algorithm  =");
outtextxy(SSAG-250,SUST+24,"Method    =");
outtextxy(SSAG-250,SUST+38,"Learningrate=");
outtextxy(SSAG-250,SUST+52,"Cycle/Image =");

bar(SSAG-140,SUST+3,SSAG-5,SALT-3);
}

```



```

    outtextxy(SSAG-130,SUST+10, algorithm_type+2);
    outtextxy(SSAG-130,SUST+24, method_type);
    if(learningrate>10) sprintf(buffer, "%e",learningrate);
    else      sprintf(buffer, "%6.5f",learningrate);
    outtextxy(SSAG-130,SUST+38, buffer);
    sprintf(buffer, "%d / %d", (int)K, (image_no));
    outtextxy(SSAG-130,SUST+52, buffer);
}

void show_status2()
{
    switch(method_type[0])
    {
        case 'S': {
            settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
            setcolor(WHITE);
            setfillstyle(SOLID_FILL,LIGHTGRAY);
            bar(SSOL+3,SUST+3,SSAG-260,SALT-3);

            outtextxy(SSOL+20,SUST+12,"Last Error=");
            outtextxy(SSOL+20,SUST+44,"Error=");
            sprintf(buffer, "%d", OLD_Error[image_no]);
            outtextxy(SSOL+220,SUST+12, buffer);
            sprintf(buffer, "%d", Error[image_no]);
            outtextxy(SSOL+220,SUST+44, buffer);
            break;
        }
        case 'D': {
            settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
            setcolor(WHITE);
            setfillstyle(SOLID_FILL,LIGHTGRAY);
            bar(SSOL+3,SUST+3,SSAG-260,SUST+40);

            outtextxy(SSOL+15,SUST+10,"Last Total =");
            outtextxy(SSOL+15,SUST+24,"Totalerror =");
            outtextxy(SSOL+195,SUST+10,"Last Error =");
            outtextxy(SSOL+195,SUST+24,"Error   =");

            bar(SSOL+115,SUST+3,SSOL+170,SUST+40);
            sprintf(buffer, "%d", OLD_Totalerror);
            outtextxy(SSOL+130,SUST+10, buffer);
            sprintf(buffer, "%d", Totalerror);
            outtextxy(SSOL+130,SUST+24, buffer);

            bar(SSOL+315,SUST+3,SSOL+370,SUST+40);
            sprintf(buffer, "%d", OLD_Error[image_no]);
            outtextxy(SSOL+310,SUST+10, buffer);
            sprintf(buffer, "%d", Error[image_no]);
            outtextxy(SSOL+310,SUST+24, buffer);
        }
    }
}

```

```

        delay(1000);

        if((image_no==ts_entry_quantity)&&(ts_entry_quantity))
        {
            settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
            bar(SSOL+3,SUST+40,SSAG-260,SALT-3);
            if(Totalerror<OLD_Totalerror) setcolor(RED);
            else setcolor(WHITE);
            outtextxy(SSOL+15,SUST+46,"Total =");
            sprintf(buffer, "%d (%d)",Totalerror,OLD_Totalerror);
            outtextxy(SSOL+145,SUST+46, buffer);
        }
        break;
    } }

}

int c_break(void)
{
    strcpy(message_text1," Control-Break hit. ");
    strcpy(message_text2," Program aborting ...");
    message();
    return (0);
}

void bound()
{
    int i,j;
    for (i=0;i<=(MAXDIM-1);i++)
        {
            for (j=0;j<=(MAXDIM-1);j++)
                {
                    U[i][j]=0;   Xe[i][j]=kenar;   Xy[i][j]=kenar;
                    E[i][j]=0;   Y[i][j]=kenar;  }
        }
}

/*=====GRAPHICS=====*/

void graphinst()
{
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        strcpy(message_text1,"Graphics error ! ");
        strcpy(message_text2,grapherrormsg(errorcode));
        message();
        getch();
        exit(1); /* terminate with an error code */
    }
}

```

```

    }
}

void editor()
{
    int i;

    /*----- Menu -----*/

    setcolor(BLUE);
    rectangle(MSOL,MUST,MSAG,MALT);
    rectangle(MSOL+1,MUST+1,MSAG-1,MALT-1);
    line(MSOL,MALT-41,MSAG,MALT-41);

    cursor_bar(MSOL+1,MALT-40,MSAG-1,MALT-1,CYAN);    /* Menu button
color */

    for(i=0;i<=5;i++)    /* Button shadows */
    {
        setcolor(WHITE);
        line(MSOL+2+105*i,MALT-39,MSOL+104+105*i,MALT-39);
        line(MSOL+3+105*i,MALT-38,MSOL+103+105*i,MALT-38);
        line(MSOL+4+105*i,MALT-37,MSOL+102+105*i,MALT-37);
        line(MSOL+2+105*i,MALT-39,MSOL+2+105*i,MALT-2);
        line(MSOL+3+105*i,MALT-38,MSOL+3+105*i,MALT-3);
        line(MSOL+4+105*i,MALT-37,MSOL+4+105*i,MALT-4);

        setcolor(DARKGRAY);
        line(MSOL+5+105*i,MALT-36,MSOL+101+105*i,MALT-36);
        line(MSOL+5+105*i,MALT-36,MSOL+5+105*i,MALT-5);
        line(MSOL+6+105*i,MALT-35,MSOL+2+105*i,MALT-39);

        line(MSOL+2+105*i,MALT-1,MSOL+104+105*i,MALT-1);
        line(MSOL+2+105*i,MALT-2,MSOL+104+105*i,MALT-2);
        line(MSOL+3+105*i,MALT-3,MSOL+103+105*i,MALT-3);
        line(MSOL+4+105*i,MALT-4,MSOL+102+105*i,MALT-4);
        line(MSOL+105+105*i,MALT-40,MSOL+105+105*i,MALT-2);
        line(MSOL+104+105*i,MALT-39,MSOL+104+105*i,MALT-2);
        line(MSOL+103+105*i,MALT-37,MSOL+103+105*i,MALT-3);
        line(MSOL+102+105*i,MALT-36,MSOL+102+105*i,MALT-4);
    }

    setcolor(BLUE);    /*----- Template -----*/
    rectangle(TSOL,TUST,TSAG,TALT);
    rectangle(TSOL+1,TUST+1,TSAG-1,TALT-1);
    for (i=1;i<13;i++) { line(TSOL,TUST+30*i,TSAG,TUST+30*i); }
    line(TSOL+80,TUST,TSOL+80,TALT);
    line(TSAG-40,TUST,TSAG-40,TALT);

    setfillstyle(SOLID_FILL, GREEN);
    bar(TSAG-38,TUST+2,TSAG-2,TUST+15);

```

```
setfillstyle(SOLID_FILL,RED);
bar(TSAG-39,TUST+16,TSAG-1,TUST+29);
```

```
setcolor(YELLOW);
outtextxy(TSOL+ 7,TUST+ 15,"Template");
outtextxy(TSOL+150,TUST+ 15,"Value");
outtextxy(TSAG-33,TUST+ 6,"Inc");
outtextxy(TSAG-33,TUST+ 18,"Dec");
outtextxy(TSOL+20,TUST+ 45,"a1 =");
outtextxy(TSOL+20,TUST+ 75,"a2 =");
outtextxy(TSOL+20,TUST+105,"a3 =");
outtextxy(TSOL+20,TUST+135,"a4 =");
outtextxy(TSOL+20,TUST+165,"a5 =");
outtextxy(TSOL+20,TUST+195,"b1 =");
outtextxy(TSOL+20,TUST+225,"b2 =");
outtextxy(TSOL+20,TUST+255,"b3 =");
outtextxy(TSOL+20,TUST+285,"b4 =");
outtextxy(TSOL+20,TUST+315,"b5 =");
outtextxy(TSOL+20,TUST+345," I =");
```

```
setcolor(BLUE);          /*----- Cell -----*/
rectangle(CSOL,CUST,CSAG,CALT);
rectangle(CSOL+1,CUST+1,CSAG-1,CALT-1);
line((CSAG+CSOL)/2,CUST,(CSAG+CSOL)/2,CALT);
line(CSOL,(CALT+CUST)/2,CSAG,(CALT+CUST)/2);
line(CSOL,(CALT+CUST)/2+1,CSAG,(CALT+CUST)/2+1);
line(CSOL,(CUST+20),CSAG,(CUST+20));
line(CSOL,((CALT+CUST)/2+20),CSAG,((CALT+CUST)/2+20));
```

```
setcolor(YELLOW);
outtextxy(CSOL+65,CUST+8,"INPUT");
outtextxy((CSAG+CSOL)/2+30,CUST+8,"INITIAL STATE");
outtextxy(CSOL+30,(CUST+CALT)/2+8,"DESIRED OUTPUT");
outtextxy((CSAG+CSOL)/2+30,(CALT+CUST)/2+8,"ACTUAL OUTPUT");
```

```
setcolor(BLUE);          /*----- Status -----*/
rectangle(SSOL,SUST,SSAG,SALT);
rectangle(SSOL+2,SUST+2,SSAG-2,SALT-2);
```

```
setcolor(YELLOW);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
outtextxy(SSOL+70,SUST+20," Supervised Learning Algorithm Tool");
settextstyle(SMALL_FONT,HORIZ_DIR,4);
outtextxy(SSOL+455,SUST+53,"Designed by Sinan Karamahmut ");
```

```
cursor_bar(Bar1_X1,Bar1_Y1,Bar1_X2,Bar1_Y2,BLUE);
menutext(0);
}
```

```

void menutext(int r)
{
    settextstyle(0,HORIZ_DIR,2);

    setcolor(YELLOW);
    outtextxy(MSOL+22,MALT-27,"LOAD");
    outtextxy(MSOL+127,MALT-27,"SAVE");
    if(r==0) {
        if(POS1==3) {
            cursor_bar(Bar1_X1+210,Bar1_Y1,Bar1_X2+210,Bar1_Y2,BLUE); }
        else {
            cursor_bar(Bar1_X1+210,Bar1_Y1,Bar1_X2+210,Bar1_Y2,CYAN); }

            outtextxy(MSOL+238,MALT-27,"RUN");
        }
    else
        { cursor_bar(Bar1_X1+210,Bar1_Y1,Bar1_X2+210,Bar1_Y2,RED);
          outtextxy(MSOL+232,MALT-27,"STOP");
        }
    outtextxy(MSOL+331,MALT-27,"SETUP");
    outtextxy(MSOL+431,MALT-27,"ENVIR");
    outtextxy(MSOL+546,MALT-27,"EXIT");

    setcolor(WHITE);
}

void cell(char type)
{
    int i,j;
    int COLOR;
    for(j=1;j<=M;j++)
    {
        for(i=1;i<=N;i++)
        {
            setfillstyle(SOLID_FILL,8);
            switch(type)
            {
                case 'i' : if ( U[i][j]<0 ) setfillstyle(SOLID_FILL,15); break;
                case 's' : if ( Xe[i][j]<0 ) setfillstyle(SOLID_FILL,15); break;
                case 'e' : COLOR=find_color(E[i][j]);
                          setfillstyle(SOLID_FILL,COLOR); break;
                case 'o' : COLOR=find_color(Y[i][j]);
                          setfillstyle(SOLID_FILL,COLOR); break;
            }
            bar(CELLX+(i-1)*UNI,CELLY+(j-1)*UNI,CELLX+(i)*UNI,CELLY+(j)*UNI);
        }
    }
}

int find_color(float cell_value)

```

```

{
    int COLOR, which;

    if(cell_value <= 0) which = floor(5 * cell_value);
    else which = ceil(5 * cell_value);
    switch (which)
    {
        case 6: COLOR = DARKGRAY; break;
        case 5: COLOR = DARKGRAY; break;
        case 4: COLOR = LIGHTGRAY; break;
        case 3: COLOR = BROWN; break;
        case 2: COLOR = BLUE; break;
        case 1: COLOR = RED; break;
        case 0: COLOR = MAGENTA; break;
        case -1: COLOR = LIGHTRED; break;
        case -2: COLOR = LIGHTBLUE; break;
        case -3: COLOR = LIGHTMAGENTA; break;
        case -4: COLOR = YELLOW; break;
        case -5: COLOR = WHITE; break;
        case -6: COLOR = WHITE; break;
    }
    return(COLOR);
}

void celltype(char type)
{
    switch(type)
    {
        case 'i' : CELLX = CSOL + BOS;          CELLY = CUST + 20 + BOS; break;
        case 's' : CELLX = CSOL + BOS + (CSAG - CSOL) / 2; CELLY = CUST + 20 + BOS; break;
        case 'e' : CELLX = CSOL + BOS;          CELLY = (CALT + CUST) / 2 + 20 + BOS; break;
        case 'o' : CELLX = CSOL + BOS + (CSAG - CSOL) / 2; CELLY = (CALT + CUST) / 2 + 20 + BOS; break;
    }
}

void pixel(int dim)
{
    switch (dim)
    {
        case 1 : break;
        case 2 : UNI = 70; break; case 3 : UNI = 50; break;
        case 4 : UNI = 35; break; case 5 : UNI = 30; break;
        case 6 : UNI = 25; break; case 7 : UNI = 22; break;
        case 8 : UNI = 18; break; case 9 : UNI = 16; break;
        case 10 : UNI = 15; break; case 11 : UNI = 14; break;
        case 12 : UNI = 12; break; case 13 : UNI = 11; break;
        case 14 : UNI = 11; break; case 15 : UNI = 10; break;
    }
}

```

```

        case 16: UNI= 9; break; case 17: UNI= 9; break;
        case 18: UNI= 8; break; case 19: UNI= 8; break;
        case 20: UNI= 7; break; case 21: UNI= 7; break;
        case 22: UNI= 7; break; case 23: UNI= 6; break;
        case 24: UNI= 6; break; case 25: UNI= 6; break;
        case 26: UNI= 6; break; case 27: UNI= 5; break;
        case 28: UNI= 5; break; case 29: UNI= 5; break;
        case 30: UNI= 5; break;
    }
}

int stroke()
{
    int contin=1, keyb_code, code=0;

    while(contin)
    {
        if( (keyb_code=getch())!=0 )
        {
            switch(keyb_code)
            {
                case 13: code=13; contin=0; break; /*ENTER*/
                case 27: code=27; contin=0; break; /*ESC*/
                case 112: code=112; contin=0; break; /*P*/
                default: contin=0; break; /*nothing*/
            }
        }
        else
        {
            switch(getch())
            {
                case 72: code=1; contin=0; break; /*UP*/
                case 75: code=2; contin=0; break; /*LEFT*/
                case 77: code=3; contin=0; break; /*RIGHT*/
                case 80: code=4; contin=0; break; /*DOWN*/
                case 73: code=5; contin=0; break; /*PgUp*/
                case 81: code=6; contin=0; break; /*PgDown*/
                case 71: code=7; contin=0; break; /*Home*/
                case 79: code=8; contin=0; break; /*End*/
            }
        }
    }
    return(code);
}

void cursor_bar(int X1,int Y1,int X2,int Y2,int color)
{
    setfillstyle(SOLID_FILL,color);
    bar(X1,Y1,X2,Y2);
}

```

```

void window1(int X1,int Y1,int X2,int Y2)
{
    setfillstyle(SOLID_FILL,BLUE);
    bar(X1,Y1,X2,Y2);
    setcolor(YELLOW);
    rectangle( X1+3,Y1+3, X2-3,Y2-3);
    rectangle( X1+5,Y1+5, X2-5,Y2-5);
}

void window1_text(int X1,int Y1,int which,int quantity,char *text[])
{
    int i;
    setcolor(WHITE);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
    if(which!=0) { outtextxy(X1+10,Y1-15+(which*30),text[which]); }
    else { for(i=1;i<=quantity;i++)
            outtextxy(X1+10,Y1-15+(i*30),text[i]);
    }
}

void save_screen(int X1,int Y1,int X2,int Y2,int dim)
{
    unsigned size;

    size = imagesize(X1,Y1,X2,Y2); /* get byte size of image */

    if (( imagebuffer[dim] = malloc(size)) == NULL)
    {
        closegraph();
        strcpy(message_text1,"Not enough heap space");
        strcpy(message_text2," in save_screen().");
        message();
        exit(1);
    }
    getimage(X1,Y1,X2,Y2, imagebuffer[dim]);
}

void restore_screen(int X1,int Y1,int dim)
{
    putimage(X1,Y1, imagebuffer[dim], COPY_PUT);
    free(imagebuffer[dim]);
    imagebuffer[dim]=0;
}

void file_menu(int X1,int Y1,int X2,int Y2,char files[30])
{
    char *head,*tail,*posit;
    int i,contin=0;
    char which_file[13];

```



```

head=( char * ) malloc(2600);
tail=filelist(head,files);
posit=head;

```

```

save_screen(X1,Y1,X2+Shadow,Y2+Shadow,2);
cursor_bar(X1+Shadow,Y1+Shadow,X2+Shadow,Y2+Shadow,DARKGRAY);
window1(X1,Y1,X2,Y2);
cursor_bar(X1+6,Y1+6,X2-6,Y1+26,DARKGRAY);
print_filenames(posit,tail,X1,Y1);

```

```

while(contin==0)

```

```

{
    switch(stroke())

```

```

    {
        case 1:if(POS3!=1)                /*UP*/

```

```

        {
            cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),BLUE);

```

```

            POS3--;

```

```

            cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);
            print_filenames(posit,tail,X1,Y1);

```

```

        }

```

```

    else

```

```

    {
        if(posit!=head)

```

```

        {
            posit-=13;

```

```

            cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);

```

```

            cursor_bar(X1+6,Y1+6,X2-6,Y1+26,DARKGRAY);

```

```

            print_filenames(posit,tail,X1,Y1);

```

```

        }

```

```

    }

```

```

    break;

```

```

    case 4:if(posit!=(tail-(POS3*13)))    /*DOWN*/

```

```

        { if(POS3==12)

```

```

            {

```

```

                posit+=13;

```

```

                cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);

```

```

            cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);

```

```

                print_filenames(posit,tail,X1,Y1);

```

```

            }

```

```

        else

```

```

        {

```

```

            cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),BLUE);

```

```

            POS3++;

```

```

            cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);

```

```

                print_filenames(posit,tail,X1,Y1);

```

```

            }

```

```

        }

```

```

    break;

```

```

    case 5:if(POS3!=1)                /*PgUp*/

```

```

        {
            cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);
            POS3=1;
            cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);
            print_filenames(posit,tail,X1,Y1);
        }
        else
        {
            if(posit!=head)
            {
                for(i=1;(i<=11)&&(posit!=head);i++) posit-=13;
                cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);
                cursor_bar(X1+6,Y1+6,X2-6,Y1+26,DARKGRAY);
                print_filenames(posit,tail,X1,Y1);
            }
        }
        break;
    case 6: if(posit!=(tail-(POS3*13)))          /*PgDown*/
        { if(POS3==12)
            {
                for(i=1;(i<=11)&&(posit!=(tail-(POS3*13)));i++) posit+=13;
                cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);
                cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);
                print_filenames(posit,tail,X1,Y1);
            }
            else
            {
                cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);
                POS3=12;
                cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);
                print_filenames(posit,tail,X1,Y1);
            }
        }
        break;
    case 7:          /*Home*/
        posit=head;
        POS3=1;
        cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);
        cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);
        print_filenames(posit,tail,X1,Y1);
        break;
    case 8:          /*End*/
        posit=tail-(12*13);
        POS3=12;
        cursor_bar(X1+6,Y1+6,X2-6,Y2-6,BLUE);
        cursor_bar(X1+6,Y1+6+((POS3-1)*20),X2-6,Y1+6+(POS3*20),DARKGRAY);
        print_filenames(posit,tail,X1,Y1);
        break;
    case 13: posit=posit+(13*(POS3-1));  /*ENTER*/
            strcpy(which_file,posit);

```

```

        load_up(which_file);
        POS3=1; contin=1;
        break;
    case 27: POS3=1; contin=1; break;    /*ESC*/
    }
}
free(head);
restore_screen(X1,Y1,2);
}

char *filelist(char *head,char *dir)
{
    char *firstfile,*tail;
    struct ffbk ffbk;
    int done;
    done = findfirst(dir,&ffbk,1);
    firstfile=head;
    while (!done)
    {
        strcpy(firstfile,ffbk.ff_name);
        firstfile+=13;
        done = findnext(&ffbk);
    }
    tail=firstfile;
    return(tail);
}

void print_filenames(char *posit,char *tail,int X1,int Y1)
{
    char *filepos;
    int i=0;
    setcolor(WHITE);
    settxtstyle(DEFAULT_FONT,HORIZ_DIR,1);
    filepos=posit;

    do { outtextxy(X1+10,Y1+15+(i*20),filepos);
        i++; filepos+=13;
    } while ( i<12 && filepos !=tail );
}

void load_up(char file[13])
{
    switch(POS1)
    {
        case 1: switch(POS2)
                {
                    case 1: read_template(file); TEM_OK=1; break;
                    case 2: read_trainingset(file); TSE_OK=1; break;
                }
        break;
    }
}

```

```

    }
}

void save_name(int X1,int Y1,int X2,int Y2)
{
    int i=0,READ=1;
    char c[2];
    c[0]=0;c[1]=0;
    for(i=0;i<30;i++) save_file_name[i]=0;
    save_screen(X1,Y1,X2+Shadow,Y2+Shadow,2);
    cursor_bar(X1+Shadow,Y1+Shadow,X2+Shadow,Y2+Shadow,DARKGRAY);
    window1(X1,Y1,X2,Y2);
    outtextxy(X1+65,Y1+10,"Save file as");
    setfillstyle(SOLID_FILL,CYAN);
    bar(X1+10,(Y1+Y2)/2,X2-10,Y2-10);
    i=0;
    while(READ)
    {
        switch(*c=getch())
        {
            case 13: READ=0; break;
            case 8: i--; save_file_name[i]=0;
                    bar(X1+10,(Y1+Y2)/2,X2-10,Y2-10);
                    break;
            case 27: restore_screen(X1,Y1,2); return;
            default: strcpy(&save_file_name[i],c);
                    i++; break;
        }
        setcolor(WHITE);
        outtextxy(X1+20,(Y1+Y2)/2,save_file_name);
    }
    save_up();
    restore_screen(X1,Y1,2);
}

void save_up()
{
    switch(POS2)
    {
        case 1: write_output(); break;
        case 2: write_template(); break;
        case 3: write_setup(); break;
    }
}

void algorithm()
{
    int contin=0; POS3=1;
    save_screen(Setup_box_X1,Setup_box_Y1,Setup_box_X1+154,Setup_box_Y1+80);

```

```

window1(Setup_box_X1,Setup_box_Y1,Setup_box_X1+154,Setup_box_Y1+80);
cursor_bar(Setup_box_X1+6,Setup_box_Y1+6,Setup_box_X1+148,Setup_box_Y1+40,DARKGRAY);
algorithm_text();
while(contin==0)
{
    switch(stroke())
    {
        case 1: cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+
            (POS3-1)*34,Setup_box_X1+148,Setup_box_Y1+40+(POS3-1)*34,BLUE);
            POS3--; if(POS3==0) POS3=2;
            cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+
            (POS3-1)*34,Setup_box_X1+148,Setup_box_Y1+40+
            (POS3-1)*34,DARKGRAY);
            algorithm_text();
            break;
        case 4: cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+
            (POS3-1)*34,Setup_box_X1+148,Setup_box_Y1+40+(POS3-1)*34,BLUE);
            POS3++; if(POS3==3) POS3=1;
            cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+
            (POS3-1)*34,Setup_box_X1+148,Setup_box_Y1+40+
            (POS3-1)*34,DARKGRAY);
            algorithm_text();
            break;
        case 13: if(POS3==1) { strcpy(algorithm_type,"B-RBLA"); algorithm_text();}
            else { strcpy(algorithm_type,"P-RPLA"); algorithm_text();}
            break;
        case 27: contin=1;break;
    }
}
restore_screen(Setup_box_X1,Setup_box_Y1,2);
}

void algorithm_text()
{
    if(algorithm_type[0]=='B') { settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
        setcolor(LIGHTGREEN);
        outtextxy(Setup_box_X1+15,Setup_box_Y1+14,"RBLA");
        setcolor(WHITE); outtextxy(Setup_box_X1+15,Setup_box_Y1+44,"RPLA");
    }
    else { settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
        setcolor(WHITE); outtextxy(Setup_box_X1+15,Setup_box_Y1+14,"RBLA");
        setcolor(LIGHTGREEN);
        outtextxy(Setup_box_X1+15,Setup_box_Y1+44,"RPLA");
    }
}

void method()
{

```

```

    int contin=0; POS3=1;
    save_screen(Setup_box_X1,Setup_box_Y1,Setup_box_X1+147,Setup_box_Y1+80);
    window1(Setup_box_X1,Setup_box_Y1,Setup_box_X1+147,Setup_box_Y1+80);
    cursor_bar(Setup_box_X1+6,Setup_box_Y1+6,Setup_box_X1+141,Setup_box_Y1+40,DARKGRAY);
    method_text();

    while(contin==0)
    {
        switch(stroke())
        {
            case 1: cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+(POS3-1)*34,Setup_box_X1+141,Setup_box_Y1+40+(POS3-1)*34,BLUE);
                    POS3--; if(POS3==0) POS3=2;
                    cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+(POS3-1)*34,Setup_box_X1+141,Setup_box_Y1+40+(POS3-1)*34,DARKGRAY);
                    method_text();
                    break;
            case 4: cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+(POS3-1)*34,Setup_box_X1+141,Setup_box_Y1+40+(POS3-1)*34,BLUE);
                    POS3++; if(POS3==3) POS3=1;
                    cursor_bar(Setup_box_X1+6,Setup_box_Y1+6+(POS3-1)*34,Setup_box_X1+141,Setup_box_Y1+40+(POS3-1)*34,DARKGRAY);
                    method_text();
                    break;
            case 13: if(POS3==1) { strcpy(method_type,"Deterministic"); method_text(); }
                     else      { strcpy(method_type,"Stochastic"); method_text(); }
                     break;
            case 27: contin=1;break;
        }
    }
    restore_screen(Setup_box_X1,Setup_box_Y1,2);
}

void method_text()
{
    if(method_type[0]=='D') { settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
    setcolor(LIGHTGREEN);
    outtextxy(Setup_box_X1+15,Setup_box_Y1+14,"Deterministic");
    setcolor(WHITE); outtextxy(Setup_box_X1+15,Setup_box_Y1+44,"Stochastic");
    }
    else { settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
    setcolor(WHITE); outtextxy(Setup_box_X1+15,Setup_box_Y1+14,"Deterministic");
    setcolor(LIGHTGREEN);
    outtextxy(Setup_box_X1+15,Setup_box_Y1+44,"Stochastic");
    }
}

void formula()

```

```

{
    int contin=0; POS3=1;

    save_screen(Setup_box_X1,Setup_box_Y1,Setup_box_X1+144,Setup_box_Y1+110);
    window1(Setup_box_X1,Setup_box_Y1,Setup_box_X1+144,Setup_box_Y1+110);
    cursor_bar(Setup_box_X1+6,Setup_box_Y1+8,Setup_box_X1+138,Setup_box_Y1+40,DARKGRAY);
    formula_text();
    while(contin==0)
    {
        switch(stroke())
        {
        case 1: cursor_bar(Setup_box_X1+6,Setup_box_Y1+8+(POS3-1)*32,Setup_box_X1+138,Setup_box_Y1+40+(POS3-1)*32,BLUE);
                POS3--; if(POS3==0) POS3=3;
                cursor_bar(Setup_box_X1+6,Setup_box_Y1+8+(POS3-1)*32,Setup_box_X1+138,Setup_box_Y1+40+(POS3-1)*32,DARKGRAY);
                formula_text();
                break;
        case 4: cursor_bar(Setup_box_X1+6,Setup_box_Y1+8+(POS3-1)*32,Setup_box_X1+138,Setup_box_Y1+40+(POS3-1)*32,BLUE);
                POS3++; if(POS3==4) POS3=1;
                cursor_bar(Setup_box_X1+6,Setup_box_Y1+8+(POS3-1)*32,Setup_box_X1+138,Setup_box_Y1+40+(POS3-1)*32,DARKGRAY);
                formula_text();
                break;
        case 13: switch(POS3)
                {
                    case 1: strcpy(x_type,"e"); formula_text(); break;
                    case 2: strcpy(x_type,"h"); formula_text(); break;
                    case 3: strcpy(x_type,"r"); formula_text(); break;
                }
                break;
        case 27: contin=1;break;
        }
    }
    restore_screen(Setup_box_X1,Setup_box_Y1,2);
}

void formula_text()
{
    switch(x_type[0])
    {
    case 'e': settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
              setcolor(LIGHTGREEN);
              outtextxy(Setup_box_X1+15,Setup_box_Y1+14,"Euler");
              setcolor(WHITE); outtextxy(Setup_box_X1+15,Setup_box_Y1+44,"Heun");
              setcolor(WHITE); outtextxy(Setup_box_X1+15,Setup_box_Y1+74,"Runge-Kutta");
              break;
    }
}

```

```

        {
            strcpy(&c_beta[i],c);
            i++;
            c_beta[i]=0;
        }
        break;
    }
    setcolor(WHITE);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
    outtextxy(Setup_box_X1+30,Setup_box_Y1+75,c_beta);
}
if(c_beta[0]) beta=atof(c_beta);
restore_screen(Setup_box_X1,Setup_box_Y1,2);
}

void learningrate_change()
{
    int i=0,READ=1;
    char c[1],c_learn[11];
    char learn_text[30]="Coefficient= ";

    c[0]=0;c[1]=0;
    c_learn[0]=0;
    c_learn[1]=0;

    save_screen(Setup_box_X1,Setup_box_Y1,Setup_box_X1+260,Setup_box_Y1+90);
    window1(Setup_box_X1,Setup_box_Y1,Setup_box_X1+260,Setup_box_Y1+90);
    sprintf(buffer,"%7.3e",learningrate);
    strcat(learn_text,buffer);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
    outtextxy(Setup_box_X1+15,Setup_box_Y1+15,learn_text);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1); setcolor(WHITE);
    outtextxy(Setup_box_X1+15,Setup_box_Y1+48,"Enter new learningrate value");
    cursor_bar(Setup_box_X1+6,Setup_box_Y1+60,+Setup_box_X1+254,Setup_box_Y1+84,CYAN);
    while(READ)
    {
        switch(*c=getch())
        {
            case 13: READ=0; break;
            case 8: i--; c_learn[i]=0;

            cursor_bar(Setup_box_X1+6,Setup_box_Y1+60,Setup_box_X1+254,Setup_box_Y1+84,CYAN);

            break;
            case 27: restore_screen(Setup_box_X1,Setup_box_Y1,2);return;
            default: if((*c>=48)&&(*c<=57)||(*c==46))
                {
                    strcpy(&c_learn[i],c);
                    i++;

```



```

        c_learn[i]=0;
    }
    break;
}
setcolor(WHITE);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(Setup_box_X1+30,Setup_box_Y1+65,c_learn);
}

if(c_learn[0]) learningrate=atof(c_learn);
restore_screen(Setup_box_X1,Setup_box_Y1,2);
}
void cycle_change()
{
    int i=0,READ=1;
    char c[1],c_cycle[11];
    char cycle_text[20]="Cycle = ";

    c[0]=0;c[1]=0;
    c_cycle[0]=0;
    save_screen(Setup_box_X1,Setup_box_Y1,Setup_box_X1+175,Setup_box_Y1+100);
    window1(Setup_box_X1,Setup_box_Y1,Setup_box_X1+175,Setup_box_Y1+100);
    sprintf(buffer,"%d",cycle);
    strcat(cycle_text,buffer);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
    outtextxy(Setup_box_X1+25,Setup_box_Y1+15,cycle_text);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
    setcolor(WHITE);
    outtextxy(Setup_box_X1+25,Setup_box_Y1+45,"Enter new cycle ");
    outtextxy(Setup_box_X1+25,Setup_box_Y1+55,"value");
    cursor_bar(Setup_box_X1+6,Setup_box_Y1+70,Setup_box_X1+169,Setup_box_Y1+
94,CYAN);
    while(READ)
    {
        switch(*c=getch())
        {
            case 13: READ=0; break;
            case 8: i--; c_cycle[i]=0;

            cursor_bar(Setup_box_X1+6,Setup_box_Y1+70,Setup_box_X1+169,Setup_box_Y1+
94,CYAN);

            break;
            case 27: restore_screen(Setup_box_X1,Setup_box_Y1,2);return;
            default: if((*c>=48)&&(*c<=57))
            {
                strcpy(&c_cycle[i],c);
                i++;
                c_cycle[i]=0;
            }
            break;

```

```

    }
    setcolor(WHITE);
    settxtstyle(DEFAULT_FONT,HORIZ_DIR,2);
    outtextxy(Setup_box_X1+30,Setup_box_Y1+75,c_cycle);
}
if(c_cycle[0]) cycle=atoi(c_cycle);
restore_screen(Setup_box_X1,Setup_box_Y1,2);
}

void reset_func()
{
    int i; bound(); TEM_OK=0; TSE_OK=0;
    First_K=1; N=0;M=0; image_no=1;
    Totalerror=0; OLD_Totalerror=0; Error_counter=0;
    for(i=1;i<=ts_entry_quantity;i++) { Error[i]=0; OLD_Error[i]=0; }
    for(i=0;i<=4;i++)
        { t_a[i]=0; a[i]=0; color_a[i]=0;
          t_b[i]=0; b[i]=0; color_b[i]=0; }
    t_current=0;
    current=0;
    color_current=0;
    clear_status();
    show_template();
}

void step1() /* Menu selection */
{
    show_template();

    while(RUN==0)
    {
        switch(stroke())
        {
            case 2: cursor_bar(Bar1_X1+(Bar1_step)*(POS1-1),Bar1_Y1,
                               Bar1_X2+(Bar1_step)*(POS1-1),Bar1_Y2,
                               CYAN);
                    POS1--; if(POS1==0) POS1=6;
                    cursor_bar(Bar1_X1+(Bar1_step)*(POS1-1),Bar1_Y1,
                               Bar1_X2+(Bar1_step)*(POS1-1),Bar1_Y2,
                               BLUE);
                    menutext(0); /* RUN appears */
                    break;

            case 3: cursor_bar(Bar1_X1+(Bar1_step)*(POS1-1),Bar1_Y1,
                               Bar1_X2+(Bar1_step)*(POS1-1),Bar1_Y2,
                               CYAN);
                    POS1++; if(POS1==7) POS1=1;
                    cursor_bar(Bar1_X1+(Bar1_step)*(POS1-1),Bar1_Y1,
                               Bar1_X2+(Bar1_step)*(POS1-1),Bar1_Y2,
                               BLUE);

```

```

        menutext(0); /* RUN appears */
        break;

    case 13: step2(); break;
    } } }

void step2() /* If Enter */
{
    switch(POS1)
    {
        case 1: step3(Loadm_X1, Loadm_Y1, Loadm_X2, Loadm_Y2,
                      Loadm_entry, Loadm_text);
                show_template();
                break;
        case 2: step3(Savem_X1, Savem_Y1, Savem_X2, Savem_Y2,
                      Savem_entry, Savem_text);
                break;
        case 3: if(!TEM_OK)
                {
                    strcpy(message_text1, "Initial Template not");
                    strcpy(message_text2, "loaded");
                    message();
                    break;
                }
                if(!TSE_OK)
                {
                    strcpy(message_text1, "Trainingset not loaded");
                    strcpy(message_text2, " ");
                    message();
                    break;
                }
                else
                { RUN=1; break; }
        case 4: step3(Setupm_X1, Setupm_Y1, Setupm_X2, Setupm_Y2,
                      Setupm_entry, Setupm_text);
                break;
        case 5: step3(Envirm_X1, Envirm_Y1, Envirm_X2, Envirm_Y2,
                      Envirm_entry, Envirm_text);
                break;
        case 6: step3(Exit_X1, Exit_Y1, Exit_X2, Exit_Y2,
                      Exit_entry, Exit_text);
                break;
    } }

void step3( int X1, int Y1, int X2, int Y2, int entry, char *text[])
{
    int contin=0; /* SubMenu */

    save_screen(X1, Y1, X2+Shadow, Y2+Shadow, 1);
    cursor_bar(X1+Shadow, Y1+Shadow, X2+Shadow, Y2+Shadow, DARKGRAY);

```

```

window1(X1,Y1,X2,Y2);
cursor_bar(X1+6,Y1+10,X2-6,Y1+10+30,DARKGRAY);
window1_text(X1,Y1,0,entry,text);

while(contin==0)
{
    switch(stroke())
    {
case 1: cursor_bar(X1+6,Y1+10+(30*(POS2-1)),X2-6,Y1+10+(30*POS2),
    BLUE);
        window1_text(X1,Y1,POS2,entry,text);
        POS2--; if(POS2==0) POS2=entry;
        cursor_bar(X1+6,Y1+10+(30*(POS2-1)),X2-6,Y1+10+(30*POS2),
        DARKGRAY);
        window1_text(X1,Y1,POS2,entry,text);
        break;
case 4: cursor_bar(X1+6,Y1+10+(30*(POS2-1)),X2-6,Y1+10+(30*POS2),
    BLUE);
        window1_text(X1,Y1,POS2,entry,text);
        POS2++; if(POS2==(entry+1)) POS2=1;
        cursor_bar(X1+6,Y1+10+(30*(POS2-1)),X2-6,Y1+10+(30*POS2),
        DARKGRAY);
        window1_text(X1,Y1,POS2,entry,text);
        break;
case 13: contin=step4(); break;
case 27: POS2=1; contin=1; break;
    } }
    restore_screen(X1,Y1,1);
}

int step4()
{
    switch(POS1)
    {
        case 1: switch(POS2) /* Load Submenu */
            {
case1:file_menu(Load_f_X1,Load_f_Y1,Load_f_X1+120,Load_f_Y1+260,temfiles);
            break;
case2:file_menu(Load_f_X1,Load_f_Y1,Load_f_X1+120,Load_f_Y1+260,tsefiles);
            break;
case 3: POS2=1; return(1);
            }
        return(0);
        case 2: switch(POS2) /* Save Submenu */
            {
case 1: save_name(Save_box_X1,Save_box_Y1,
                Save_box_X1+250,Save_box_Y1+70);
                break;
case 2: save_name(Save_box_X1,Save_box_Y1,
                Save_box_X1+250,Save_box_Y1+70);
            }
    }
}

```

```

        break;
    case 3: save_name(Save_box_X1,Save_box_Y1,
        Save_box_X1+250,Save_box_Y1+70);
        break;
    case 4: POS2=1; return(1);
    }
    return(0);

case 4: switch(POS2) /* Setup Submenu */
    {
        case 1: algorithm(); break;
        case 2: method(); break;
        case 3: formula(); break;
        case 4: beta_change(); break;
        case 5: learningrate_change(); break;
        case 6: cycle_change(); break;
        case 7: POS2=1; return(1);
    }
    return(0);
case 5: switch(POS2) /* Environment Submenu */
    {
        case 1: break; case 2: break; case 3: break;
        case 4: break; case 5: POS2=1; return(1);
    }
    return(0);
case 6: switch(POS2) /*Exit Submenu*/
    {
        case 1: reset_func(); return(1); case 2: closegraph(); exit(1);
    }
    return(0);
}
return(0); }

```

ÖZGEÇMİŞ

Sinan Karamahmut, 1969 yılında Almanya 'nın Hannover kentinde doğdu. Grund und Hauptschule Mühlenberg İlkokulu 'nu 1979 yılında, Integrierte Gesamtschule Mühlenberg Ortaokulu 'nu 1984 yılında tamamladı. Aynı yıl içerisinde Türkiye 'ye kesin dönüş yaptı. 1987 yılında Üsküdar Anadolu Lisesi 'nden beşinci olarak mezun olduktan sonra İ.T.Ü. Elektrik-Elektronik Fakültesi, Elektronik ve Haberleşme Bölümü 'ne girdi. Eylül 1991 'de bu fakülteden, Eylül ayı bitirenler arasında 3'üncü olarak mezun oldu. Halen İ.T.Ü. Elektrik-Elektronik Fakültesi, Devreler ve Sistemler Anabilim Dalı 'nda Araştırma Görevlisi olarak çalışmaktadır. Çok iyi derecede Almanca ve İngilizce bilmektedir.

