YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞI YARDIMIYLA DESEN TANIMA ve BULUNAN DÜZELTME FONKSİYONUNUN İNCELENMESİ

Matematik Mühendisi Ayşegül ERKEN

F.B.E.Matematik Mühendisliği Anabilim Dalı Matematik Mühendisliği Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Prof. Dr. Mustafa SİVRİ

İSTANBUL, 2000

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

ÖNSÖZ

Bu çalışmada bana desteğini esirgemeyen, yol gösteren tez danışmanım Sayın Prof. Dr. Mustafa SİVRİ' ye, yardımlarını esirgemeyen, bana her zaman değerli zamanlarını ayıran Arş. Gör. Işım Genç Demiriz ve Arş. Gör. Dr. Elif Tekin Tarım'a çok teşekkür ederim.

ÖZET

YAPAY SİNİR AĞI YARDIMIYLA DESEN TANIMA ve BULUNAN DÜZELTME FONKSİYONUNUN İNCELENMESİ

Yapay Sinir Ağı; karar verme problemlerinde sıkça kullanılan Yapay Zeka tekniklerinden biridir. Yapay Sinir Ağı, teorik hale getirilmiş zeka ve beyin faaliyetlerinin matematiksel modelidir. Bu konu üzerinde konuşma, görüntü tanıma, karar verme, kontrol gibi çeşitli alanlarda insan gibi bir davranış elde etmek amacıyla çalışılmaktadır.

Bu tezde Yapay Sinir Ağı konusu genel olarak tanıtılmış ve temel noktalar üzerinde durulmuştur. Bazı önemli Yapay Sinir Ağı modelleri tanıtılmış ve bunların kullanılmasını sağlayan algoritmalar verilmiştir. Uygulama olarak şekil-desen tanıma örneği verilmiştir. Bu örnek Pascal programlama dilinde yazılmıştır. Sistemde 10*10 'luk desen alanı olan 4 sınıflı 100 nöron tanımlanmış ve bunlar eğitilmiştir. Rastgele desen seçilerek [1, 1/20] aralığında bozulma oranları verilerek, elde edilen bozuk desen sisteme verilmiştir. Buradaki amaç; gelen bozuk desenin düzgün desenler içinde hangisine benzediğini bulmaktır. Bunun için Hopfield Algoritması kullanılmıştır. Hopfield algoritmasının yaptığı iş, eğittiğimiz desenlerin ağırlıklı ortalamasını bulup, bunları hücreler arası ağırlık katsayılarına paylaştırmak ve bizim verdiğimiz bozuk deseni bu ağırlıktan geçirerek doğru değere yakınsatmaktır. Bu algoritma, girdiyi doğru olana yakınsatana kadar işlemlerine devam eder. Hopfield her zaman tam bir sonuç veremeyeceğinden, Hopfield ' dan çıkan sonuç, gelen desenin düzgün desenler içinde en çok hangisine uyduğunu puan vererek araştıran GiveMaxSimilar prosedüründen geçirilmiştir. 100 deneme yapılarak bozulma oranlarına karşılık gelen düzeltme oranları bulunmuş ve SPSS 8.0 programıyla düzeltme fonksiyonu bulunmuştur.

Son olarak, düzeltme fonksiyonunun olasılık yoğunluk fonksiyonu bulunmuş, ve bozulma oranının düzgün dağılım olması halinde olasılık yoğunluk fonksiyonunun beklenen değeri hesaplanmıştır.

Anahtar Kelimeler: Yapay Sinir Ağı, Desen tanıma, Hopfield algoritması, Yoğunluk fonksiyonu, Beklenen değer

ABSTRACT

PATTERN RECOGNITION BY USING ARTIFICIAL NEURAL NETWORKS AND EXAMINATION OF THE FOUND CORRECTION FUNCTION

Artificial Neural Network is one of the techniques of Artificial Intelligence which is often used in decision making problems. Artificial Neural Network is the mathematical model of intelligence and brain activities which are theorized. This topic is studied to obtain behaviors that are similar to that of human-beings in various areas such as speech recognition, image recognition, decision making, control.

In this thesis, Artificial Neural Network is generally described and some basic points are presented. Some important Artificial Neural Network models are described and the algorithms related with these models are given. As an application, pattern recognition sample is given. This sample is written in Pascal programming language. In the system, There are 100 neurons with 4 class which has 10*10 pattern areas. And they are trained. Patterns are selected randomly and corruption ratio is given in the interval of [1,1/20] and corrupted pattern is given into the system. The aim is to find the corrupted pattern given by the system belongs to which correct pattern. Hopfield Network is used in order to find it. The task of Hopfield Network is to find the weighted mean of the trained patterns and to distribute them to the weight coefficient between cells and to pass corrupted pattern through this weight in order to converge on the exactly correct value. This algorithm continues to perform its iterations until converging the input to the correct value. Since Hopfield doesn't always give the exact result, the consequence of Hopfield algorithm passes from Give Max Similar procedure. 100 tests are made and correct ratios which correspond to corrupted patterns are found. Correct function is found by SPSS 8.0 programme.

Finally, when the corruption ratio is uniform distrubition, the expected value of the probability density function is calculated.

Keywords: Artificial neural networks, Pattern recognition, Hopfield algorithms, Density function, Expected value.

1. AKILLI KARAR SİSTEMLERİ

1. 1 Giriş

Günümüzün yaşam koşullarında, hızlı ve etkin karar verme kaçınılmazdır. 1950'lerden bu yana bilgisayarlar, karar verme durumunda olan kişilerin hep yanında olmuştur. Ancak bilgisayarların rolü bilgileri hızlı işleyip karar vericiye karar vermesini kolaylaştıracak ve hızlandıracak şekilde sunması ile sınırlı kalmıştır. Oysa bugün, gelişen teknoloji ve yeni bilgi temsili metodlarının ortaya çıkması bilgisayarların da karar verme veya en azından ulaştıkları sonucu açıklama yeteneklerine sahip olabileceğini göstermiştir.

Karar problemlerinin çözümü için günümüzde akıllı karar sistemleri geliştirmeye doğru bir eğilim vardır. Bilgi teknolojisinin gelişmesi ile klasik bilgi sistemlerinden (Yönetim Bilgi Sistemi, Karar Destek Sistemi v.b.) modern bilgi sistemlerine bir geçiş yaşanmıştır. Bu geçiş, Yapay Zeka teknikleri kullanılarak olmuştur.

Karar verme problemlerinde sıkça kullanılan Yapay Zeka teknikleri Uzman Sistemler ve Yapay Sinir Ağlarıdır. Bunlar ya tek başlarına ya biri diğeri ile birleştirilerek ya da diğer bazı yeni tekniklerle (Bulanık Kümeler, Akıllı Veritabanı Sistemleri v.b) bütünleştirilerek akıllı karar sistemlerinin temelini oluşturur.

Gerçek hayatta karşılaşılan birçok problemde "en iyi" (optimal) kararın bulunması çok önemlidir. Doğru ve zamanında verilmiş kararlar verimliliği ve parasal girdinin etkin biçimde kullanılmasını sağlar. Karar sayısı az ise, kararların en iyisini seçmek kolay olabilir. Ancak çoğunlukla, yeteri kadar küçük problemler için bile, bu seçim karmaşık bir süreç gerektirir. 1950'lerden bu yana bilgisayarlar karar verme durumunda olan kişilerin hep yanında olmuştur. Ancak uzun bir süre bilgisayarların rolü, sadece bilgileri hızlı işleyip karar vericiye karar vermesini kolaylaştıracak ve hızlandıracak şekilde sunması ile sınırlı kalmıştır. Bugün bilgisayar ve çevre birimleri ile bilgi iletişimdeki baş döndürücü gelişmeler sayesinde, bilgisayarlar bu görevlerini en iyi şekilde yerine getirmektedirler. Ama insanoğlu, bilgisayarların daha da fazlasını, örneğin karar verme ve bir çok karar içinden en iyisini seçme gibi görevleri de üstlenmesini istemektedir.

Karar verme sürecinde bilgisayarların, stratejik kararları verecek olan karar vericinin yerini alması hiç bir şekilde olası olmamasına rağmen bugün, gelişen teknoloji, bilgi çeşitliliğinin ve miktarının artması, yeni bilgi temsil yöntemlerinin (Kurallar (Rulles), Çerçeveler (Frames), Anlam Ağları (Semantic Networks), Yapay Sinir Ağları (Artificial Neural Networks) v.b. ortaya çıkması ve Yapay Zeka (YZ) çalışmalarındaki gelişmeler, bilgisayarların da karar

vericinin yerine "karar verme / alma" veya en azından ulaştıkları sonucu "açıklama" yeteneklerine sahip olabileceğini göstermektedir.

Bu özelliği bilgisayarlara kazandıran sistemlere Bilgi Tabanlı Sistemler (Knowledge Based Systems) (BTS) veya kısaca akıllı sistemler (intelligent systems) adı verilmektedir. Bilgi Tabanlı Sistemler, bir insan-uzman tarafından gerçekleştirilebilecek bir dizi faaliyeti gerçekleştirmek için bir veya daha fazla yapay zeka tekniğini ve gerektiğinde diğer bazı kuram ve yöntemleri kullanan bir bilgisayar sistemi (donanım ve yazılım) olarak tanımlanabilir.

1. 2 Karar Verme ve Klasik Karar Sistemleri

İster işletmecilikte, ister bilimde isterse tıp alanında olsun, bir problem ile ilgili çeşitli verilerden karar vermeyi kolaylaştıracak bilgileri elde etme sürecinden (karar analizi) sonra elde edilen bilgiler doğru zamanda, doğru yerde, doğru şekilde olmalıdır ki, karar analisti durumundaki yönetici, bilim adamı veya hekim etkin ve doğru kararlar verebilsin. Bu kişiler, onlara kararlarını vermelerinde yardımcı olacak bilgileri (enformasyon) genellikle kendi bilgilerinin ve zekalarının yanısıra Bilgi İşlem Sistemi (BİS), Yönetim Bilgi Sistemi (YBS), Karar Destek Sistemi (KDS) gibi klasik sistemleri veya akıllı sistemleri kullanarak elde edebilir .

Kendisinden daha gelişmiş sistemlerin bir bileşeni olan Bilgi İşlem Sistemi ile işlenmemiş, ham verilerden karar vermeyi kolaylaştıracak bilgiler (istatistiksel, grafiksel v.b.) elde edilir. Bilgisayarla ilgisi olan herkes bu sistemi kullanır.

Yönetim Bilgi Sisteminin kesin bir tanımı olmamasına rağmen, karar vermeyi doğrudan etkileyecek özellikte ve şekilde yönetim bilgisi, yani yöneticinin planlama, örgütleme, kontrol işlevlerini yerine getirmesine yardımcı olan bilgiyi üreten sistem olarak düşünülebilir. Yönetim Bilgi Sistemi, bilgisayarlardan çok daha önce, insanların ortak amaçlarını gerçekleştirmek amacı ile biraraya geldikleri andan itibaren geçerli olan bir kavramdır. Günümüzde Yönetim Bilgi Sistemini bilgisayarlar ve bilgisayarların olanakları ile bütünleştirmek alışılagelmiştir. Modern Yönetim Bilgi Sistemleri bilgisayarları dolayısıyla Bilgi İşlem Sistemini de kapsayacak şekilde tasarlanır. Yönetim Bilgi Sistemleri, karar vermeyi doğrudan etkileyecek bilgiler üretmesine rağmen pasif sistemlerdir ve karar vermedeki rolleri sınırlıdır. Yönetim Bilgi Sistemleri yaygınlaşmaya başladıkları sıralarda Yönetim Bilimi, Yöneylem Araştırması, Sistem Analizi ve Sistem Mühendisliği gibi alanlardaki gelişmeler sonucunda karar analizi sürecinde Marjinal analiz, Giriş çıkış analizi, Kuyruk teorisi, Envanter teorisi, Proje programlama (PERT CPM), Güvenirlilik ve kalite

kontrol, Tahmin, Grup teknolojisi (parçaların sınıflandırılması ve gruplanması) gibi yeni yöntemler kullanılmaya başlanmıştır. İşte Karar Destek Sistemi, Yönetim Bilgi Sistemi ile bu yöntemlerden uygun olan bir tanesinin birleştirilmesinden oluşur. Son yıllarda ise bu araçların yanısıra Yapay Zeka teknikleri de aynı amaçla kullanılmakta ve akıllı karar sistemleri doğmaktadır .

1. 3 Yapay Zeka ve Akıllı Karar Sistemleri

İnsan davranışlarını benzeterek çalışacak bilgisayarların veya akıllı sistemlerin gerçekleştirilebilmesi için yapılan çalışmaların başlangıcı da 1950'lere dayanmaktadır. 1950'lerin sonlarında bu alanda çalışan bir grup bilimadamı dikkatlerini iki yaklaşım üzerinde yoğunlaştırmışlardı. Birincisi, beynin biyolojik yapısına bakılmaksızın, sadece insan davranışları taklit edilerek akıllı sistemlerin geliştirilmesi; ikincisi ise beynin yapısı ve çalışması benzetilerek akıllı sistemlerin gerçekleştirilmesidir. Birinci yaklaşımdan Yapay Zeka, ikinci yaklaşımdan da Yapay Zekanın bir alt dalı olan Yapay Sinir Ağları doğmuştur.
Yapay Zeka çalışmalarının önde gelen isimlerinden Feigenbaum ve McCorduck Yapay Zekayı "bilginin sembolik olarak temsili ile mantıksal işlemleri yapan tekniklerin ve insana özgü akıllılık derecesinde hareket eden bilgisayar programlarının geliştirilmesi" olarak tanımlamaktadır . Yapay Zekanın amacı ise "araçtan bağımsız zekanın yapısını araştırmak ve ileri yazılım teknikleri yardımı ile akıllı makineler yapmaktır. Yapay Zeka, disiplinlerarası bir araştırma alanı olarak birçok bilim dalından etkilenmiştir. Gittikçe gelişen ve yenilenen Yapay Zeka teknikleri zamanla etkilendikleri alanları etkilemeye ve o alanlarda uygulanmaya başlamıştır. Örneğin insan görme sisteminin çalışma prensibine benzer prensiple makinelerin görmesini sağlamayı hedefleyen Makinelerde Görme (Machine Vision), kontrollu hareket edebilen mekanik araçlar üretmeyi hedeflemekle Robotik (Robotics), insan konuşmasını ve sentezini hedeflemekle Konuşma İşlem (Speech Processing), doğal dili anlama ve üretmeyi hedeflemekle Doğal Dil İşleme (Natural Language Processing), matematik ve mantık teoremlerini ispatlamayı hedeflemekle Teorem İspatlama (Teorem Proving) günlük dilde ifade edilmiş genel problemleri çözmeyi hedeflemekle Genel Problem Çözme (General Problem Solving), şekil tanıma ve sınıflandırmayı hedefleyerek Şekil Tanıma (Pattern Recognition), Oyun oynama (Game Playing), Makine Öğrenmesi (Machine Learning) gibi alt dallar ortaya çıkmıştır. Yapay Zekanın da uğraşı alanı aslında karar verme üzerine yoğunlaşmaktadır. Ancak Yapay Zekanın amacı basit anlamda karar vermeyi desteklemek değil, amacından da anlaşılabileceği gibi kendi kendine karar verebilecek makineler geliştirmektir. Böyle bir makine karar verirken "zekasını" insana özgü bir şekilde

kullanmalıdır. Başka bir deyişle deneyimle öğrenebilmeli, bilgisinin sınırlarını tanımalı ve gerçek bir yaratıcılık gösterebilmelidir.

1. 3. 1 Yapay zeka, uzman sistem, yapay sinir ağı

Günümüzde işletmeler, bilgi miktarı ve çeşitliliğinin çok fazla olduğu dinamik ortamlarda faaliyet gösterir. Finans sektörü buna güzel bir örnek teşkil eder. Bilim ve tıp kurumları için de aynı şey söylenebilir. Örneğin astronomi alanında çok fazla ve çok çeşitli gözlemsel veri (güneşe, yıldızlara, galaksilere v.b. ilişkin) mevcuttur. Biriktirilmesi yıllar alan bu verilerin analiz edilmesi ve yararlı bilgi haline dönüştürülmesi de çok zaman alır. Gerek işletmelerde gerek bilimde mikro ve makro düzeyde alınacak olan kararların istatistiksel tekniklere ve bu amaçla geliştirilmiş olan matematiksel modellere dayandırılması çok önemlidir . Ancak bu yöntemlerin klasik bilgi sistemleri ile bütünleştirilip kullanılabilmesi için verilerin tam olması ve veriler arasında bazı ilişkilerin ortaya konmuş olması gerekir. Birçok olayda ve problemde ise eksik veri ve tam belirlenememiş ilişkiler söz konusudur. Ayrıca uzmanlık gerektiren alanlarda da insan-uzmanların her an hazır bulunması söz konusu değildir. İşte, Yapay Zeka alanı içinde gelişim gösteren Uzman Sistemler, Yapay Sinir Ağları veya Uzman Sistem-Yapay Sinir Ağı, Bulanık Kümeler-Uzman Sistemler, Bulanık Kümeler-Yapay Sinir Ağlı gibi bütünleşik sistemler böyle durumlarda kullanılabilecek ve karar vericiyi tatmin edici sonuçlara ulaştırabilecek araçlardır.

Uzman Sistemler herhangi bir alanda, o alanın uzmanı olan bir insanın vereceği kararları verebilen, benzer tavsiyelerde bulunabilen "akıllı" bilgisayar yazılımlarıdır ve belli bir sahadaki problemlerin çözümü için geliştirilmiştir. Uzman Sistemlerdeki bütün bilgiler, ilgili sahadaki uzmanlar tarafından sağlanır. Böyle bir sisteme sahip olmak, kişiyi uzman yapmaz, fakat bir uzmanın yapacağı işin bir kısmını veya tamamını yapmasını sağlar. Burada en önemli nokta Uzman Sistemlerin kullanacağı bilgilerin sisteme tanıtılması , başka bir ifade ile karar vermede yardımcı olacak bilgilerin bilgisayara girilmesidir. Uzman Sistemler bilgi tabanlı sistemlerdir ve bu sistemlerin mimari yapısına sahiptirler. Uzmanlık alanı ile ilgili bilgilerin depolandığı Bilgi Tabanı ve bu bilgilerden sonuç çıkaran Çıkarım Makinesi, Uzman Sistemlerin iki temel elemanıdır. Kullanıcı Arabirimi elemanı ise Uzman Sistemlerin kullanıcı ile etkileşimini sağlamaktadır. Uzman Sistemlerde Bilgi Tabanları genellikle "NE BİLİRİZ?" sorusuna, Çıkarım Makinesi ise "NE YAPARIZ?" sorusunun cevabı olan bilgileri içerir. Bu nedenle Bilgi Tabanındaki bilgiler uzmanlık sahasına göre değişirken, Çıkarım Makinesindekiler değişmez.

Günümüzde hemen her alanda kullanılan Uzman Sistemlerin, yabancı para değerlerinin takibi ve tahmini, yatırım danışmanlığı, kredi yönetimi ve müşteri değerlendirme, faiz karşılığında ödünç para alma işlemlerini onaylama, sigorta risklerini değerlendirme, yatırım fırsatlarını değerlendirme gibi uygulamalarda kullanımının yaygınlaştığı görülmektedir Uzman Sistemler, modern bilgi sistemleri olmasına rağmen, ancak karar verme kuralların çok açık ve bilginin güvenilir olduğu problemlerde başarı ile uygulanabilmektedir. Oysa birçok alanda böyle değildir ve aşağıdaki iki durum gözlenir:

1) karar verme kuralları ya çok açık değildir veya bir kural yoktur.
2) bilgi kısmen yanlıştır.

Son yıllarda bu iki durumdan birinin veya her ikisinin görüldüğü problemlerin çözümünde Bulanık Kümeler (Fuzzy Sets) ve Yapay Sinir Ağı kullanılması önerilmektedir.

Bir Yapay Sinir Ağı, beyindeki nöronların benzeri olan birçok basit işlem elemanından (yapay nöron) oluşmuş dinamik bir bilgi temsil ve bilgi işleme sistemidir. Yapay nöronlar herbirinin belli işlevi olan katmanlar şeklinde örgütlenmiştir ve Yapay Sinir Ağının bu yapısına Yapay Sinir Ağı mimarisi denir. Yapay Sinir Ağları beynin bazı fonksiyonlarını ve özelikle öğrenme yöntemlerini benzetim yolu ile gerçekleştirmek için tasarlanır ve geleneksel yöntem ve bilgisayarların yetersiz kaldığı sınıflandırma , özörgütleme, kümeleme, duyu-veri işleme, çok-duyulu makine gibi alanlarda başarılı sonuçlar verir. Yapay Sinir Ağlarının özellikle tahmin problemlerinde kullanılabilmesi için çok fazla bilgi ile eğitilmesi gerekir. Yapay Sinir Ağların eğitimi için çeşitli algoritmalar geliştirilmiştir. Günümüzde Yapay Sinir Ağı uygulamaları ya geleneksel bilgisayarlar üzerinde yazılım simülatörleri kullanılarak, veya özel donanım içeren bilgisayarlar kullanarak gerçekleştirilmektedir.

Lapedes ve R. Farber (1987) bir Yapay Sinir Ağının çok karışık zaman serilerinin nokta tahmininde kullanılabileceğini ve elde edilen sonuçların Lineer Tahmin Metodu gibi klasik metodlara göre çok daha kesin olduğunu göstermişlerdir. Kar Yan Tam (Hong Kong Üniversitesi) ve Melody Y. Kiang (Arizona State Üniversitesi) geliştirdikleri Yapay Sinir Ağını işletmelerin iflas gibi finansal güçlüklerini tahmin etmede kullanmışlardır . Bu örnekler yüzlerce örnek uygulamadan sadece ikisidir.

Sembolik işlemin ve paralel işlemin temsilcileri olan Uzman Sistemler ve Yapay Sinir Ağları rekabet eden değil, birbirini tamamlayan iki Yapay Zeka aracıdır. Bu araçları birarada kullanarak daha güçlü analiz ve karar verme araçları elde edilebilir. Bu düşünce onların, birbirinin tersi olan ve birbirini tamamlayan özelliklerinin olmasından kaynaklanır. Uzman Sistemler çok açık ve kesin kurallar gerektiriyorken Yapay Sinir Ağları için bunun tersi geçerlidir. Diğer taraftan Yapay Sinir Ağların ulaştıkları sonuçları "açıklamaları" açıklamada çok zayıftırlar. Uzman Sistemler ise kullanıcı arabirimleri sayesinde bunu kolayca

başarabilirler . Buradan hareketle Uzman Sistem-Yapay Sinir Ağı sistemi düşüncesi doğmuştur.

1. 3. 2 Akıllı karar sistemleri

Akıllı sistemlerin tasarımında karşılaşılan en büyük zorluk insan-uzmanların bilgisini içerecek olan Bilgi Tabanı'nın kurulmasıdır. Bilgi Tabanı insanların bilgi ve uslamlama (reasoning) şeklinin bir modeli olmalıdır. İnsan uslamlaması da genellikle belirsizlik ve kesinsizlik ile başa çıkma özelliği taşır. Bu özelliği akıllı sistemlere de yansıtabilmek için Bulanık Mantık kuramından yararlanılır ve birçok çalışmada görüldüğü gibi Bulanık Akıllı Sistemler (Fuzzy İntelligente Systems), Bulanık Karar Sistemleri (Fuzzy Decision Systems) gibi kavramlar ortaya çıkmaktadır.

Akıllı bir sistemin Bilgi Tabanının kurulmasındaki zorluğu aşmak için sözkonusu sistem, *öğrenen* sistem olarak tasarlanmalıdır. Böyle bir sistemin sahip olması gereken özelliklerden bazıları şunlardır :

1. Örnekler veya açıklamalar yolu ile öğrenebilmelidir.
2. Verilerdeki ilişkili veri gruplarını keşfedebilmelidir.
3. İlişkili veri gruplarını biraraya getirerek daha yüksek seviyede bilgiler elde edebilmelidir.
4. Kullanıldıkça performansını düzeltebilmelidir.
5. Bünyesindeki bilgileri zaman zaman kullanıcı bakış açısı ile inceleyebilmelidir.
6. Kesinsizlik ile başa çıkma özelliğini taşımalıdır.

Bu durumda akıllı sistemlere en iyi adaylar Uzman Sistem-Yapay Sinir Ağı, Bulanık Uzman Sistemler, Bulanık Yapay Sinir Ağları ve Bulanık Akıllı Bilgi Sistemleri gibi bütünleşik sistemlerdir.

1.3.3 Örnek akıllı sistemler

Uzman Sistem-Yapay Sinir Ağı (US-YSA) sistemlerinde genel yaklaşım, ağdan elde edilen sonuçların bir Uzman Sistem içerisine "gömülerek " kullanılmasıdır . Bu sayede klasik Uzman Sistem yapısının sorun yaratan bilgi edinme (knowledge acquisition) aşamasına ve Yapay Sinir Ağının elde ettiği sonuçları açıklayamama problemine çözüm bulunmuş olur . Barker'ın "Küçük İşletmelerin Finansal Faaliyetlerinin Karma Bir Sistem (US-YSA) ile Tahmin Edilmesi" isimli çalışmasında, küçük işletmelerin finansal oranlarını analiz eden ve elde ettiği sonuçlara göre işletmenin borçlanarak sermayesini arttırabilme olasılığını tahmin eden bir sistem geliştirilmiştir . Finansal oranların analizi bir Uzman Sistem tarafından

kolayca yapılabilir ancak tahminde bulunmak daha zordur. Bu sebeple geliştirilen sistem Uzman Sistem ile Yapay Sinir Ağını birleştiren karma bir sistem olmuştur.

Maria Zemankova öğrenebilme özelliğine sahip bir Bulanık Akıllı Bilgi Sistemi geliştirmiştir FİLİP adını taşıyan bu sistem bir ilişkisel Veritabanı, bir Bilgi Tabanı ve Çıkarım Makinesinden oluşmuştur. Bilgi Tabanı ve Çıkarım Makinesi bulanık küme teorisine dayanır. Bilgi Tabanında yer alan kavramlar iki yoldan buraya yerleştirilir (ya açık bir şekilde tanımlanarak veya örnekler sunulup sistemin bu örneklerden tanımlar çıkarması sağlanarak). İkinci durumda sistemin öğrenmesi söz konusudur. Daha önce öğrenilmiş olan kavramlar yeni kavramların tanımlanmasında kullanılır. Kavramlar bulanık kümeler veya bulanık kurallar şeklinde temsil edilir.

1. 4 Sonuç

Burada anlatılmak istenen , temelinde Bilgi Tabanlı Sistem olan akıllı sistemlerin önemi vurgulanmak istenmiştir. Beynin yapısı ve çalışması benzetilerek akıllı sistemlerin gerçekleştirilmesini sağlayan Yapay sinir ağı bu tezde anlatacağım konudur. Yapay Sinir Ağları, Yapay Zekanın bir alt dalı olarak doğmuştur.

2. YAPAY SİNİR AĞLARI

2. 1 Giriş

Yapay sinir ağları ya da kısaca YSA; insan beyninin çalışma sisteminin yapay olarak benzetimi çabalarının bir sonucu olarak ortaya çıkmıştır. En genel anlamda bir YSA insan beynindeki birçok nöronun, ya da yapay olarak basit işlemcilerin birbirlerine değişik etki seviyeleri ile bağlanmasıyla oluşan karmaşık bir sistem olarak düşünülebilir. Önceleri temel tıp birimlerinde insan beynindeki nöronların matematiksel modelleme çabaları ile başlayan çalışmalar, geçtiğimiz 15 sene içerisinde disipline bir şekil almıştır. YSA bugün fizik, matematik, elektrik ve bilgisayar mühendisliği gibi çok farklı bilim dallarında araştırma konusu haline gelmiştir. YSA ' nın pratik kullanımı genelde , çok farklı yapıda ve formlarda bulunabilen enformasyon verilerini hızlı bir şekilde tanımlama ve algılama üzerinedir.
Aslında mühendislik uygulamalarında YSA 'nın geniş çaplı kullanımının en önemli nedeni, klasik tekniklerle çözümü zor problemler için etkin bir alternatif oluşturmasıdır.

2. 2 Tanım

Yapay Sinir Ağları, birbirlerine yoğun bir şekilde paralel olarak bağlanmış basit işlem elemanlarından oluşmuş ve gerçek dünyadaki nesnelerle biyolojik sinir sisteminin yaptığının aynı şekilde etkileşmek üzere hiyerarşik olarak düzenlenmiş ağlardır. YSA , teorik hale getirilmiş zeka ve beyin faaliyetinin matematiksel modelleridir. YSA paralel dağılmış bir bilgi işleme sistemidir. Bu sistem tek yönlü işaret kanalları (bağlantılar) ile birbirine bağlanan işlem elemanlarından oluşur. YSA yaklaşımının temel düşüncesiyle, insan beyninin fonksiyonları arasında benzerlik vardır. Bu yüzden YSA sistemine insan beyninin modeli denilebilir. YSA çevre şartlarına göre davranışlarını şekilleyebilir. Girişler ve istenen çıkışların sisteme verilmesi ile kendisini farklı cevaplar verebilecek şekilde ayarlayabilir. Ancak son derece karmaşık bir iç yapısı vardır. Onun için bugüne kadar gerçekleştirilen YSA; biyolojik fonksiyonların temel nöronlarını örnek alarak yerine getiren kompoze elemanlardır. Aynı şekilde biyolojik beyin, tecrübe ile öğrenme ve bilgiyi kendi kendine yorumlama, hatta eksik bilgilerden sonuçlar çıkartma kabiliyetine sahiptir. Bu, daha çok biyolojik sistemlerin hücreler üzerinde dağılmış bilgiyi paralel olarak işleme özelliklerinden kaynaklanır. Hücreler birbirine bağlı ve paralel çalıştıklarından bazılarının işlevini yitirmesi halinde, diğerleri

çalıştığı için sinir sistemi, fonksiyonunu tamamen yitirmez. İşte yapay sinir ağları, bu özellikleri bünyesinde toplayacak şekilde geliştirilmiştir.

Yapay sinir ağları beynin bazı fonksiyonlarını ve özellikle öğrenme yöntemlerini benzetim yolu ile gerçekleştirmek için tasarlanır ve geleneksel yöntem ve bilgisayarların yetersiz kaldığı sınıflandırma, kümeleme, duyu-veri işleme, çok duyulu makine gibi alanlarda başarılı sonuçlar verir. Yapay sinir ağlarının özellikle tahmin problemlerinde kullanılabilmesi için çok fazla bilgi ile eğitilmesi gerekir. Ağların eğitimi için çeşitli algoritmalar geliştirilmiştir.

2. 3 Yapay Sinir Ağının Tarihçesi

1943 yılı , YSA ' nın gelişiminin başlangıç yılı olarak kabul edilir. Bu tarihte McCulloch ve Pitts, ilk hücre modelini geliştirdiler ve birkaç hücrenin arabağlaşımını incelediler. McCulloch ve Pitts' in yaptıkları ilk Yapay Sinir Ağı Sistemi (1943), vakum tüplerinden ve çeşitli aletlerden oluşuyordu. Bu sistem genel olarak basit mantıksal işlemleri çözmede kullanıldı. Fakat, sistem büyük bir alan kaplıyor, büyük parçalardan oluşuyor ve yavaş işliyordu. Yine de bu iki araştırmacı YSA araştırmalarında, ilk fonksiyonel nöronu yaparak, yeni bir devir açtılar.

Bu tarihten sonra kısaca gelişmeler şöyle gerçekleşti. Bu on yılın içerisinde (1949), Donald Hebb, bilginin, sinirler arası bağlantılarda saklanabileceğini keşfetti. Hebb, hücre bağlantılarını ayarlamak için ilk öğrenme kuralını önerdi. Daha sonra Rosenblatt 1958 'de nöron benzeri bir kavram ortaya attı. Bu kavram perceptron idi ve bu küçük sanal makine desenleri sınıflandırmayı, bağlantı ağırlıklarını değiştirerek öğrenebiliyordu. Rosenblatt, algılayıcı (perceptron) modelini ve öğrenme kuralını geliştirdi. Bugün kullanılan kuralların temelini koydu. Bu gelişmeden sonra desen tanıma üzerinde çalışmalar gittikçe hızlandı.

1960 yıllarının başında ADALINE (Adaptive Linear combiner) denen bir sistem icat edildi. Bu yeni icat Widrow-Hoff adında güçlü öğrenme kurallarına sahip idi ve Bernard Widrow-Marcian Hoff (1960-62) tarafından icat edilmişti. Bu kural kısaca şöyle idi; desen tanıma sırasında oluşan hata payları karesel bir orantı ile hesaplanıp sisteme geri gönderiliyordu. Böylece sistem, tanıma işlemini daha doğru bir şekilde yeniden yapıyordu. Bu sistem daha sonraları geliştirilerek MADALINE (Many ADALINES) yapıldı. Bu alet, desen tanımada (pattern recognition), hava durumu tahmininde ve uyumluluk kontrol sistemlerinde kullanıldı.

Bundan sonra araştırmalar, daha çok olayın formülize edilmesi ve yeni matematiksel formüller üretme çerçevesinde yapıldı. Minsky ve Papert, 1969 yılında , algılayıcının kesin analizini yaptı ve algılayanın karmaşık lojik fonksiyonlar için kullanılamayacağını

ispatladılar. Amari (1072-77), Fukushima (1980), Kohonen (1977-82, 84, 88), Anderson (1977) ve Hopfield (1082, 1984) YSA üzerinde çalışmalar yaptılar.

1976'da Grossberg, Adaptif Rezonans Teoriyi (ART), 1982'de Hopfield, optimizasyon gibi teknik problemleri çözmek için lineer olmayan dinamik hopfield ağını ve 1984' te de Kohonen eğiticisiz öğrenen bir ağ geliştirdi.

1986 yılında Rumelhart, esas olarak 1974 yılında Werbos tarafından bulunan çok katmanlı algılayıcı tipi ağlar için geriye yayılma algoritması denen bir eğitim algoritması geliştirdi. Bu çalışmasıyla , YSA alanında bir çığır açtılar. Bugün en çok kullanılan algoritma da bu geriye yayılma algoritmasıdır.

Lapedes ve R. Farber (1987) bir sinirsel ağın çok karışık zaman serilerinin nokta tahmininde kullanılabileceğini ve elde edilen sonuçların lineer tahmin metodu gibi klasik metodlara göre çok daha kesin olduğunu göstermişlerdir. Kar Yan Tam (Hong Kong Üniversitesi) ve Melody Y. Kiang (Arizona State Üniversitesi) geliştirdikleri sinirsel ağı, işletmelerin iflas gibi finansal güçlüklerini tahmin etmede kullanmışlardır.

Böylece yeni YSA araştırma programlarının ortaya çıkışları hızlandı, araştırmalar kuvvetlenmeye başladı. Bunu, dünya çapında verilen konferanslar , yeni yayınlar izledi. YSA ile çözülecek problemlerin sayısı ve bu daldaki uygulamaların adedi azlıktan çokluğa doğru hızla ilerledi. 1990'larda artık YSA' ı için özel makineler, sistemler, mikro-çipler, yazılımlar geliştirildi, araştırma grupları arttı, kullanım alanı muazzam derecede genişledi ve bir o kadar da bu dala aktarılan maddi kaynak inanılmaz derecelere yükseldi.

YSA, son yıllarda pek çok alana başarıyla uygulanmıştır. Bu alanlardan bazıları şunlardır:Fonksiyon yaklaştırma, Şekil-desen Tanıma (Pattern Recognition), Görüntü ve Ses işleme, Sistem tanıma ve kontrol , tahminde bulunma ve karakter tanıma v. b. . . Bu sistem kullanılarak, desen tanımanın yanısıra gelen bilginin (ses veya görüntü) kirliliğini, cızırtısını ve bozukluğunu en aza indirilmesi veya tamamen düzeltilmesi işlemleri başarı ile yapılabilmektedir. Böyle bir teknolojiye iş çevrelerden parasal destek de büyüktür. Bunun başlıca sebebi, YSA ile insanı taklit etmenin kolaylığı ve olabilirliğidir. Ticari çevreler için bu algoritmaların anlamı; doğruluğun maksimuma çıkartılması, tutarın (ücretin) en aza indirilmesidir.

2. 4 Yapay Sinir Ağlarının Kullanım Nedenleri

Öncelikle örnek tanıma (pattern recognition) tekniğinin gerekliliği, gerçek dünya ile bilgisayar ilişkisinin başlamasıyla ortaya çıkmıştır. Bu önem YSA 'nın çok güçlü örnek tanıma tekniği olarak ortaya çıkmasına ve gelişmesine neden olmuştur.

YSA gerçek dünyaya ait ilişkileri tanıyabilir. Sınıflandırma, kestirim (tahmin), ve fonksiyon uydurma gibi görevleri yerine getirebilirler. Bunların genel ilişkilerle, ayrık örnekler arasındaki boşluğu birbirlerine bir köprü gibi bağladığını söylemek yanlış olmaz.

YSA 'nın , ani değişen değerlerden örnekler alarak, doğrudan doğruya örneklerin birbirleri arasındaki benzer ilişkileri öğrenme yetenekleri vardır. YSA bir konuda ve olaydaki verilerden hareketle önceki bilgileri bilmese bile sonuç çıkarma yetenekleri de vardır. Çünkü YSA klasik programlama ile çalışmazlar. Bir YSA 'nın geliştirilmesi, bildiğimiz yazılım geliştirmeye benzemez. Zaten aralarındaki en büyük fark, ağların bir işi yamaları için eğitilmeye gereksinim duymalarıdır. Bildiğimiz gibi klasik programlama böyle değildir.

Bu eğitim işlevleri, değişkenlerin tanımlanması, çevrimlerin oluşturulması, şartların tespiti ve bunların testi ile derleyici üzerinde koşturma gerektirmezler. İşte bütün bunların yerine eğitme prosedürleri, seçim, analiz ve verilerin bir raya getirilmesi ile başlar. Bir YSA 'nın geliştirilmesindeki ilk ve en kritik adım budur.

Bir ağ gerekli ilişkileri oluştururken, ağı geliştiren kişi bunun farkına bile varmaz. Ağ bu ilişkileri bulup yapar ve eğer bir örnek sonuç çıkaracaksa bunu da bulup ortaya çıkarır. Oysa klasik programlamada kodun derlenmesi sırasında kaynak programın adım adım izlenmesi mümkündür, hatta yerine göre de gereklidir. Bir YSA verilerle uğraşırken yineleme çevrimi olarak bunların nasıl öğrenileceğinin kontrolünü yapmak için de bir takım parametrelere sahiptir. İşte bu parametreler , seçilen değerlerle iyi eğitilmiş bir ağın ne kadar etkili olacağını belirler. Deneyler yaparak uygun parametre değerlerinin bulunması, özellikle bazı algoritmaların çok fazla hesaplama zamanı gerektirmesi nedeniyle bu ağların geliştirilmesine engel olmuştur. Hatta öyle ki bir tek öğrenme işlevi bile saatlerce veya günlerce sürebilir. Bu yüzden paralel işlemcili mimarilerden yararlanma yoluna gidilmiştir.

YSA ' nı avantajlarını şu şekilde sıralayabiliriz:

1. YSA verilerden hareketle, bilinmeyen ilişkileri akıllıca hemen ortaya çıkarabilir. Bu özellikleri, uygulama açısından son derece önemlidir. Ayrıca veri toplama için bir ön sorgulama ya da açıklama gerekmiyor.

2. Ağlar genelleştirilebilir. Bir örnekten hareketle , diğer örneklerdeki benzerlikleri doğru olarak anlayabilirler. Daha önceden sistemin eğitilmiş olduğu konularda, eksik bilgi verilerek sonuç istenirse, sistem bunu geneller veya tamamlar. Bu özellik gerçekten önemlidir çünkü, gerçek dünyada bilgiler eksik veya kirlidir. Örneğin; sisli havada görülen bir cismin insan tarafından kısa bir uğraşla tanınabilmesi. Buradaki görüntü kirli (puslu) olmasına rağmen tanınabilmektedir.

3. YSA doğrusal değildir (None-Linear). Bu da bütün sonuçların herbir girdiyle etkileşim içerisinde olduğu manasına gelir. Doğrusal bir sistemde girdilerin biri değiştirildiğinde, çıktı

bu değişmeyle orantılı olarak değişir ve bu etki sadece değiştirilen girdinin değeri ile bağlantılıdır. Çünkü girdiler seri olarak işlenir. Doğrusal olmayan sistemlerde ise çıktılar bütün girdilere bağlıdır ve bir girdi değiştirildiğinde bu değişiklik çıktıda önemli bir değişikliğe sebep olmaz. Çünkü çıktı bütün girdilerin paralel işlenmesi ile elde edilir. Gerçek hayattaki sistemler genelde doğrusal değildir. Örneğin; ülke ekonomisindeki finansal durumlar, faiz oranları, fiyatlar, kurlar, borsa karmaşık bir etkileşim içerisindedirler. Dolar fiyatının artışı, aldığımız ekmeğin fiyatının değişmesinde veya faiz oranlarının artmasında etkili olabilir.

4. YSA son derece paralelliğe sahiptir. Bir çok benzer ve birbirlerinden bağımsız işlemler aynı anda simüle edilerek işlenebilir. Aynı zamanda, paralel bazı donanımlar bildiğimiz klasik mikro işlemlerden binlerce kez daha hızlıdır.

2. 5 Yapay Sinir Ağlarının Özellikleri:

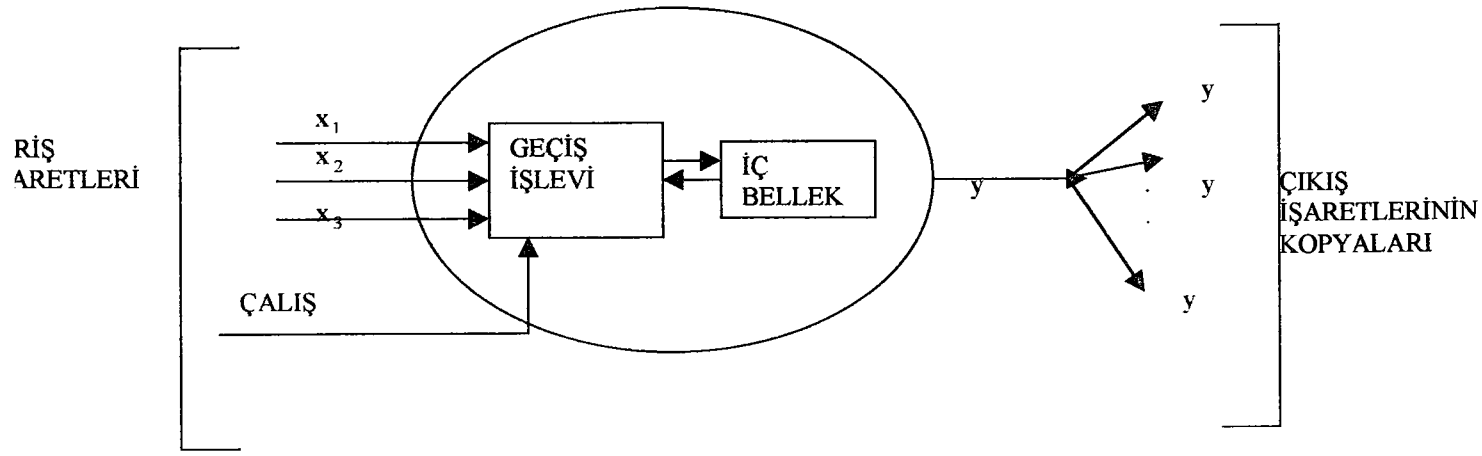
YSA 'nın alışılagelmiş bilgi işleme yöntemlerine göre üstünlükleri şu şekilde sıralanabilir:

Bellek ve Genelleme: YSA 'lar giriş bilgisini ağ içinde birçok yerel bellekler oluşturarak diğer nöronlara dağıtırlar. Nöronlar arası bağlantı ağırlıkları YSA 'nın o anki bilgi durumunu temsil eder. YSA 'nın bilgiyi bu şekilde saklaması en önemli özelliklerinden birisidir.

YSA 'da her nöronun bir yerel belleği vardır. Ağda çıkışı oluşturan giriş bilgisi, , ağın içinde birçok yerel bellek biçiminde dağıtılmıştır. YSA 'ları istenen çıkışı üretmek için bilgisayarlar gibi tam doğru girişlere ihtiyaç duymazlar.

Girişlerinde değişimler olsa bile doğru çıkışı üretebilirler. Yani sistem daha önceden tam o tipten bir şey görmemesine rağmen doğru çıkış üretme yeteneğine sahiptir.

YSA belleğinin yapısı; eksik, tam seçilemeyen bir giriş uygulandığında bile mantıklı çıkış üretmeye uygundur. Bu kurala " genelleme " adı verilir.



Şekil 2.1. Bir YSA işleme elemanının genel yapısı

Non Lineerlik ve Paralellik: YSA 'ları lineer değildirler. Bu özellikleri sayesinde daha karmaşık problemleri lineer tekniklerden daha doğru çözerler. Matematiksel olarak çözümün zor olduğu nonlineer davranışlar, YSA ile hissedilebilir, algılanabilir ve bilinebilir. YSA 'ları aynı zamanda son derece paralelliğe sahiptir. Alışılagelmiş bilgi işlem yöntemlerinin çoğu seri işlemlerden oluşmaktadır. Bu da özellikle hız problemlerine sebep olmaktadır. Mesela bilgisayarlar, beyine göre çok hızlı çalışmasına rağmen beynin toplam hızı bilgisayara göre kıyaslanamayacak kadar yüksektir. YSA' nda ise aynı katmandaki hücreler arasında zaman bağımlılığı yoktur. Bu yüzden eşzamanlı çalışabilirler. Başka bir deyişle bağımsız işlemleri aynı anda çok hızlı yürütebilirler. Bu da hızı çok arttırmaktadır.

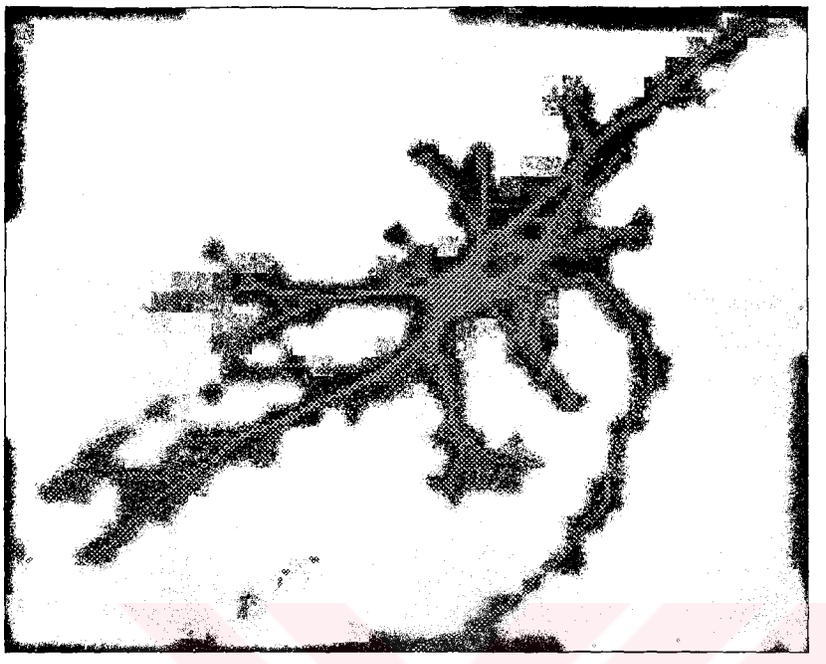
Hata Toleransı: YSA 'ında her işlem elemanı (nöron) kendi kendine yeterli olduğundan ve diğer nöronların içinde yapılan tüm işlemlere karşı kayıtsız kendi işini sürdürdüğünden hata toleranslıdırlar. YSA 'ları paralel dağılmış parametreli sistem olduğundan her bir işlem elemanı izole edilmiş ada gibi düşünülebilir. Herhangi bir işlem elemanının zarar görmesi sistem performansını düşürür. Fakat hiçbir zaman sistemi durma noktasına getirmez. Nedeni ise bilginin tek bir yerde saklanmayıp, yerel belleklere dağıtılmasıdır. Bunun aksine klasik hesap sistemlerinde, az bir zarar dahi sistemi durma noktasına getirir.

Gerçekleşme kolaylığı: YSA, karışık fonksiyonlar yerine basit işlemleri içerdiği için gerçeklemek kolaydır.

Yerel Bilgi işleme: YSA 'nda her bir işlem birimi, çözülecek problemin tümü ile ilgilenmek yerine, sadece problemin bir parçasıyla ilgilenmektedirler. Hücrelerin çok basit işlem yapmalarına rağmen, sağlanan görev paylaşımı sayesinde çok karmaşık ve zor problemler çözülebilmektedir.

Öğrenebilirlik: Alışılagelmiş veri işleme yöntemlerinin 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. YSA ise problemleri, verilen örneklerle çözer. Çözülecek problemler için yapı aynıdır.

2. 6 İnsan Beyni



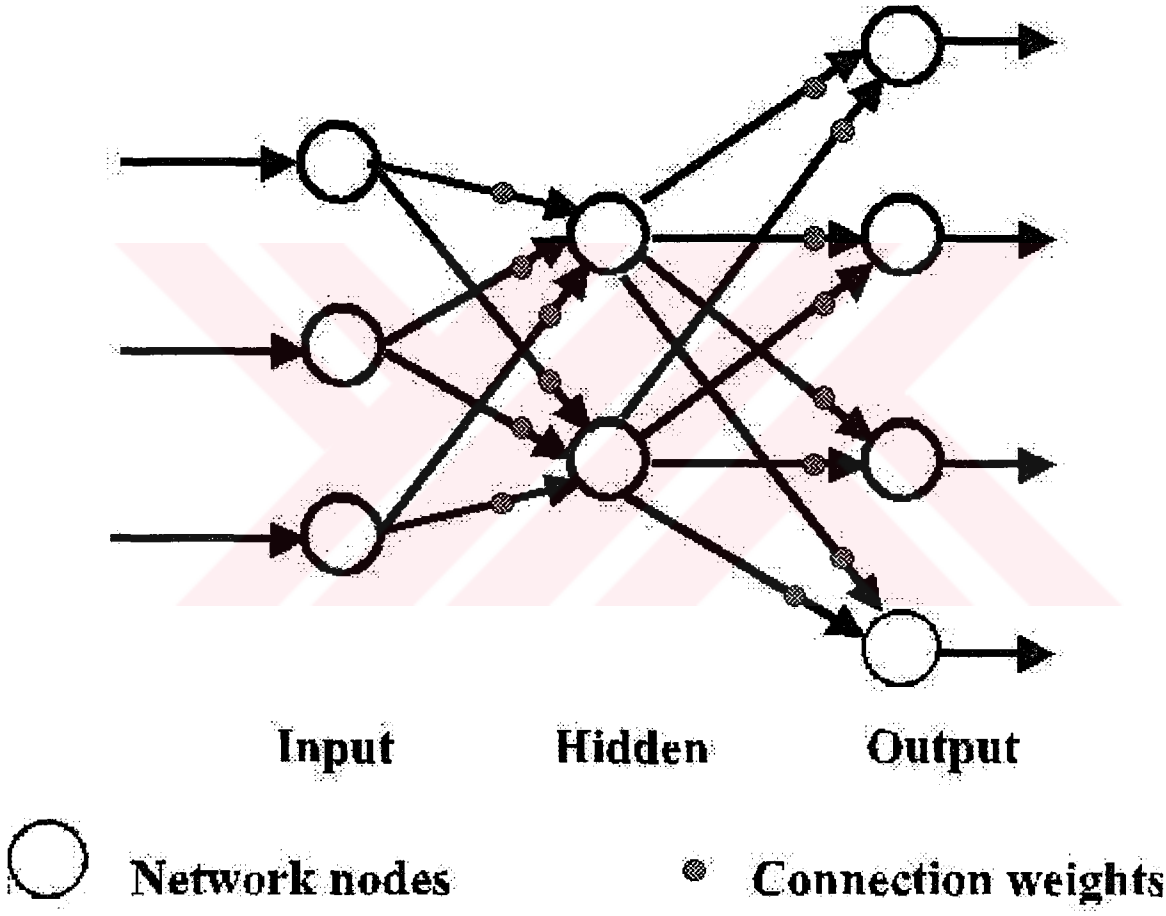
Şekil 2.2. İnsan beyni

Şimdi birazda insandaki sinir sistemine kısa bir göz atıp, beynin işleyişine bakalım. Beyin ve omurilik birlikte Merkezi Sinir Sistemini oluştururlar. Beyinden ve omurilikten çıkan , vücudun her yerine ulaşan sık ve beyaz sinir hücreleri de Peripheral (Çevresel) Sinir Sistemini oluşturur. Merkezi sinir sistemi iki bölümden oluşuyordu. 1) Beyin ; entegre, kurumsal, düşünce, davranış kontrolü. 2) Omurilik ; beyinden gelen ve beyine giden mesajları taşır, omurga refleks merkezidir.

Beyin canlıların en önemli organıdır. Bütün canlılar bu organ sayesinde vücutlarını kontrol eder ve karar verirler. Bu organ insanlarda çok daha gelişmiştir. Bir insan beyni yaklaşık 1,4 kg ağırlığındadır. Buna rağmen içerisinde milyarlarca sinir hücresi barındırırlar. Peki zeka neye bağlıdır ? Beyindeki sinir hücrelerinin fazlalığına veya beynin büyüklüğüne mi ? Bunun cevabı kesin olarak bilinmese de genel kanı, sinir hücreleri arasındaki bağların çokluğunun önemli rol oynadığıdır. Yani sinir hücreleri arasındaki bağlar (dentritler) ne kadar karışık ve çoksa, düşünce gücünün o denli fazla olduğu görüşü yaygındır. Ve bu bağlar beyin jimnastiği yaptıkça, düşündükçe gelişir. Aslında bu bağların her biri bilgiler arası bağlantıları ifade eder. Burası çok önemlidir çünkü bilgisayar ortamında da önemli bir noktayı teşkil eder. Beyinde bir diğer merak konusu da hafızadır. Bilgiler nerede depolanır ? Bunun da cevabı bilinmiyor. Bir görüş bilgilerin sinir hücrelerinde saklandığıdır, diğer bir görüş ise beynin özel bir hafıza

merkezi olduğudur. Fakat bu konudaki araştırmalar pek hızlı yürüyememektedir ve genelde bulgular teorik kalmaktadır. Bunun başlıca nedenlerinden bir canlı denek kullanılamamasıdır. Bilgisayarın araştırma alanına girişine kadar beyin hakkında bilinenler çok azdı. Daha sonra sinir hücreleri bilgisayarlar da simüle edilmeye başlandı ve duruma göre tepkileri kontrol edilmeye başlandı. Bir bakıma beynin simülasyonu yapıldı. Tabiki milyarlarca hücreyi simüle etmek şu anki teknoloji itibarı ile imkansızdır.

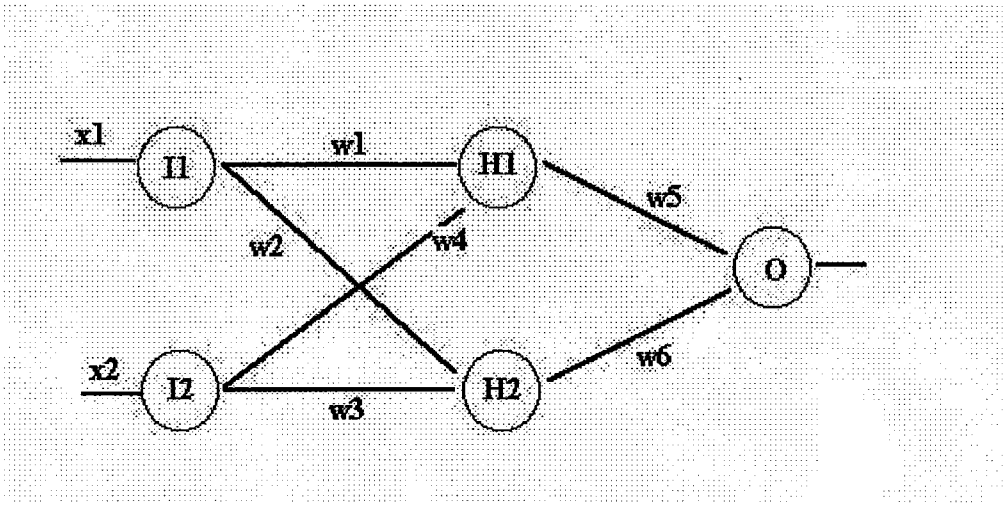
Peki sinirler bilgisayar ortamında nasıl uygulanır? Aslında kullanılan sistem tamamen algoritmalardan oluşmuştur ve mantık olarak sinir hücreleri örnek alındığından isim olarak bu seçilmiştir.



Şekil2.3.Yapay sinir ağı modeli

Yukarıda verilen şekilde her bir içi boş yuvarlak, bir sinir hücresini, girdiler dendritleri, çıktılar ise aksonları temsil etmektedir. Bu şekil genel olarak kullanılan bir şablondur. Şekilde görüldüğü gibi sistem 3 tabakadan oluşmuştur. İnsan beyninde ise bu katmanlar sayısız denecek kadar çoktur. 1. Tabaka giriş tabakasıdır ve girdi bilgileri buradan alınır ve sisteme verilir. 2. Tabaka ise gizli tabakadır ve kullanımı kullanıcıya kalmıştır. Bu tabaka, ilişkiler

arasındaki bağları daha karmaşık hale getirebilir. 3. Tabaka ise çıktı tabakasıdır ki girdiler işlenerek sonuçlar buradan alınır. Her bir yuvarlağın (sinirin) bir fonksiyonu ve eşik değeri vardır. İçi dolu ufak daireler ise bağlantı ağırlıklarını temsil eder. Konunun daha iyi anlaşılması için ufak bir örnek verelim. Aşağıda girilen iki bit'in XOR sonucunu veren bir sistem verilmiştir.



Xor, girilen iki bit üzerinde yapılan işlemdir ve girdiler göre çıktılar şöyledir.

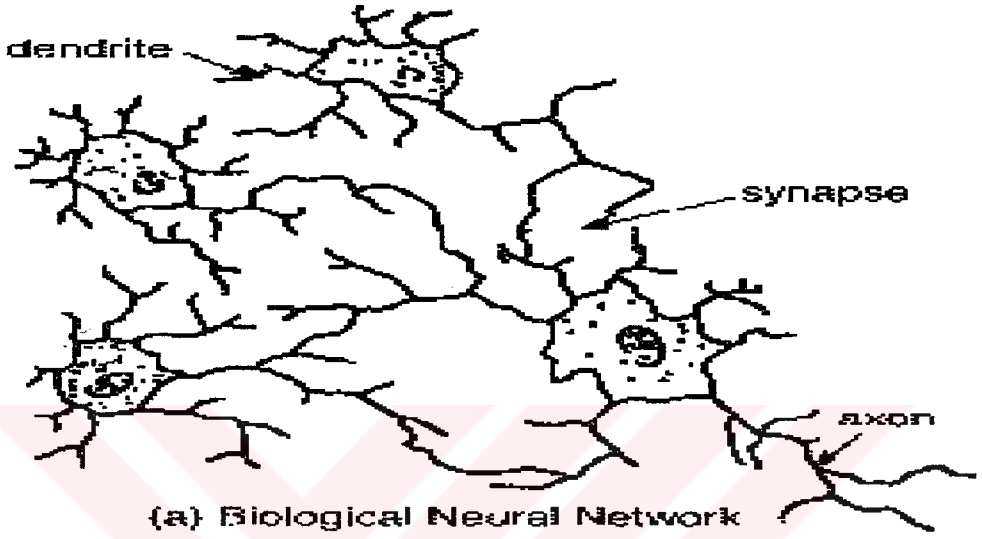
W1 = 1, W2 = -1, W3 = 1, W4 = -1, W5 = 1, W6 = 1 f(x) = { x>0 --> 1, x<=0 --> 0 }

X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Girdileri x1=0 , x2=1 olarak kabul edelim ve adım adım çözelim. İlk adımda I1=0, I2=1 olur. H1'e giren değerler şöyledir, w1*I1+w4*I2 = -1. H2'ye giren değerler şöyledir:
w2*0+w3*1 = 1. Şimdi gelen değerlerin toplamını fonksiyona koyalım. H1 için; f(-1)=0, H2 için; f(1)=1'dir. Artık O'yu hesaplayabiliriz. O=F(w5*0+w6*1) = f(1) = 1'dir. Ve çıktı 1 olmuştur.

2.7 Biyolojik Sinir Ağı Yapısı

İnsanın bilgi işleme sisteminin ana merkezi biyolojik beyindir. Sinir sisteminin temel yapı bloğu **nöron** dur. Yani vücudun çeşitli bölümlerinden ve bu bölümlere bilgi ileten hücredir. Yapay sinir ağları , basit işleme elemanlarının(nöronların) yoğun bir paralel dizisidir. Bir biyolojik nöronun şematik yapısı şekil 2.4 de gösterilmiştir



Şekil 2.4. Bir biyolojik sinir ağının basitleştirilmiş şematik yapısı

Dendrit'ler diğer hücrelerden bilgi alan nöronun girişleridir. **Aksonlar** ise diğer hücrelere bilgi taşıyan nöron çıkışlarıdır. Bir hücrenin aksonundan diğer bir hücrenin dendritine olan bağlantıya **sinaps** adı verilir. Bir sinir hücresi diğer sinir hücrelerine akson ve dentritlerle bağlıdır. Bilgi akışı elektrik sinyalleri olarak bu kanallardan iletilir.

Sinir sistemini oluşturan ve birbirlerine akson ve dentritlerle bağlı, aralarında elektriksel akış olan, birbiriyle etkileşim içinde olan, canlılarda bütün refleksel, hareketsel, algısal, düşünsel olayları sağlayan canlı hücrelere sinir hücreleri denir. İnsan sinir sistemi yaklaşık olarak 10^{12} (1 trilyon) nöron (sinir hücresi) içerir. Sinir sisteminin temel yapıtaşı olan nöronlar, çeşitli uyarılara kasılarak yanıt verirler.

Bir sinir hücresinin bir akson çıkışı vardır. Daha sonra bu aksondan yüzlerce bağlantı çıkabilir. Ve sinir hücrelerine giren yüzlerce dentritler vardır. Bu dentritlerin herbiri bir sinir hücresinden çıkan aksona bağlı kanallardır. Her akson diğer nöronlarla sinapslar vasıtasıyla bağlantı kurar.

Sinir hücrelerinin bilgi göndermesi üç çeşittir :

1) Elektriksel

2) Kimyasal

3) Mekanik.

Bir sinir hücresine giren sinapslar, o hücrenin özelliklerine göre işlenir ve tek bir çıktı (sinaps) aksondan gönderilir. Şunu unutmamak gerekir ki her bir sinir hücresine sayısız dentritler girebilir. Her bir sinir hücresi bir eşik değerine sahiptir. Buna Threshold değeri adı verilir ve Teta ile gösterilir. Eğer gelen sinapsların toplu bir işlemden geçirildikten sonraki sonuç değeri Threshold değerini aşabilirse, ulaşılan değer aksondan sinaps olarak gönderilir. Ayrıca her bir dentritin ağırlık değeri vardır. Gelen bilgi bu ağırlık değeri ile çarpılır (işlenir) ve önemine göre (ağırlık değerinin büyüklüğüne göre) yeni değeri oluşturulur ve hedef sinir hücresine yeni değeri ile girer. Yani Sinapslar nöronun girişine diğer nöronlardan gelen işaretleri ağırlık katsayıları ile çarparlar. Daha sonra nöron gelen bu işaretleri toplar. Nöronun aldığı ağırlıklı toplam , nöronun etkinlik seviyesini değiştirir. Nöral etkinlik , aynı zamanda nöronun ağ dışındaki kaynaklardan (Dış Girişler) uyarılması ile de arttırılabilir.

Burada dikkatimizi çekmesi gereken nokta, ilişkilerin derecelendirilmesidir. İki veya daha fazla bilgiler arasındaki bağlar ilişkilendirilirken, ağırlık değerleri belirlenir. Eğer sonradan bu önem derecesi değiştirilmesi gerekirse, bu değerler üzerinde oynama yapılır. Bilgisayar uygulamalarında ağırlık değerlerinin önemi büyüktür. Her bir sinir hücresi değişik eşik değerlerine sahiptir. Sinir hücrelerindeki bu değerler, kendi spesifik özelliklerine göre değişir. Örneğin; burundaki bir sinir hücresinin kokulara karşı olan hassasiyeti, gözdeki sinir hücrelerine göre daha hassas eşik değerine sahiptir. Veya göz aşırı bir ışığa maruz kalırsa, buradaki sinir hücreleri otomatik olarak ışığa karşı olan hassasiyetlerini azaltırlar yani gelen tepkiler büyük olsalar bile bu eşik değerinden geçmesin diye eşik değerlerini arttırırlar.

2. 8 İşlem Elemanları

Her işlem elemanı nispeten basit bir işi yerine getirir. Komşularından ve/veya dış kaynaklardan giriş alır ve bunu diğer elemanlara ileten bir çıkış işaretini hesaplamak için kullanılır. Bu işlemden ayrı olarak, ikinci bir görev, ağırlıkların ayarlanmasıdır. İşlem elemanlarından oluşan sistem bir çok elemanın hesaplamalarını aynı anda yürütebileceği için doğal olarak paraleldir.

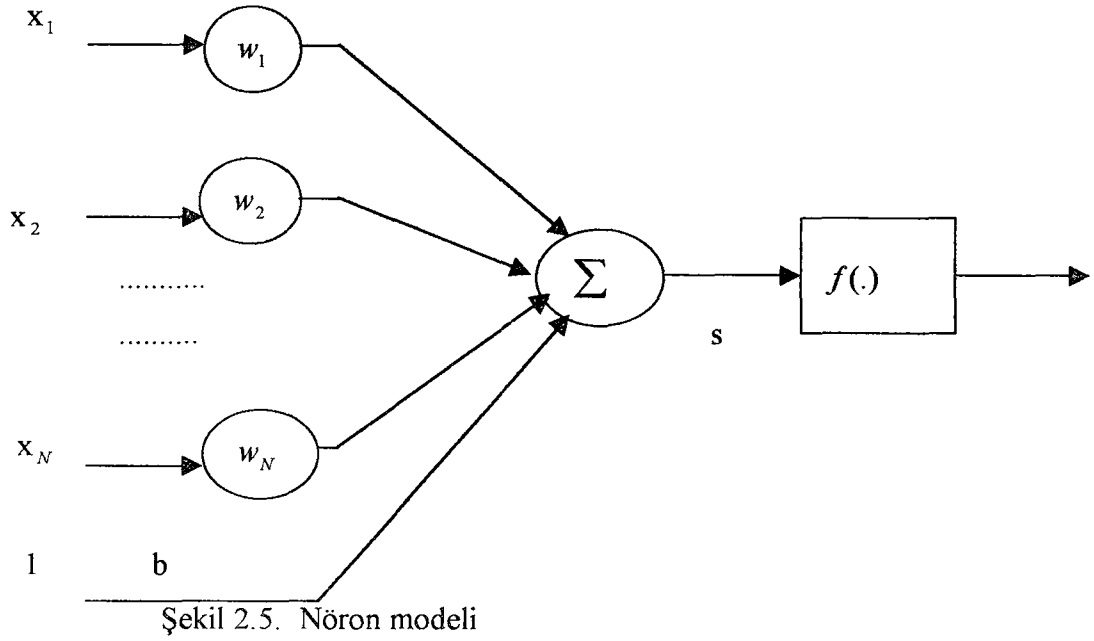
Nöral sistemler içinde üç tip eleman tanımlamanın yararı vardır; sistem dışından veri alabilen giriş elemanları, sistem dışına veri gönderen çıkış elemanları ve giriş ve çıkış işaretleri içinde kalan gizli elemanlar.

İşlem sırasında birimler eşzamanlı ve eşzamansız olarak yenilenebilirler. Eşzamanlı yenilemede bütün birimler aktivasyonlarını aynı anda yenilerler; eşzamansız yenilemede her eleman bir t zamanında aktivasyonunu yenilemek için bir olasılığa sahiptir.

2. 8. 1 İşlem elemanı (nöron)

İşlem elemanının çıkışı istenilen matematiksel tipte olabilir. Şekil 2.5 'te görüldüğü gibi işlem elemanına birden fazla giriş gelmekte ve sadece bir çıkış gitmektedir. Girişler dendritlere benzer şekilde diğer yapay hücrelerden bağlantılar vasıtasıyla işlem elemanına bilgi gelmesini sağlarlar. Bazı durumlarda bir işlem elemanı kendisine de bilgi geri gönderebilir. Bahsedilen bu bilgiler elemanlar arasında bulunan bağlantı hatları üzerinde depolanır. Her bağlantının bir ağırlığı vardır. Bu ağırlık bir işlem elemanının diğeri üzerindeki etkisini gösterir. Ağırlık büyüdükçe etki de büyür. Ağırlığın sıfır olması hiçbir etkinin olmaması, negatif olması ise etkinin ters yönde olması demektir. Bu ağırlıklar sabit olabildikleri gibi değişken de olabilirler. Toplama fonksiyonu, bir işlem elemanına gelen net girdiyi hesaplayan bir fonksiyondur. Net girdi genellikle gelen bilgilerin ilgili bağlantıların ağırlıkları ile çarpılıp toplanması ile belirlenir. Eşik fonksiyonu da , toplama fonksiyonu tarafından belirlenen net girdiyi alarak , işlem elemanının çıkışını belirleyen fonksiyondur. Genel olarak türevi alınabilen bir fonksiyon olması tercih edilir. İşlem elemanının çıkış ünitesi çıkış fonksiyonunun ürettiği işareti diğer işlem elemanlarına veya dış dünyaya aktarma işlevini yapar. İşlem elemanları ağın topolojik yapısına bağlı olarak tamamen birbirinden bağımsız ve paralel olarak çalışabilirler.

2. 8. 2 Nöron modeli



Şekil 2.5. Nöron modeli

w_i, x_i girişine ilişkin ağırlıktır. b ise sabit girişin ağırlığıdır ve eşik olarak adlandırılır. s ağırlıklı toplam olmak üzere ifadesi şu şekilde yazılabilir:

$$s = \sum_{i=1}^{N} w_i x_i + b \tag{2.1}$$

Eğer aşağıdaki tanımlar yapılırsa;

$$w = [w_1 \;\; w_2 \;\; \ldots \;\; w_M \;]^T \tag{2.2}$$

$$x = [x_1 \;\; x_2 \;\; \ldots \;\; x_N \;]^T \tag{2.3}$$

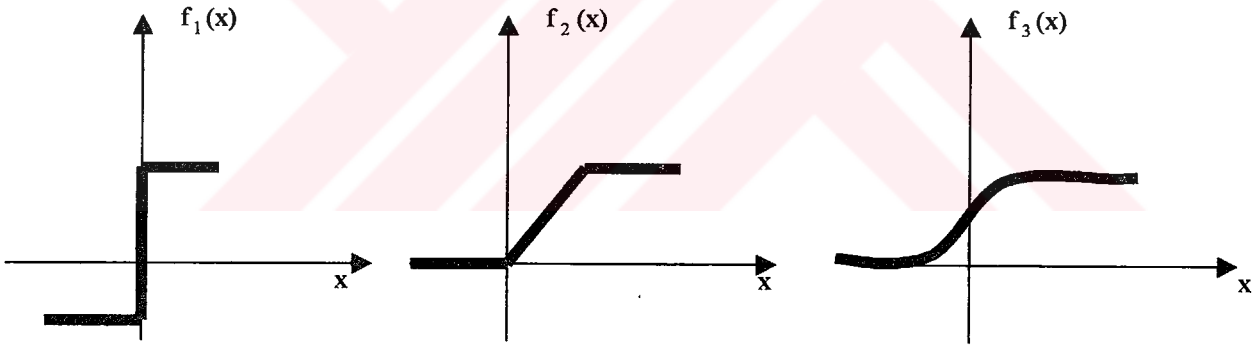
ağırlıklı toplayıcının ifadesi ve çıkış şu şekilde yazılabilir:

$$s = w^T x + b \tag{2.4}$$

$$y = f(s) \tag{2.5}$$

Buradaki f(.) lineer olmayan bir fonksiyondur.

Aşağıda f fonksiyonu için kabul görmüş 3 tip lineer olmayan fonksiyon gösterilmiştir.



Katı sınırlamalı	Eşik lojiği	Sigmoid
(Hard Limiter)	(Treshhold Logic)	(Sigmoid)
(a)	(b)	(c)

Şekil2.6. Nöron modellerinde kullanılan 3 tip lineer olmayan fonksiyon

1) Hard Limiter: Eğer işlenecek değer (x) (x>0 ve x <=1) şartını sağlıyorsa 1, (x=0) şartını sağlıyorsa 0, (x<0 ve x>=-1) şartını sağlıyorsa –1 döndürür.

2)Treshold Logic : Eğer işlenecek değer (x) (x=0) şartını sağlıyorsa 0, (x>0) şartını sağlıyorsa x değerini döndürür.

3) Sigmoid : İşlenecek değer (x), belirlenmiş sigmoid fonksiyonuna göre değer döndürür. Şekil 2. 8. 2. 1 de görüldüğü gibi yapay nöron modeli n tane ağırlıklı girişi toplamakta, eğer toplam eşik değerini geçiyorsa sonucu lineer olmayan fonksiyon tipi tarafından geçirmektedir. Nöron modeli, nöronun eşik değeri ve kullanılan lineer olmayan fonksiyon tipi tarafından belirlenmektedir. (Yaygın olarak kullanılan transfer veya işaret fonksiyonları diye adlandırılan eşik fonksiyonları sigmoit'dir.)

Logaritmik sigmoid fonksiyonu :

$$f(s) = \frac{1}{1+e^{-\lambda s}} \tag{2.6}$$

Tanjant sigmoid fonksiyonu :

$$f(s) = \frac{1-e^{-\lambda s}}{1+e^{-\lambda s}} \tag{2.7}$$

λ, fonksiyonun eğimini belirler ve genel olarak 1 veya 2 alınır. Tanjant sigmoid fonksiyonunda λ olarak 2 alınırsa , bu fonksiyon, tanh fonksiyonuna karşılık düşer.

2. 9 Elemanlar Arası Bağlantılar:

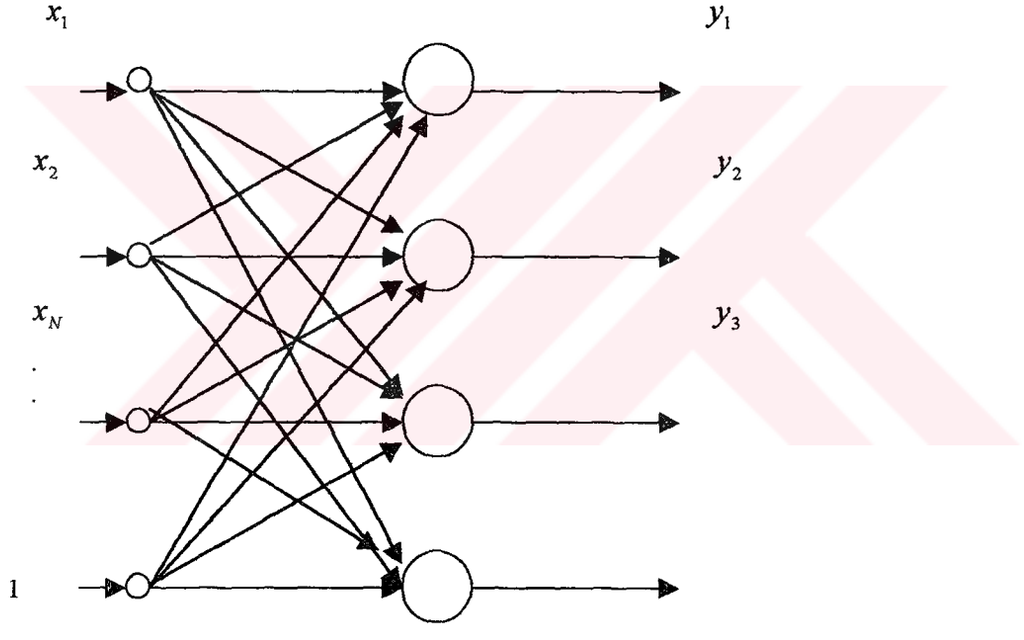
Genelde her bağlantı, i elemanının işaretinin j elemanı üzerindeki etkisini belirleyen bir w_{ij} ağırlığı ile tanımlanır. Her bağlantı anlık tek yönlü işaret iletim yolu olarak çalışır. Her işlem elemanı , herhangi bir sayıda giriş bağlantısına sahip olabilir. Bir işlem elemanından dışarı giden herhangi sayıda bağlantı olabilir, ancak bu çıkışlardan hepsi birbirinin aynı işareti taşımalıdır. İşlem elemanının çıkış işaretinde ayrıca dış giriş diye adlandırabileceğimiz bir θ bağlantısı mevcuttur.

3.1 YAPAY SİNİR AĞLARININ YAPISI

3.1.1 İleri beslemeli yapay sinir ağları:

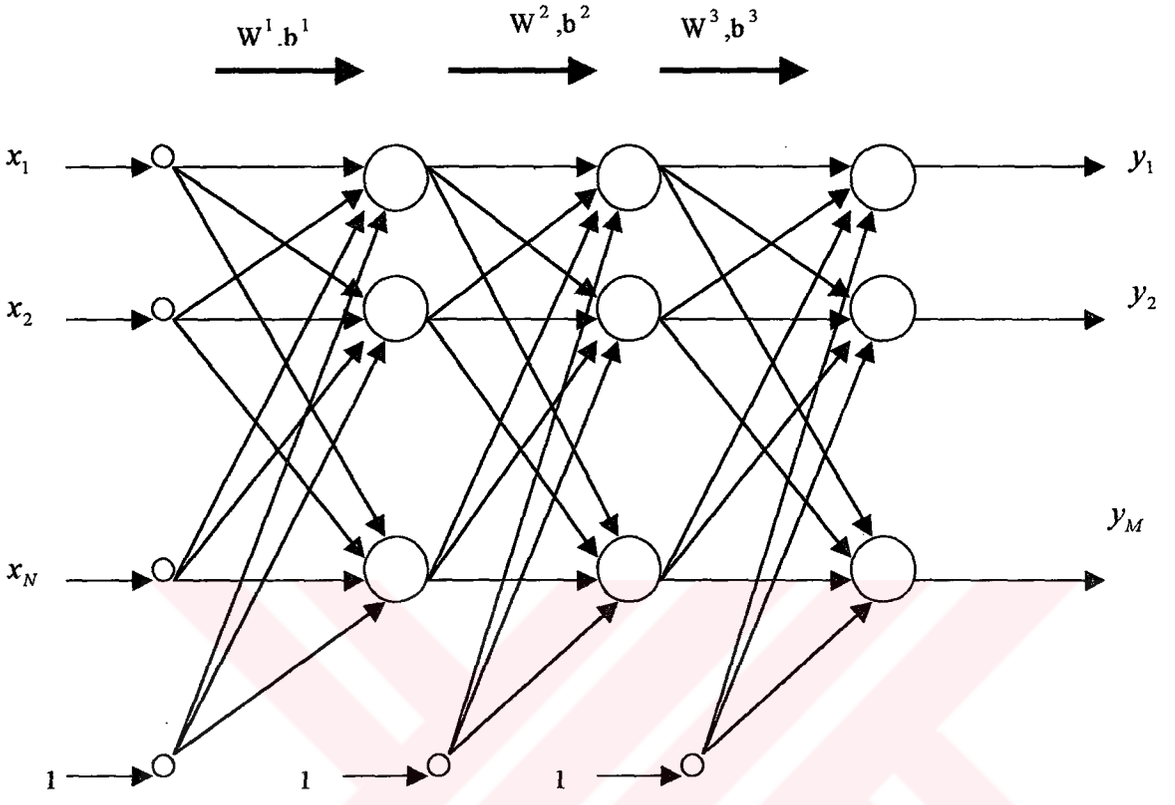
En genel halde giriş tabakası, saklı tabaka ve çıkış tabakası olmak üzere üç tabakalı bir yapıya sahiptirler. Bu tür ağ yapısında nöronlar arka arkaya beslenirler. Öğrenme aşamasında girdi modelleri ağın giriş terminallerine sunulur. Birinci tabakadaki nöronlar çıktılarını hesaplarlar bir sonraki tabakaya girdi değeri olarak gönderirler. Sırasıyla her bir tabaka aynı işlemi yapar. En uç tabakanın çıktı değerleri de işlemi sonuçlandırır.

3. 1. 1. 1 Tek katmanlı ileri beslemeli ağlar:



Şekil 3.1 : Tek katmanlı ileri beslemeli ağ

3. 1. 1. 2 Çok katmanlı ileri beslemeli ağ



Şekil: 3.2. 3 katmanlı ileri beslemeli bir ağ

Şekil 3.2 de ; 3 katmanlı ileri beslemeli bir ağ verilmiştir. Katmanların bağlanmasından oluşur. Bir katmanın girişi , bir önceki katmanın çıkışıdır . Burada giriş katmanı, sadece girişi çoğullamaktadır. Yani girişi ile çıkışları aynıdır. Giriş katmanı ile çıkış katmanı arasında kalan katmanlara saklı katmanlar adı verilmektedir. Genelde bir veya iki saklı katman kullanmak yeterli olmaktadır. Toplam katman sayısı verilirken giriş katmanı dahil edilmez. Saklı katmanlardaki hücre sayıları farklı olabilir.

W^i, i. Katmanın ağırlık matrisi, b^i , i. Katmanın eşik vektörü ve $\Gamma^i[.]$, i. Katmanın matris operatörü olmak üzere 3 katmanlı bir ağın çıkışı şu şekilde yazılabilir:

$$y = \Gamma^3[W^3\Gamma^2[W^2\Gamma^1[W^1x + b^1] + b^2] + b^3] \qquad (3.1)$$

Çok katmanlı ileri beslemeli ağlar, çıkış fonksiyonu olarak sigmoid fonksiyonları kullanırsa, çok katmanlı genlikte sürekli algılayıcı, sign fonksiyonu kullanırsa da çok katmanlı algılayıcı olarak adlandırılır.

3. 1.2 Geri beslemeli ağlar

Geri beslemeli ağlarda, bir katmandaki hücreler, sadece bir önceki katmanın çıkışlarını değil, aynı zamanda o katmanın veya sonraki katmanların da çıkışlarını giriş olarak alabilirler. Bu yüzden çıkış, sadece o anki girişlere değil, başlangıç durumuna ve önceki girişlere de bağlıdır. Bu tür ağlarda diğerlerinin aksine, tabakalar arasındaki bağlantıya ilave olarak tabakadaki her bir nöron da birbirleriyle bağlantılıdır. En popüler geri beslemeli ağ tiplerinden biri olan Kohonen (1987) ağları, kendi kendini organize edebilen, kullanımı zor olmasına karşın çok güçlü ve hızlıdırlar. Denetlenmemiş öğrenme kullanırlar. Hopfield (1985) ağları ise ağın enerjisini minimize eder ve bu durumda ağda meydana gelen değişiklikleri analiz ederek ağırlıkları adapte eder. Bu yüzden daha çok optimizasyon problemlerinde kullanılırlar.
Dinamik ağlardan en çok bilinenler, Hopfield ağı, Grossberg ağı dır.

3. 1. 3 Geri yayılma yapay sinir ağı:

Geri yayılma terimi gerçekte yapay sinir ağları için özel bir öğrenme kuralıdır. Ancak, genellikle geri yayılma algoritması kullanılan sinir ağının mimarisi olarak algılanır. Geri yayılmalı öğrenme kuralı ağ çıkışındaki mevcut hata düzeyine göre her bir tabakadaki ağırlıkları yeniden hesaplamak için kullanılmaktadır. Geri yayılmalı yapay sinir ağlarında aynı tabakadaki nöronlar arasında bağlantı mevcut değildir. Ancak, tabakadaki her bir nöron bir ileri tabakadaki her bir nörona ayrı ayrı bağlıdır ve bunların giriş değerlerini verir. Bu tür yapay sinir ağları denetimli öğrenme kuralını kullanırlar. Buna göre hem giriş hem de çıkış verilerinin mutlaka bilinmesi gerekir. Bu konu daha sonra anlatılacaktır.

3. 2 Yapay Sinir Ağlarında Öğrenme, Eğitme ve Ağırlık Uzayı:

Yapay Sinir ağı belirli bir probleme göre programlanmadığı halde o problemi çözmeyi öğrenebilir. Eğitme ve öğrenme hemen hemen YSA ların temelidir. YSA 'nda değişebilen sistem parametreleri, hücreler arası bağlantıları sağlayan sinapsları temsil eden ağırlıklarıdır.
Öğrenme, ağdaki nöronların değiştirilmesi ile değil, nöronlar arasındaki bağlantı ağırlıklarının değiştirilmesi ile sağlanır.
Tek bir nöronun çıkışının nasıl belirlendiğini göz önüne alalım. Nöronun transfer fonksiyonunun sabit olması koşulu altında çıkışını yalnızca iki şey belirler; gelen işaret ve

nörona giriş bağlantı ağırlıkları. Nöronun gelen işarete doğru yanıt vermesinde ve performansının arttırılmasında en önemli eleman bağlantı ağırlıklarıdır.

Öğrenme için bir eğitim kümesine ihtiyaç vardır. Eğitim kümesi, YSA 'na öğrenme için uygulanacak girişlerden ve varsa istenen çıkışlardan oluşmaktadır. Her adımda eğitim kümesinden sırayla veya rastgele örnekler alınmakta ve ağın bu girişlere ilişkin çıkışı bulunmaktadır. Adım sonunda veya tüm örnekler bir defa ağa uygulandıktan sonra (çevrim sonunda) belirli bir amaç ölçütü uyarınca ağın bağlantı ağırlıkları değiştirilmektedir.

Öğrenme, eğer ağırlıklar, adım sonunda değiştiriliyorsa, veri uyarlamalı öğrenme, çevrim sonunda değiştiriliyorsa grup uyarlamalı öğrenme (adaptive learning) olarak adlandırılır.

Eğitme ve öğrenme aynı şeyler değillerdir. Eğitme, ağın öğrenmesi için bir işlemdir. Öğrenme ise eğitme işleminin neticesidir. Eğitme ağına dışardan bir etkidir. Öğrenme ise ağın içsel aktivitesidir.

Günümüzde kullanılan bir çok öğrenme kuralı vardır. Bilinen en çok kullanılan öğrenme kuralları şunlardır:

1- Raslantısal (Hebb) öğrenme kuralı
2- Performans (Widrow ve Adaline) öğrenme kuralı
3- Kompetatif (Kohonen) öğrenme
4- Filtreleme (Grossberg)
5- Genelleştirilmiş Delta Kuralı Öğrenme (Geriye yayılmalı öğrenme kuralı)

YSA ' nı örneklerle eğitilmesi 3 ayrı yol ile gerçekleştirilebilir:

1- Öğreticili Eğitme (Supervised Training)
2- Skor ile Eğitme (graded training)
3- Kendini düzenleme ile eğitme (self- organization training)

Birincisi ve en geneli öğreticili eğitmedir. (supervised training): Bu teknikle ağ, çevreden aldığı girişlere karşı, bir eğiticinin belirttiği doğru çıkışı üretmek hedefiyle kendini ayarlamaktadır. Ağa hem girişler hem de bu girişlere ilişkin istenen çıkışlar verilmektedir. Ağ tarafından üretilen gerçek çıkış ile istenen çıkış arsındaki fark, bağlantı ağırlık katsayılarının ne kadar değişeceğini belirlemektedir. Her denemeden sonra ağ kendi çıkışını doğru cevaplarla karşılaştırır ve çıkış hatası kabul edilebilecek seviyeye ininceye kadar ağırlıklarını değiştirerek iterasyona devam eder.

İkincisi, skor ile eğitmedir(graded training): Öğreticili eğitimin benzeridir. Ancak bu metodda giriş işaretine karşılık istenen çıkış işaretleri verilmez. Bunu yerine ağ çıkışının durumuna göre , " başarılısın" , "başarısızsın" veya " çok yüksek", "çok düşük" gibi değerlendirmelerle ağ performansı ile ilgili geri besleme bilgisi verilir.

Üçüncü metod ise kendini düzenleme ile eğitme (self- organization training) dir. Bu yöntemde ise ağ ' a giriş örnekleri verilir, istenen çıkışlar ağa verilmez ve performansla ilgili geri besleme bilgisi verilmez. Ağ giriş işaretine göre kendisini organize eder. YSA 'nın bağlantı ağırlık katsayıları, oluşan hatayı göz önüne almaksızın bir amaç ölçütü uyarınca sadece girişlere bağlı olarak değişmektedir.

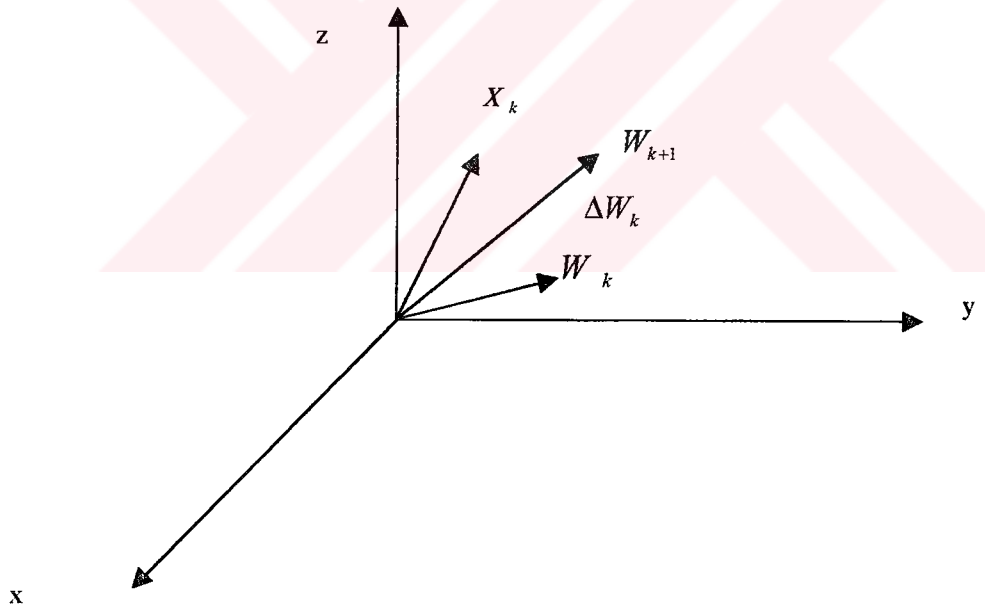
YSA her üç yöntemde de programlama yerine örnekler ile eğitilir. Ağ ağırlık maddesindeki değerler değiştirilerek örneklere göre eğitilir ve herhangi bir örnek ile yeniden karşılaştırıldığında uygun cevabı üretir. Öğrenmede iyi bir model kullanıp, ağırlıkların bu modele göre değiştirilmesi esastır.

Her bir işlem elemanının " n " adet gerçek ağırlığı ve " N " adet işlem elemanı (nöronu) olduğu gözönüne alınırsa ;

$$W = (W_{11}, W_{12}, ..., W_{1n}, W_{21}, ...W_{2n}, ..., W_{N1}, W_{N2}, ..., W_{Nn})^T \quad (3.2)$$

$$W = (W_1^T, W_2^T, ..., W_n^T) \quad (3.3)$$

Burada $W_1, W_2, ..., W_N$: İşlem elemanlarının (nöronların) ağırlık vektörleridir. Aşağıdaki şekilde eğitim süresince ağırlıkların düzeltiminin vektörel çizimi verilmiştir.



Şekil3.3. Ağırlık vektörünün değişimi

Şekilde;

X_k : Giriş işareti vektörü

W_{k+1} : Gelecekteki ağırlık vektörü

ΔW_k : Ağırlık vektörünün değişimi

W_k : Şimdiki ağırlık vektörüdür.

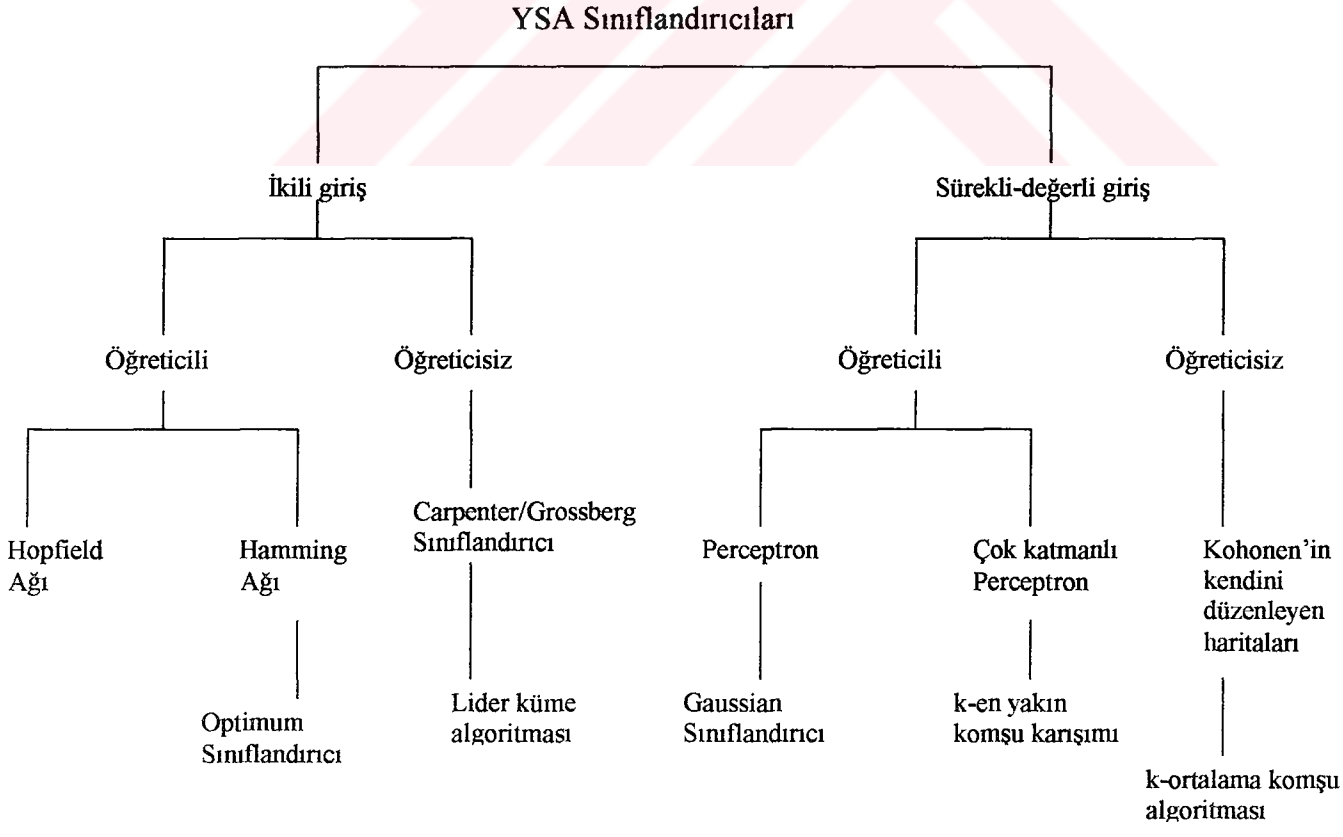
Ağırlık vektörü ile çalışan YSA ' da önemli noktalardan birisi, bir öğrenme kuralı geliştirip, ağırlık vektörü "W" yı istenilen ağ performansı verecek noktaya yöneltmektir. Genellikle öğrenme kuralı için bir performans veya maliyet fonksiyonu tanımlanır. Ağırlık değiştirme denklemi , ağ ' daki hataların karelerinin toplamını minimize edecek şekilde düzenlenir.

3. 3 Yapay Sinir Ağının Sınıflandırılması

Bir yapay sinir ağından beklenen görev, gerçek dünyadaki nesnelerle biyolojik sinir ağının yaptığı işlevi , benzer bir yolla yerine getirmesidir. Yapay nöron ağın giriş veri tipleri ikili (binary) 0 – 1 veya sürekli değerlerdir. Sürekli değer girişlerinde giriş bilgi normalize edilir. Bir işlem elemanına gelen girişler matematiksel tiplerine göre etiketlendirilerek sınıflandırılır.

3.4 Yapay Sinir Ağında Ağ Modelleri

YSA, giriş veri tiplerine göre ikili giriş (0, 1) ve sürekli değerli giriş olmak üzere şekildeki gibi sınıflandırılır:



Şekil3.4 Ağ modeli

Yapay sinir ağında sabit desenler için sınıflama metodları da ikili sayı sistemindeki girdilere ve sürekli değerlendirilen girdilere göredir. Yani;

1. İkili sayı sistemindeki girdiler
   a) Hopfield Net
   b) Hamming Net
   c) Carpenter/Grossberg Sınıflandırıcısı
2. Sürekli değerlendirilen girdiler
   a) Perceptron
   b) Çok katmanlı Perceptron
   c) Kohonen'in kendini düzenleyen haritaları.

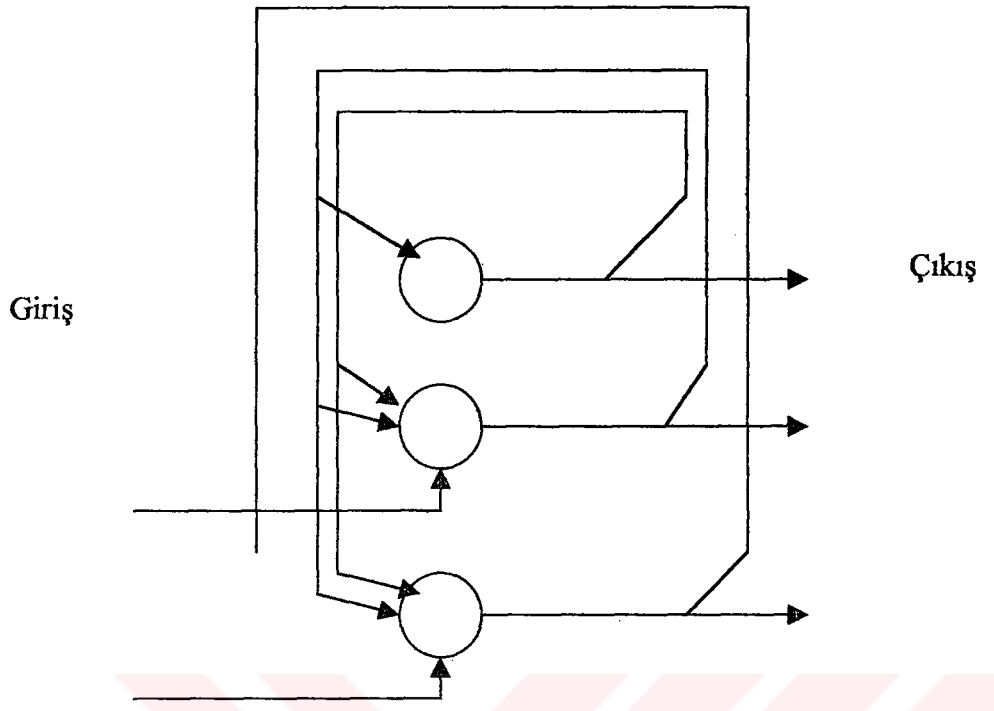
Hopfield Net, Hamming Net ve Carpenter Grossberg ikili sayı sisteminde olan girdilere sahip problemler için kullanılırlar. Şimdi bu metodlara kısaca değinelim. Sürekli değerlendirilen girdiler için kullanılan metodlara da daha sonra değinilecektir.

3. 4. 1 Hopfield ağı

Yapay nöral ağlara olan ilgi 1982' de John Hopfield tarafından yayınlanan bir makale ile artmıştır. Hopfield 'ın çalışması nöral ağ yaklaşımının bazı ilginç matematiksel özellikler içerdiğini göstermiştir.

Hopfield ağı normal olarak ikili girişler ile kullanılır. Bu ağın en kullanışlı olduğu durumlar, girişin tam doğru olarak ikili tabanda gösteriliminin mümkün olduğu durumlardır. Yani giriş elemanları, siyah ve beyaz noktalardan oluşan resimlerin pixel elemanlarını veya her karakterin 8-bit ASCII gösterilimindeki bitleri temsil edebilirler.

Hopfield kendi ağının değişik düzenlemeleri üzerinde yaptığı kapsamlı çalışmaları ile de nöral ağlara olan ilgiyi tazelemiştir. Hopfield ağı bir içerikle adreslenebilir bellek olarak veya optimizasyon problemlerinin çözümünde kullanılabilir.



Şekil 3.5. Bir içerikle adreslenebilir bellek olarak kullanılabilecek Hopfield ağı

Şekil3.5 'te gösterilen Hopfield ağında, katı sınırlamalı lineer olmayan fonksiyona sahip N tane hesapsal birim vardır. Bu ağın ikili giriş ve çıkışları –1 ve +1 değerlerini alabilmektedirler. Her birimin çıkışı, diğerlerine w_{ij} olarak gösterilen ağırlıklar üzerinden geribesleme olarak gönderilmiştir. Ağın çalışması aşağıdaki tabloda açıklanmıştır. Önce her sınıf için verilen örnek bilgilerden yararlanarak , verilen formüle göre ağırlıklar ayarlanır. Daha sonra sıfır anında ağa hangi sınıfa ait olduğu bilinmeyen bir bilgi girilir. Böylece ağ çıkışları bu bilinmeyen bilgi ile uyuşmak üzere zorlanır. Bu ilk koşullamadan sonra, ağ verilen formülü kullanarak ayrık zaman adımları ile iterasyon yapar. Ardıl çıkışlar aynı olduğu zaman, ağın yakınsadığı kabul edilir. Yakınsamadan sonra nöron çıkışlarındaki bilgi ağ çıkışıdır.

Hopfield ve diğerleri Tablodaki denklemleri kullanarak nöron çıkışları güncellendiğinde ve ağırlıklar simetrik olduğunda ($w_{ij} = w_{ji}$) bu ağın yakınsadığını göstermişlerdir. Roska, ağırlık matrisinin simetrik olmaması durumunda, ağırlık matrisinin bazı şartları sağlaması halinde ağın yakınsayacağını göstermişlerdir. Hopfield ağı bir içerikle adreslenebilir bellek olarak kullanıldığında, yakınsamadan sonra ağ çıkışı, tam saklanan bellek bilgisi olarak kullanılır. Hopfield ağı bir sınıflayıcı olarak kullanıldığında, yakınsamadan sonra çıkış, herhangi bir

örnek ile uyuşup uyuşmadığının belirlenmesi için M örnek ile karşılaştırılmalıdır. Eğer uyuşursa, çıkış, örneği çıkış bilgisi ile uyuşan sınıf olacaktır. Eğer uyuşma yok ise, bir " uyuşma yok " sonucu oluşacaktır.

3.4. 1. 1 Hopfield ağı algoritması:

Adım1 : Bağlantı ağırlıklarını belirle

$$w_{ij} = \begin{cases} \sum_{s=0}^{M-1} c_i^s . c_j^s & , i \neq j \\ 0 & , i = j \end{cases} \qquad 0 \leq i, j \leq N-1 \tag{3.4}$$

Burada w_{ij}, i. nci elemandan j. nci elemana olan bağlantı ağırlığı; +1 veya –1 değerlerini alabilecek c_i^s ise s. nci sınıfın örneğinin i. nci elemanıdır.

Adım2 : Verilen Yeni Girdiye göre Node ayarlarını yap

$$y_i(0) = c_i \quad , \quad 0 \leq i \leq N-1 \tag{3.5}$$

Başlangıç değerleri nodelara yerleştir. Bu formülde $y_i(t)$, i. nci elemanın t anındaki çıkışını, +1 veya –1 değerlerini alabilecek c_i ise giriş bilgisinin i. nci elemanını göstermektedir.

Adım3: Yakınsayana dek iterasyon yap

$$y_i(t+1) = f_k\left[\sum_{i=0}^{N-1} w_{ij} . y_i(t)\right] \quad , \quad 0 \leq j \leq N-1 \tag{3.6}$$

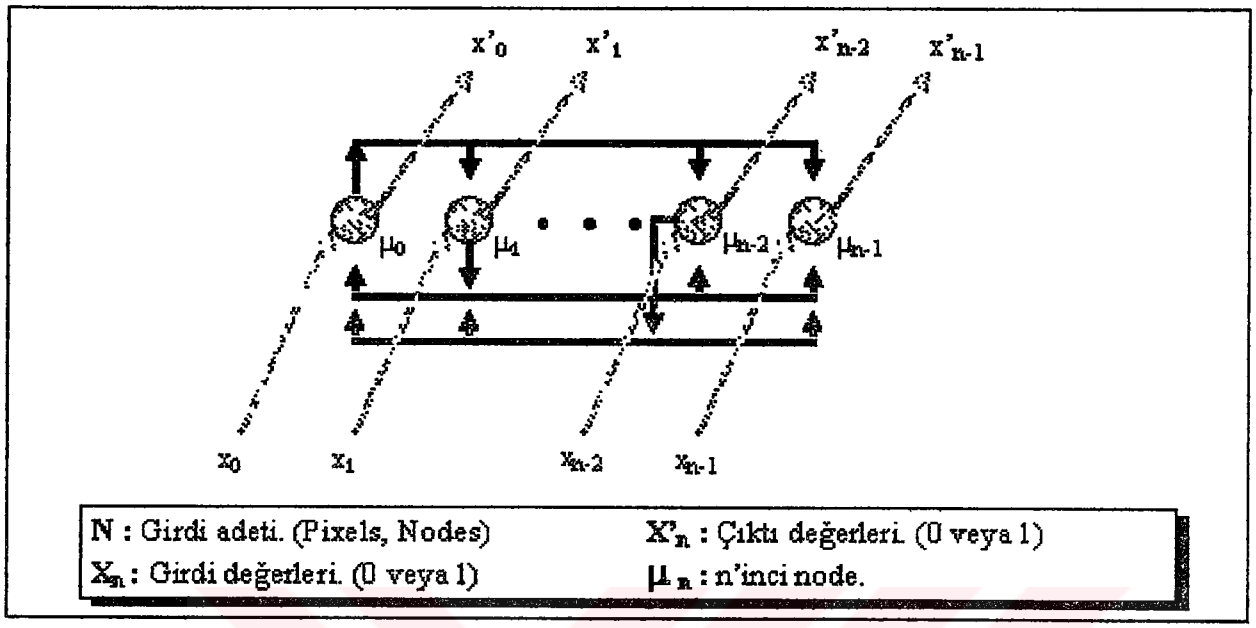
Burada f_k fonksiyonu +1 ve –1 değerleri arasında katı sınırlamalı bir fonksiyondur. Bu işlem , bir önceki çıktı değerleri ile şu anki çıktı değerlerinin eşit olmasına kadar yapılır.

$$f_k(x) = \begin{cases} 1 & , 0 < x \leq 1 \\ 0 & , x = 0 \\ -1 & , 0 > x \geq -1 \end{cases} \tag{3.7}$$

Adım 4: Sonuç elde edildikten sonra bir dahaki deneme için 2. Adıma git.

Aşağıdaki şekildeki Hopfield Net modelinde içi dolu yuvarlakların herbiri bir node'u temsil etmektedir. Her bir node'a girdi adetin-1 tanedir. Bir node'dan çıkan sonuç diğer nodelara

girdi olarak geri döner. Girdiler 0 zamanda yani aynı anda uygulanır ve çıktılar x'_n lerinden alınır.



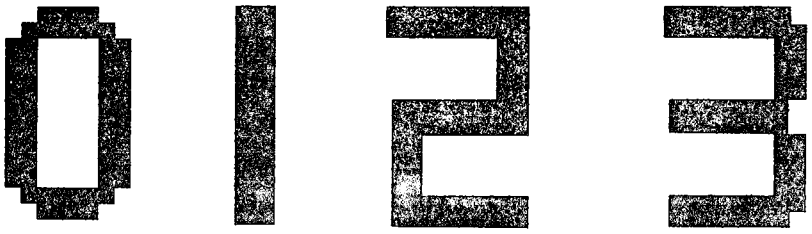
Şekil 3.6. Hopfield net modeli

Şekil 3.6 da içi dolu yuvarlakların herbiri bir nodu temsil etmektedir. Her bir noda girdi adeti n-1 tanedir. Bir noddan çıkan sonuç diğer nodlara girdi olarak geri döner. Girdiler 0 zamanda yani aynı anda uygulanır ve çıktılar x'_n lerinden alınır.

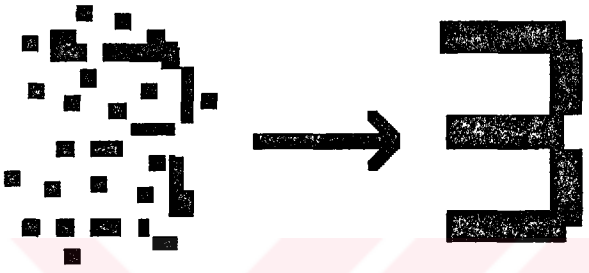
Hopfield Net 'in çalışmasını harf tanıma örneği ile açıklayalım:

Bu metodda hazırlanan sinir ağı belirli konular üzerinde önceden eğitilmeli yani sisteme baz alınacak değerler öğretilmelidir. Eğer karakter tanıtma ile uğraşıyorsak, ilk olarak karakterler sisteme verilmeli ve bu şekilde eğitilmelidir.

Aşağıdaki karakterler ile eğitilmiş olan bir sistemin var olduğunu varsayalım. Bu karakterleri 10 x 12 büyüklüğünde kabul edersek 120 noktaya sahibiz demektir ki bu da 120 noda ihtiyacımız var demektir.



Aşağıdaki gibi kirli (noisy) bir şekili Hopfield Net'e verirsek, 8 adım sonunda gerçek sayı bize çıktı olarak verilir. İterasyon ilerledikçe çıkış, örnek şekle daha çok benzemiş ve 8. İterasyonda 3 sayısı için verilen örnek şekle yakınsamıştır. Bilgisayar benzetimleri eğer birkaç bağlantı kopmuş olsa bile, belleğin çalışmaya devam ettiğini göstermiştir. Bu sonuç, bellekteki kodlanmış bilginin ağırlık matrisi üzerinde saklanmış olmasından dolayıdır.
Böylece hiçbir sinaptik bağlantı kritik olmaz. Her dakika beyinde bazı nöronların ölmesine rağmen, beynin çalışmasına devam ettiği düşünülürse bu davranış insan beyninin çalışması ile uyumludur.



Yukarıdaki şekilde bozuk şekil, 0. 25 ihtimalle orjinal 3 karakterinden bozularak elde edilmiştir. Bozuk şekil verildikten sonra 0 zamanında çıktı gerçek karakter gibi olur. Düzgün karakter, bu sistemde 8 hamleden (düzeltmeden) sonra bulunur
Hopfield ağının, bir içerikle adreslenebilir bellek olarak kullanıldığında, iki temel sınırlaması vardır. Öncelikle saklanabilecek ve doğru bir şekilde hatırlanabilecek bilgilerin sayısı oldukça sınırlıdır. Eğer çok fazla bilgi saklanırsa, ağ, saklanan örnek bilgilerden farklı olan yeni bir hayali bilgiye yakınsayabilir. Ağ sınıflayıcı olarak kullanıldığında böyle bir sahte bilgi bir "uyuşma yok " sonucunu oluşturacaktır. Hopfield, saklanan sınıf sayısı (M) , ağ içindeki eleman sayısının (N) 0.15 katından az olduğunda ve örnek bilgiler rastlantısal olarak üretildiğinde bu sonucun nadir olarak ortaya çıktığına işaret etmiştir. Bu nedenle, saklanan sınıf sayısı tipik olarak 0.15. N 'in altında tutulmalıdır. (Yani Verimli bir Hopfield sistemi elde etmek istiyorsak, nod sayısının, sınıfların (karakterlerin) adeti, nod sayısının en fazla 15'de biri sayısında olmalıdır (M<=0. 15*N). Eğer sınıf sayısı bu sınırı geçerse sistem artık verileri kaldıramaz olur ve sonuçlar saçmalaşır.) Örneğin , sadece 10 sınıfı saklaması istenen bir Hopfield ağı 70 ' den fazla eleman ve yaklaşık 5000 bağlantı içerecektir. 5000 tane bağlantı olması demek, bu sistemde 5000 tane ağırlık var demektir. 70 nodlu bir sistem yaklaşık 10*7 'lik bir karakter demektir ve bu da kötü bir çözünürlüktür. Hopfield ağının ikinci bir sınırlaması, bir örnek bilginin diğer bir örnek bilgi ile bir çok ortak biti vara, bu

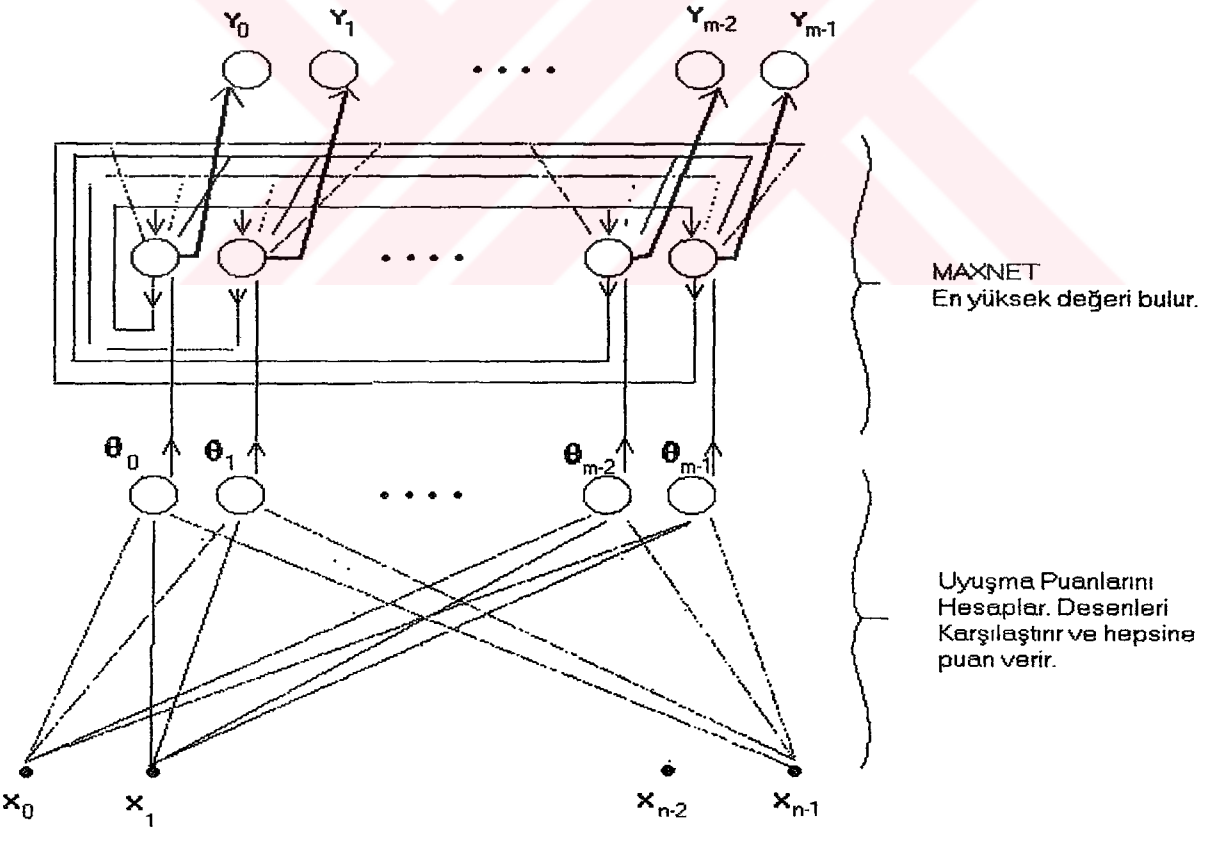
örnek bilginin kararsız olması olarak ortaya çıkmaktadır. Burada kararsızlık, sıfır anında ağ' a uygulanan bir örneğin başka bir örneğe yakınsaması olarak anlaşılmalıdır. Bu sorun bazı ortogonalizasyon yöntemleri kullanılarak ortadan kaldırılabilir ve performans arttırılabilir. Hopfield'ın modeli daha önceki lineer yaklaşımlı modellere baskın çıkan bir lineer olmayan yaklaşım kullanmışlardır. Çalışmaları, nöral ağların eksik ve bozuk bilgi ile daha iyi uğraştığını göstermiştir. Hopfield 'a göre ağın incelenmesi için çalışmalar giriş-çıkış ilişkisinin lineer olmaması üzerinde yoğunlaşmalıdır. Hesaplamanın özü lineer olmayan işlemlerde yatmaktadır.

3.4.2 Hamming net:

Hopfield 'ın gelişmiş halidir. İki tabakadan oluşur. Optimum sınıflayıcıdır.

3.4.3 Carpenter Grossberg

CG Hamming net'in bir kısım algoritmalarını kullanır. Bu metod yukarıdaki iki sistemin de gelişmiş halidir. En esaslı sınıflama metodudur.

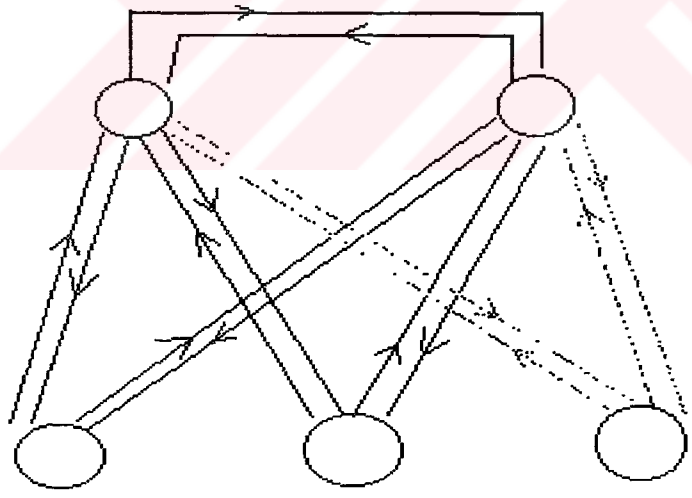


M = Desen Sayısı, N = Desenin toplam Pixel Sayısı (Nöron Sayısı)
Y = Çıktı Uçları, X = Giriş Uçları

Şekil 3.7 Network ün yapısı

Şekilde 4 kısım gözükmektedir. Birinci kısımdan veriler girilir. Yani sistemin veri giriş uçları burasıdır. Burada N adet giriş nodu vardır. (N=Desenin çözünürlüğü. Örneğin 8*16). İkinci kısım, Low Layer diye adlandırdığımız katmandır. Low Layer'da, girilen girdilerin, gerçek verilerle ne kadar uyuştuğuna dair puanlamalar hesaplanır. Üçüncü kısım, MaxNet Layer' ıdır. LowLayer'dan cıkan veriler MaxNet Layer'a giriş yapar ve bu katman, verilen veriler arasında, en yüksek puanı olanı hesaplar. Dördüncü kısım da çıktı bölümüdür. MaxNet Layer hesaplamalarını tamamladıktan sonra, bu kısımdaki değerleri yeniden düzenler. Eğer çıktı ucu sıfırdan büyükse, seçilen sınıfı işaret eder, küçükse, seçim işleminden geçemediğini gösterir. MaxNet birden fazla seçim yapabilir.

Mesela y1 ve y5 sıfırdan büyük değer alabilir. Bu gibi durumlar için, bazı işlemler yapılarak (bazı katsayılarla oynanarak) MaxNet'in tek bir çıktı vermesini sağlarız. Sonuca bir çıktı ucun değeri sıfırdan büyük şekilde elimizde kalır. Örneğin sıfırdan büyük olan nod Y3 ise , sonucumuz 3 numaralı sınıf olur. Burada M adet çıktı ucumuz vardır. Bu da bize kesinlikle bir sınıf sonucu verir. Sistem geriye bir desen göndermez, daha önce eğitim aldığı sınıflardan birini seçerek "bu desen 4 nolu desene çok benziyor" gibi kesin bir sonuç gönderir.



Şekil 3.8 Hücreler arası gidiş ve dönüş yolları

Şimdi kabaca algoritmaya bakalım;

Hopfield'da nöronlar arası sadece gidiş yolu vardı. Şimdi ise birinci kısım hücreler ile ikinci kısım hücreler arasında hem gidiş hem de dönüş yolu vardır. Şekil 3.7 'de bu çizilmemiş ama Şekil 3.8 'de bu gösterilmiştir. Normalde hücreler arası yollarda tek ağırlık değeri (weight) vardı, şimdi ise iki tane vardır. Bunlar:

1. Aşağıdan Yukarı (BottomUp)
2. Yukarıdan Aşağı (TopDown). Yani her bir yol için artık iki ağırlık değeri kullanılır.

3.4.3.1 Carpenter / Grossberg Algoritması:

Adım 1. Initialization

t(i,j) = 1 (t -> Yolların TopDown ağırlık değeri. Hücreler arası bütün TopDown ağırlık değerleri 1'e eşitlenir.)

b(i,j) = 1/(1+N) (b -> Yolların BottomUp ağırlık değerleri. Hepsi ifade edilen değere eşitlenir)

E < 1/M

E - > Epsilon

0<=i<=N-1

0<=j<=M-1

Adım 2. Girdi(Input) değerini gir. Burada giriş nodları ayarlanır.
Çıkış nodlarının (Y) değerleri sıfırlanır.

Adım 3. Compute Matching Scores

$$Y(j) = \sum_{i=0}^{N-1} b(i,j) * x(i)$$

0<=j<=M-1

Y-> Çıkış uçları. Sekil 1'de gösterildiği gibidir.

x ise girdilerdir. Girdiler 0 veya 1 olabilir.

Adım 4. Select Best Matching Examplar

$$Y(j) = Y(j) - \left(E^* \sum_{i=0}^{M-1} b(i,j)*x(i) , i<>j \right)$$

E -> Epsilon

$0 <= j,i <= M-1$

Bunun sonucunda, Output (Y) nodlarında bazı değerler oluşur. 4. aşama, Y'lerden sadece bir tane sıfırdan büyük değer kalasıya kadar yapılır. Ve bu aşamanın sonucunda değeri sıfırdan büyük olan Y değeri seçilmiş olunur. Örneğin sadece y(2) kalmış olsun . Sonuç 2'dir. Buradan çıkan sonuca ResultClass diyelim.

Adım 5. Vigilance Test

Diğerlerinden farklı olarak, çıkan sonuca bir güvenilirlik testi yapılır. İlk olarak bizim belirlediğimiz bir Ro değeri vardır. ($0 < Ro < 1$) Bu bizim kabul edebileceğimiz eşik değerini gösterir.

$$||X|| = \sum_{i=0}^{N-1} x(i)$$

j = ResultClass

$$\|T\_X\| = \sum_{i=0}^{N-1} t(i,j)*x(i)$$

{

$$\frac{\|T\_X\|}{\|X\|} > Ro$$, 7. step'e git

Değilse , 6. adıma git.

}

Adım 6. Seçilen deseni değerlendirme. (Bir dahaki iterasyonda bu sınıfı değerlendirmeye kalkma, bu sınıf, sınıfta kaldı)

Buradan Adım 3'e atlanır. Değerlendirilmeyen nodun değeri geçici olarak sıfıra eşitlenir.

Adım 7. Adapt Best Matching Examplar

Bu aşama sadece eğitim (Train) için yapılır. Eğer eğitim işlemi yapılmayacaksa, yukarıdan çıkan sonuç,

sonuç değer olarak döner. Eğer sistem eğitilecekse, bu aşamaya gelinir. Carpenter Grossberg'de istenirse, bir girdi verilir ve sistem bu girdinin kabul edilebilir sınırlar içinde en çok hangi desene benzediğini bulur. Daha sonra bu desen ile eski desen karıştırılır ve ortak desen için yeni eğitim tamamlanmış olur. Örneğin, sisteme A harfi verilmiş ve eğitilmiş olsun.

Daha sonra sisteme A'nın az bozuk halini verildi. Normalde bu az bozuk A harfi de bu sınıf içine dahil edilebilir. Şimdi bu aşamada, formülü uygulayarak, iki harfin karışımı için bu desenin eğitimi yaptırılır. Ve bundan sonra bu sınıf iki harfin karışımı olacak şekilde adapte edilmiş olunur. Artık gelecek harfler, bu iki harfin karışımı ile karşılaştırılır. Gerçek dünya

verilerini kullanırken çok işe yarayacak bir özelliktir. Örneğin; 3 adet sınıfımız var ve şu an boş olsun. A yı verdik, sistem bize 3 döndürdü ve bu desen 3 nolu sınıfa kondu ve değerler ayarlandı. B'yi verdik ,2 döndürdü ve 2 nolu sınıfa kondu. C verdik 1 döndürdü ve birinci sınıfa koyduk. Şimdiye kadar sınıflar boş olduğu için her gelen harf boş bir sınıfı kaptı. Bundan sonra da eğitim işlemine devam edilecekse; D verildi diyelim ve sistem bunu en çok ikinci sınıfa benzetti. Bundan sonra D-B harfinin karışımı için bir eğitim yapıldı ve 2 nolu sınıf yeniden uyarlandı (update edildi).

Eğer böyle bir karışım istemiyorsak ve eğitim vereceğimiz her desen için boş bir sınıf varsa yani o kadar sınıf tanımlanmışsa, yukarıdaki işlemleri yapmadan direk bu aşamaya gelebiliriz. Eğer böyle yapacaksak , ResultClass değişkeninde bir değer olmaz. Bunu bizim bildirmemiz gerekir. Örneğin, AdaptBestmatching(1), AdaptBestmatching(2), AdaptBestmatching(3)...

j = ResultClass

$$k = \frac{\sum_{i=0}^{N-1} t(i,j)*x(i)}{\sum_{i=0}^{N-1} t(i,j)*x(i)}$$

$$b(i,j) = \frac{t(i,j)*x(i)}{0.5+k}$$

$$t(i,j) = t(i,j) * x(i)$$

$$0<=i<=N-1$$

Adım 8. İkinci adıma giderek işlemlere devam et. Başa dön.

Algoritmanın yapısı bu şekildedir. Şimdi sistemin eğitildiğini kabul edelim. Test için şu işlemler yapılır.

```
Ro = 0.90;
Setinput;
Repeat
SelectBest;
VigilanceTest;
Until PassVigilanceTest;
```

Ro - Kabul edeceğimiz eşik değeri. Vigilance Test bu değere göre test eder.
Setinput ile girdiler verilir.
SelectBest'te iki işlem yapılır.

1. ComputeBestMatching - Step 3.
2. SelectBestMatching - Step 4.

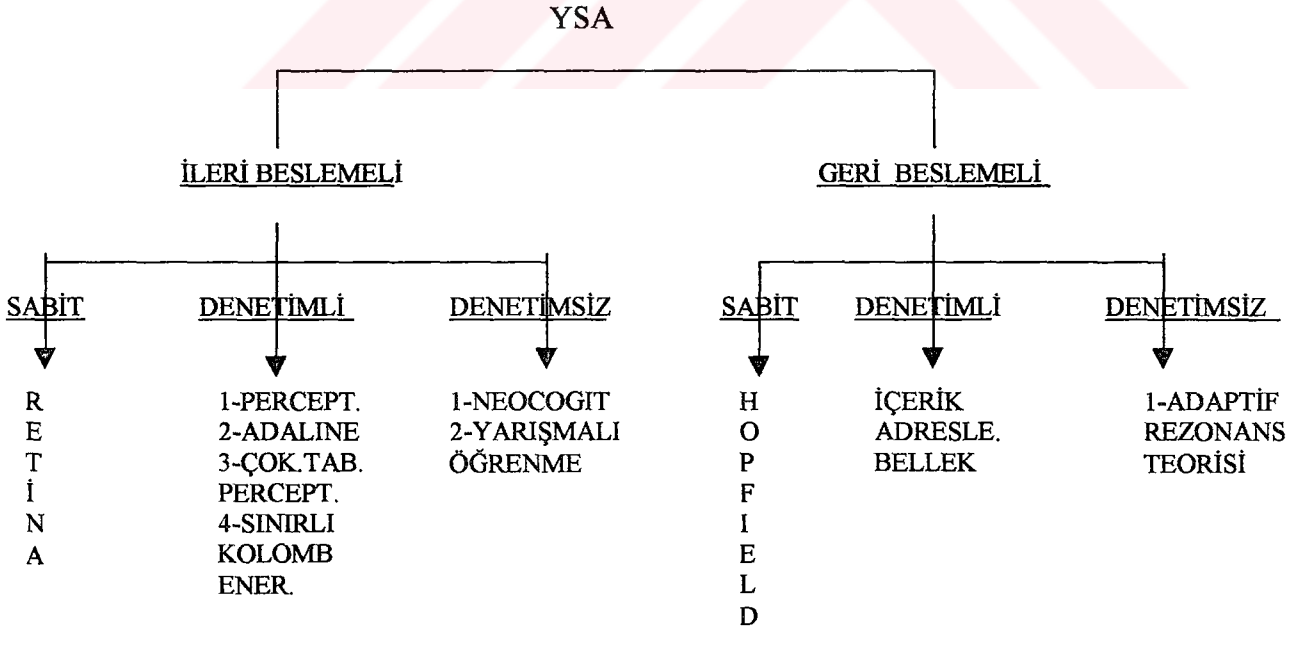
VigilanceTest - Step 5. Bu fonksiyon bize testin geçilip geçilemediğini gösterir. (PassVigilanceTest)

Eğer test geçilirse, sonuç bulunmuş olur, geçemezse, son bulduğu sınıfı (deseni) ihmal ederek tekrar döngüye devam eder. Her seferinde test ettiği sınıf sayısı bir azalır. Yukarıda verdiğimiz örneğin sonsuz döngüye girme ihtimali büyüktür. Çünkü sistem bize 0.90 oranında doğruluk payı içeren bir sonucu hiçbir zaman döndürmeyebilir. Bunu aşmak için ;

```
Ro = 1;
Repeat
 Setinput;
 L:=0;
 Repeat
  SelectBest;
  VigilanceTest;
  inc(L);
 Until (PassVigilanceTest) or (L=M);
 if PassVigilanceTest=False then Ro:=Ro-0.05;
Until (PassVigilanceTest);
```

Bu sefer, hiç bir desen VigilanceTest'i geçemezse, döngü bütün sınıfları değerlendirdikten sonra (L=M) olunca biter. Ama sistem düzgün bir sonuç bulamamıştır, test geçilememiştir. Bu durumda, Ro eşik değerimiz 0.05 kadar azaltılır. Ve herşey yeniden baslar. Bu işlem kabul edilebilir bir sonuç elde edilesiye kadar devam eder. Sonsuz döngü oluşmaz Böylece maksimum Ro ' la optimum sonucu bulmuş oluruz. Setinput'da daha önce ihmal edilmiş tüm sınıflar tekrar değerlendirilir.

Carpenter Grossberg algoritmasını da bu şekilde açıkladıktan sonra, ağ modellerine geri dönelim. YSA ile ilgili araştırma ve geliştirmeler neticesinde, bir çok ağ modeli ve eğitme algoritmaları bulunmuştur. Yapay ağ mimarileri, modelledikleri biyolojik sistemlere oldukça basitleştirilmiş yaklaşıklıkları temsil ederler. Şimdiye kadar geliştirilen YSA modellerinin her biri belli uygulama alanlarında başarılı olmuşlardır. Tüm bu modeller yoğun ara bağlantılar tarafından bir araya getirilen işlem elemanlarından oluşmalarına rağmen birbirlerinden, öğrenme kuralı, bağlantı topolojisi ve kullanılan nöral modeli ile farklı olmaktadırlar.
Yapay sinir ağlarını yapı olarak ileri beslemeli veya geri beslemeli ağlar şeklinde iki sınıfa ayırmak mümkündür. Şekilde bazı tanınmış ağlar öğrenme yöntemlerine ve ileri ya da geri beslemeli bir yapıya sahip oluşlarına göre sınıflandırılmıştır.



Şekil3.9 Yapay Sinir Ağının yapısı

Bunların bir kısmını yukarıda açıklamıştık. Şimdi diğerlerinden en çok kullanılanlarını açıklayalım:

3.5 Denetimli yapay sinir ağları:

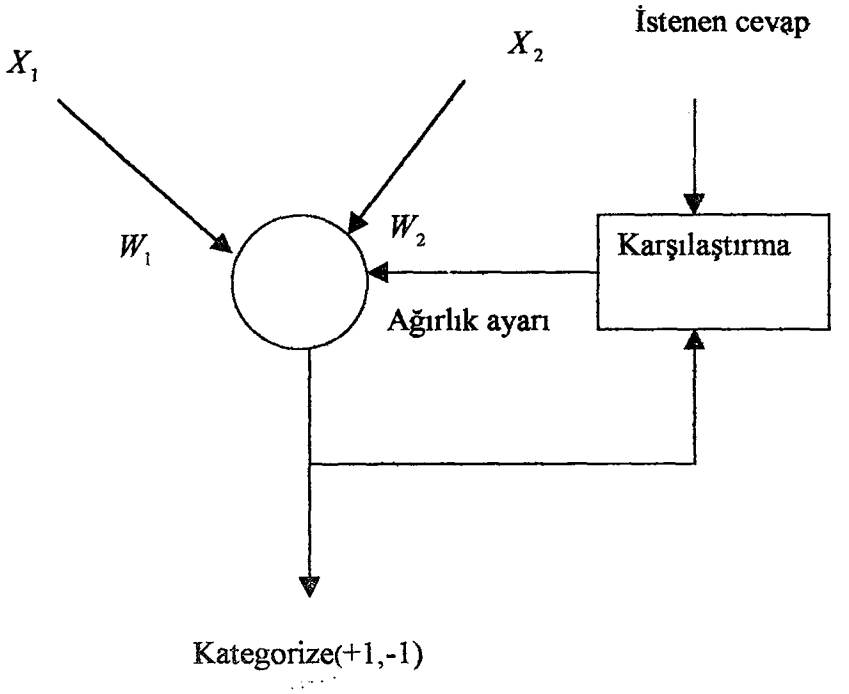
Denetimli öğrenme yöntemlerinde ağ, kendi çıkışı ve arzu edilen çıkış arasındaki farkı azaltacak şekilde parametrelerini değiştirir. Ağı izleyici kümesi, rastgele olarak seçilmiş noktalardan meydana getirilir. "Perceptron", Çok Tabakalı Ağ, Kısıtlı Kolomb Enerjisi (RCE), Büyüme ve Öğrenme (GAL) ağını denetimi öğrenme yöntemi kullanan ağlara örnek olarak verebiliriz.

3.5.1 Perceptron

Peceptron ağı, ilk 1943 yılında Mc Culloch ve Pitts tarafından saptandı. Bu ilk perceptron modeline göre, giriş bilgisinin mevcut iki sınıftan hangisine eşit olabileceğini bulacak şekilde eğitilen basit bir ağdır. Daha sonra 1960 yıllarında F. Rosenblatt yukarıdaki ağ tipini biraz daha geliştirdi. (Hebb, 1949). Ama Minsky ve Papert bu tek katmanlı perceptronun XOR (ayrıcalıklı veya) işlemini gerçekleştiremediğini ispatladılar. XOR gibi 3 veya daha fazla sınıfa ihtiyaç duyulan problemleri çözmek için yapılması gereken işlem, YSA'na yeni katmanlar eklemektir. Eşik bağlarıyla oluşturulan karar bölgesi şeklinin karmaşıklığı sadece eklenmiş olan katmanların sayısıyla sınırlıdır. Yazılı karakter tanıyabilir , karmaşık karakterleri tanıyamaz bilinen en eski YSA'dır.

3.5.2 Adaline:

Adaptif lineer elemanlar için minimum hata ile öğrenme tekniği olan Adaline, 1960 'ta Bernard Widrow ve Ted Hoff tarafından bulunmuştur. Öğreticili eğitimin ilk pratik örneğini vermişlerdir. Adaline basit iki kutup çıkışlıdır. Eğer girişin net ağırlıklı toplamı 0'dan büyük ise çıkış +1 ve eğer net ağırlıklı giriş toplamı 0'dan küçükse çıkış –1'dir. Tipik olarak +1 çıkış A ve –1 çıkış olarak B olarak kategorize edilebilir. Aşağıdaki şekilde iki giriş vektörü olan basit bir Adaline gösterilmiştir.



Şekil 3.10 Basit Adaline

Şekilden de görüldüğü gibi, Adaline gerçek çıkışı ile istenen değeri karşılaştırmaktadır. Bu bilgi ile ağırlık ayarları yapılır. Adaline hatası,

Hata = < istenen değer > - < gerçek çıkış > dır.

Hata bulunduktan sonra ağırlık ayarları delta kuralı öğrenme yöntemi ile yapılır. Delta kuralı, ağırlıklardaki değişimleri:

$$W = W^{eski} + \frac{\beta E x}{|x|^2} \tag{3.8}$$

ifadesi ile hesaplanır.

Bu ifadede

β : 0 ve 1 arasında sabit bir öğrenme katsayısıdır.

E :hesap edilen hata değeridir.

x : $(X_1 ve X_2)$ bileşenlerinden oluşan giriş vektörü,

W : (W_1, W_2) bileşenlerinden oluşan ağırlık vektörleridir.

Adaline, en küçük kareler toplamı veya delta kuralı olarak adlandırılan öğrenme kuralını kullanır. Geriye yayınım ağı , delta kuralını çok katlı ağa göre uyarlar. Bu uyarlamaya genelleştirilmiş delta kuralı denir. Geriye ayınım ağı, Adaline öğrenme yönteminin bir varyasyonudur. Adaline, Radar bozucuların etkisizleştirilmesi, telefon hatlarında adaptif

dengeleyici olarak uygulama alanı bulmuştur. Giriş çıkış arasında lineer bir ilişki olduğunu kabul eder. Güçlü bir öğrenme kuralı vardır. Halen kullanılmaktadır.

3.5.3 Geriye yayınım:

Yazılı metinden söz sentezi, robot kollarının adaptif kontrolü ve otomatik kontrol alanında çok geniş uygulama alanı bulmuştur. Günümüzde en popüler ağdır. Geriye yayınım ağı, Adaline öğrenme yönteminin bir varyasyonudur. Adaline , en küçük kareler toplamı veya delta kuralı olarak adlandırılan öğrenme kuralını kullanır. Geriye yayınım ağ' ı, delta kuralını çok katlı ağ' a göre uyarlar. Bu uyarlamaya genelleştirilmiş delta kuralı denir.

3.5.4 Geriye yayılmalı yapay sinir ağı

Geriye Yayılmalı YSA, çok katmanlı perceptron ağ yapısı üzerinde geriye yayılma yöntemiyle genelleştirilmiş delta öğrenme kuralının uygulandığı bir ağdır. Minsky ve Papert 1969'da iki katmanlı ileri beslemeli ağların tek katmanlı perceptrondaki bir çok sınırlamayı ortadan kaldırdığını göstermiş, fakat gizli katmanların ağırlıklarının nasıl değiştiği konusuna bir çözüm getirememişlerdir. Geriye yayılma bu probleme bir çözümdür. Geriye yayılma aynı zamanda, Widrow ve Hoff tarafından öne sürülen Delta kuralının lineer olmayan aktivasyon fonksiyonları ve çok katmanlı ağlar için genelleştirilmesi olarak da düşünülebilir. Literatürde Delta kuralına Widrow-Hoff kuralı, LMS (Least Mean Square: En Küçük Karesel Ortalama) kuralı da denmektedir.

Örnek olarak üç katmanlı bir YSA'nın geriye yayılma (Back Propagation) algoritması ile eğitilmesini gözönüne alalım. Ağdaki her bir düğüm tek bir işlem elemanıdır. Düğümler katmanlarda düzenlenir. Veri girişleri giriş düğümlerinden yapılır. Giriş düğümleri pasiftir ve hesaplanabilir değildir. Giriş ile çıkış katmanı arasında kalan "gizli" düğümlere ağırlıklı veri değerleri bu katmandan iletilir . Her BP düğümü için ayrıca bir eşik değer girişi daha mevcuttur. Bu girişler düğümün bias'ı ya da referans seviyesi olarak adlandırılır. Bütün gizli düğümler, tüm giriş verilerini alır . Fakat her biri farklı bir ağırlık kümesine sahip olduğu için, değerler kümesi de farklıdır. Her bir gizli düğüm kendisine gelen girişleri işleyerek sonucu çıkış düğümlerine aktarırlar . Çıkışlar da farklı ağırlık kümesine sahiptir. Geriye Yayılmalı ağ için gizli ve çıkış düğümleri, kendisine giren girişleri iki aşamada işlerler. Her giriş kendisinin ağırlığı ile çarpılır, çarpımlar toplanır ve sonra bu toplam bir fonksiyon üzerinde çıkış elde edebilmek üzere, bir sonraki katmana geçirilir. Bu transfer fonksiyonuna "Sigmoid fonksiyonu" denir. Sigmoid fonksiyonu bir nonlineerlik oluşturur. Bu nonlineerli ağın

yeteneğine daha fazla güvenilmesini sağlar. Geriye Yayılma öğrenme algoritmasının en büyük özelliği, hatalara karşı verdiği cevapta, ağırlıklarının değerini değiştirebilmesidir. Hataları hesaplamak için, girişi etiketlenmiş örneklerden oluşan veri dizisinin hedeflenen çıkış örneklerini de içermek zorunluluğu vardır.

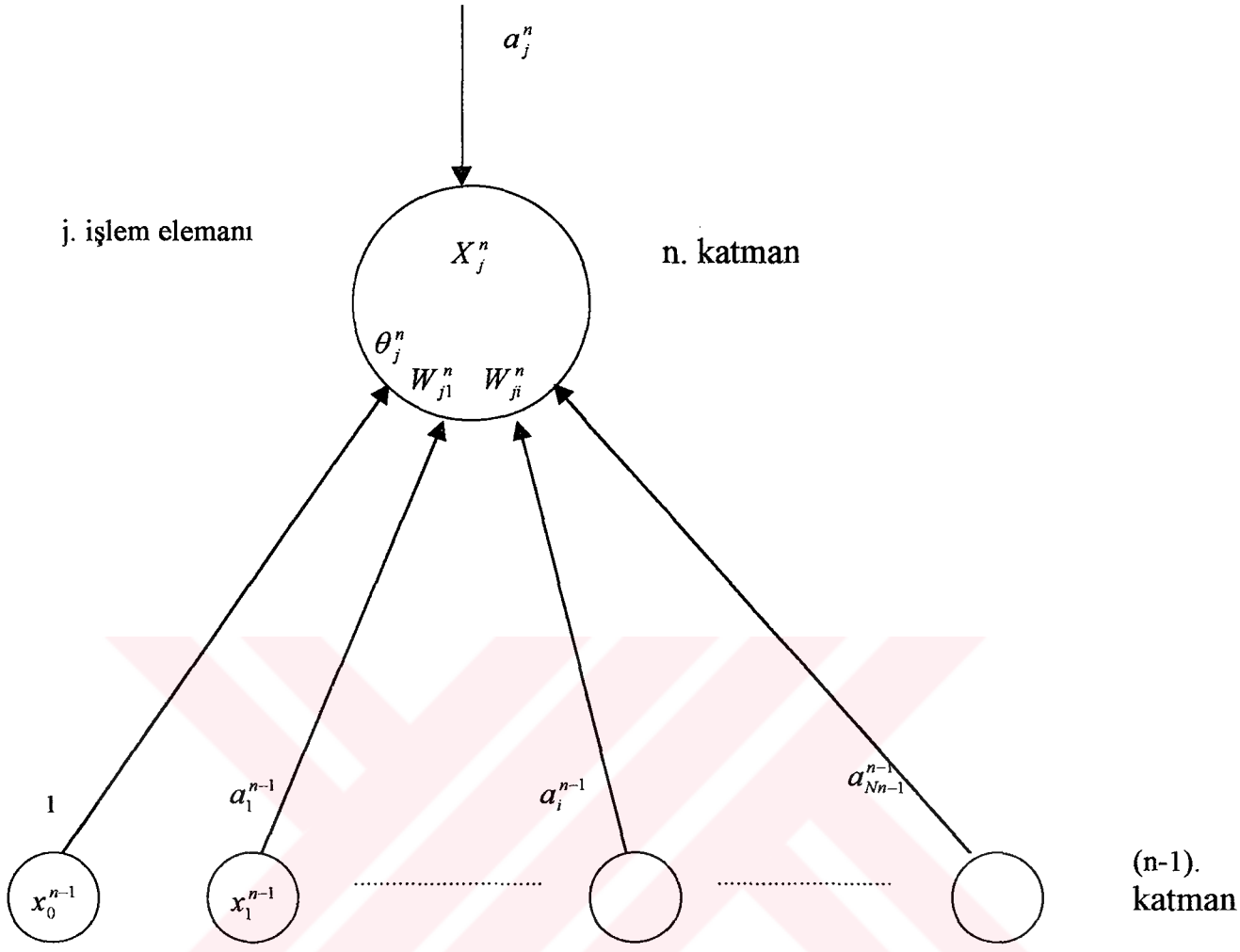
Geriye yayılmalı ağın eğitilmesi sırasında ağ, her giriş örneğini, çıkış düğümlerinde sonuç üretmek üzere gizli katmanlardaki düğümlere geçirir. Sonra da çıkış katmanındaki hataları bulabilmek için amaçlanan hedef sonuçtan gerçek sonucu elde eder. Bundan sonra ağ çıkış hatalarının türevini çıkış katmanından geriye doğru gizli katmanlara geçirirler. Bunu yaparken de, orijinal ağırlık bağlantılarını kullanırlar. Geriye Yayılmalı ismini veren de işte bu hataların geriye doğru yayılması özelliği olmuştur.

Her çıkış ve gizli düğümlerin hata değerleri bulunduktan sonra, düğümler kendi hatalarını azaltmak için kendi ağırlıklarını ayarlar. Ağırlık değiştirme denklemleri ağdaki hataların karelerinin toplamını minimize edecek şekilde düzenlenir. Bu söylenenlerin denklemsel gösterilişleri aşağıda verilmiştir(Karlık, 1994).

3.5.4.1 Geriye yayılma öğrenme kuralı:

Bu çözümün arkasındaki temel düşünce, çıkış katmanının hücrelerine ait hataların geriye yayılmasıyla gizli katmanların hücrelerine ait hataların belirlenmesi sonucu ağın bağlantı ağırlıklarının bu hataları minimum yapacak şekilde değiştirilmesidir.

Gidilecek yolun daha iyi anlaşılması için ilk önce notasyonlar üzerinde durulacaktır(Şekil 3.6). n=0, 1, . . . , çıkış olmak üzere katman numarasını temsil eder. 0 numaralı katman yapay sinir ağına gelen girişlerin bulunduğu katmandır. Bu katmanın 'sıfır katmanı' olarak adlandırılmasının bir nedeni de burada girişlerin bir işlem yapılmadan çıkışa verilmesi ve bir sonraki katmandaki işlem elemanlarına dağıtılmasıdır. Her katmandaki eleman sayısı N_n ile gösterilsin:



Şekil 3.11 Geriye yayılmalı Ağ Yapısı Notasyonu

n. katmandaki j nolu işlem elemanını incelediğimizi düşünelim. Bu elemana gelen girişler bir önceki katmandaki elemanların çıkışları olan X_j^n dir. Bu elemanın çıkışı ise şöyle tanımlıdır:

$$x_j^n = \sum_{i=1}^{N_{n-1}} a_i^{n-1} w_{ji}^n + \theta_j^n \qquad (3.9)$$

θ eşiktir. θ_j^{n-1} eşik değerini w_{ji}^{n-1} 'a eşitlersek ve $a_{n-1,0} = 1$ alırsak denklemi daha basitçe gösterebiliriz:

$$x_j^n = \sum_{i=0}^{N_{n-1}} a_i^{n-1} w_{ji}^n \qquad (3.10)$$

$$a_j^n = F(x_j^n) \tag{3.11}$$

F fonksiyonu lineer olmayan bir fonksiyon olup genelde şekildeki gibi bir sigmoid fonksiyonudur.

En küçük karesel ortalama hata yöntemi, ortalama karesel hata fonksiyonunu minimize eden bütün ağırlık değerlerini , gradyen azalma denen bir metodla bulur. Ortalama karesel hata , aşağıdaki formülle ifade edilir:

$$F(w) = \lim_{N \longrightarrow \infty} \frac{1}{N} \sum_{k=1}^{N} (y_k - y_k^1)^2 \tag{3.12}$$

Burada y_k arzu edilen çıkış, y_k' elde edilen çıkıştır.

Bu eşitlik $F(w) = E[(y_k - y_k')^2]$ biçiminde de yazılabilir. Burada E istatistikteki beklenen değer operatörüdür. Beklenen değer operatörü , parantezler arasındaki büyüklüğün belirli bir olasılık yoğunluk fonksiyonuna sahip rastgele bir değişken olduğu varsayımı altında, parantezleri içindeki bu büyüklüğün ortalamasını hesaplar.

Geriye yayılma algoritmasını gerçeklemede giriş örnekleri rastgele seçildiği için artık minimum yapmaya çalışacak eşitlik aşağıdakine dönüşür. Bir X=($x_1, x_2, ..., x_i$) giriş vektörü için

$$E = \frac{1}{2} \sum_{j=1}^{N_{çıkı}} (d_j - a_j^{çıkış})^2 \tag{3.13}$$

Burada d istenen çıkışı, a elde edilen çıkışı gösterir.

Temel düşünce varolan örüntü üzerinde her ağırlığa göre ölçülen hatanın türevinin eksi işaretlisiyle orantılı olarak ağırlığı değiştirmektedir. :

$$\Delta_p w_j = -\gamma \frac{\partial E^P}{\partial w_j} \tag{3.14}$$

Burada γ orantı sabitidir.

Geriye yayılma kuralında bir çevrimde bütün girişlerin hatalarının toplamına (E=$\sum E^P$,

p: girişörnek sayısı) bakıldığı gibi, bir giriş için yapılan hatadan da yola çıkılabilir. Oluşturulan programda bu sunucu tipinde işlem yapıldığından eşitlikler ona uygun olarak anlatılacaktır. Bu anlatım genel ifadeyi bozmaz.

Ağırlık değerlerinin (3.13) eşitliğine olan etkisini çıkarmak için $\frac{\partial E}{\partial w}$ yı oluşturmak , ve böylece bu ifadeyi minimum yapacak w_{ij} değerlerini bulmak gerekir. Önce çıkış düğümleri daha sonra gizli katmanlardaki düğümler için ardışık olarak bağıntılar elde edilir.

$$\frac{\partial E}{\partial w_{ji}^{\text{çıkış}}} = \frac{\partial E}{\partial x_j^{\text{çıkış}}} \frac{\partial x_j^{\text{çıkış}}}{\partial w_{ji}^{\text{çıkış}}} \tag{3.15}$$

(3.11) denkleminden yararlanarak (3.15) eşitliğinin sağ tarafındaki ikinci çarpanı,

$$\frac{\partial x_j^{\text{çıkış}}}{\partial w_{ji}^{\text{çıkış}}} = a_i^{\text{çıkış}-1} \tag{3.16}$$

olarak bulunur. Aynı eşitlikteki birinci çarpan

$$\frac{\partial E}{\partial x_j^{\text{çıkış}}} = -\delta_j^{\text{çıkış}} \tag{3.17}$$

ile gösterilir.

$$\frac{\partial E}{\partial x_j^{\text{çıkış}}} = \frac{\partial E}{\partial a_j^{\text{çıkış}}} \frac{\partial a_j^{\text{çıkış}}}{\partial x_j} \tag{3.18}$$

olarak alınırsa (3.18) de (3.14) ve (3.12) eşitliklerinden faydalanarak,

$$\frac{\partial E}{\partial x_j^{\text{çıkış}}} = -(d_j - a_j^{\text{çıkış}}) F'(x_j^{\text{çıkış}}) \tag{3.19}$$

$$\delta_j^{\text{çıkış}} = (d_j - a_j^{\text{çıkış}}) F'(x_j^{\text{çıkış}}) \tag{3.20}$$

biçiminde elde edilir.

F( ) olarak sigmoid fonksiyonu kullanıldığında

$$F'(x_j^{\text{çıkış}}) = \frac{\partial}{\partial x_j^{\text{çıkış}}} \frac{1}{1+e^{-x_j^{\text{çıkı}}}} \tag{3.21}$$

$$= \frac{1}{(1+e^{-x_j^{\text{çıkı}}})^2} (e^{-x_j^{\text{çıkı}}}) \tag{3.22}$$

$$= \frac{1}{(1+e^{-x_j^{\text{çıkı}}})} \frac{(e^{-x_j^{\text{çıkış}}})}{(1+e^{-x_j^{\text{çıkı}}})}$$

$$= a_j^{\text{çıkış}} (1 - a_j^{\text{çıkış}})$$

Artık kısaltma açısından

$$\delta_j^{\text{çıkış}} = a_j^{\text{çıkış}} (1 - a_j^{\text{çıkış}})(d_j - a_j^{\text{çıkış}}) \tag{3.23}$$

denir.

(3.16) da değerleri yerine konabilir:

$$\Delta w_{ji} = \gamma \; a_j^{\text{çıkış}} (1 - a_j^{\text{çıkış}})(d_j - a_j^{\text{çıkış}}) \; a_i^{\text{çıkış}-1}$$

$$w_{ji}^{\text{çıkı}}(t+1) = \gamma \; \delta_j^{\text{çıkış}} \; a_i^{\text{çıkış}-1} + w_{ij}^{\text{çıkış}}(t) \qquad (3.24)$$

Eğer (3.20) denkleminde x_j çıkış hücresi değilse, hücrenin ağın çıkış hatasına katkısı bilinmemektedir.

Ancak, hata ölçüsü gizli katmanlardan çıkış katmanına olan net girişlerin bir fonksiyonu olarak yazılabilir. Bir gizli hücre için hata işareti , aralarında direkt bağlantı bulunan hücrelerin hata işaretleri ve o bağlantıların ağırlıkları cinsinden ardışık olarak belirlenir. Sigmoid aktivasyon fonksiyonu için

$$\delta_i^n = F'(x_i^n)\sum_{j=1}^{N_{n+1}} d_j^{n+1} w_{ji}^{n+1} = a_i^n(1-a_i^n)\sum \delta_j^{n+1} w_{ji}^{n+1} \qquad (3.25)$$

dir. w'nin değişimi yine (3.15) deki gibi

$$w_{ji}^n(t+1) = \gamma \; \delta_j^n a_i^{n-1} + w_{ji}^n(t) \qquad (3.26)$$

olur. Bu geriye yayılma için en genel ifadedir.

3.5.5 Çok katmanlı perceptron (multi-layer perceptron)

Çok katmanlı perceptron giriş ve çıkış katmanları arasında birden fazla katmanın kullanıldığı YSA sistemleridir. Gizli katman (hidden layer) olarak isimlendirilen bu katmanlarda, düğümleri aracısız giriş olmayan ve aracısız çıkış veremeyen üniteler vardır. Şekilde çok katmanlı perceptronun genel yapısı verilmiştir.

İki katmanlı ağlarda veriler giriş katmanı tarafından kabul edilirler. Ağ içinde yapılan işlemler sonucunda çıkış katmanında oluşan sonuç değer işlenen cevap ile karşılaştırılır. Bulunan cevap ile istenen cevap arasındaki herhangi bir ayrılık varsa ağırlıklar bu farkı azaltacak şekilde yeniden düzenlenir. Girişteki değer, ağırlıklar uygun noktaya ulaşıncaya kadar değişmez. Hesaplanan çıkışlar istenilen cevaplarla karşılaştırılarak sonuçta gerekirse hata işaret edilir. Hata işareti gizli birimlerden çıkış birimine olan ağırlıkları değiştirmekte kullanılır. Ama bunu yaparken giriş katmanından gizli katmana gelenin değiştirilip değiştirilemediğini düşünmek gerekir. Gizli birimlerden ne tür bir çıkış istendiği bilinemeyeceği için gizli birimlerin çıkışında hata işareti verilmesi kolay bir şey değildir. Bunun yerine her bir birimin çıkış biriminin hatalarına olan etkisi bilinmelidir. Bu hatalı birim için gizli birime bağlı olan çıkış birimlerinin hata işaretlerinin ağırlıkları toplamı alınarak yapılır.

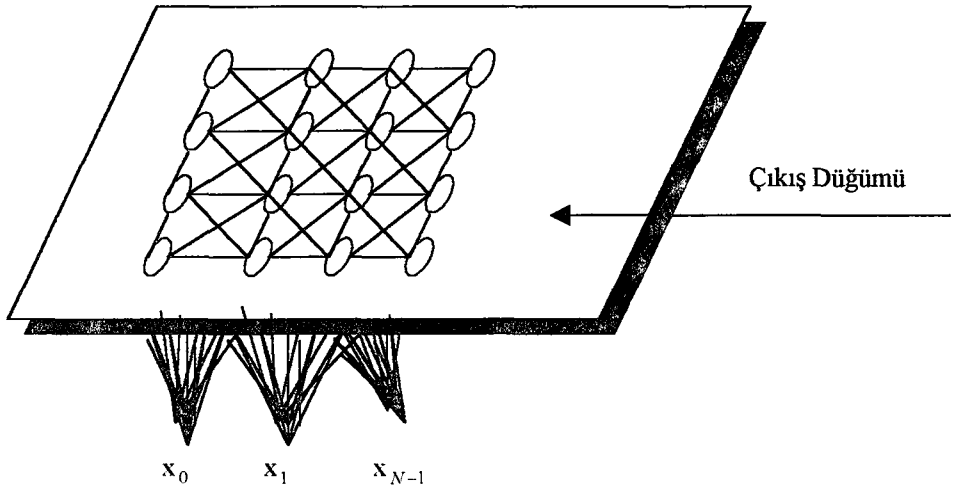
Çok gizli katmana sahip sistemlerde her sistemin hata işaretleri, bir önceki katmanın düzeltilmiş işaretlerinden çıkartılarak işlem tekrarlanır. Sonuç olarak ağırlık düzeltme işlemi çıkış seviyesine bağlı ağırlıklardan başlar ve işlem ters yönde, giriş seviyesine varana kadar devam eder. Sonuçta sistem hatalar yapar, ama bu hatalardan bir şeyler öğrenip isteneni bulana kadar işleme devam eder. Bu yönteme " hatanın geriye yayılma algoritması" (Back-propagation algorithms) denir.

3. 6 Denetimsiz yapay sinir ağları:

Denetimsiz öğrenme yöntemlerinde ağ, sınıfların giriş uzayındaki dağılımını inceleyerek kendi parametrelerini değiştirir. Bu yöntemde arzu edilen çıkış yoktur.

3.6.1 Kohonen ağı

Hücreler komşularına meksika şapkası bağlantı biçimi ile bağlıdır. Bu bağlantı sayesinde yarışmalı öğrenmede (competitive learning) bir hücrenin kazanması temin edilir. Şekil 3.12 'de Kohonen ağının yapısı ve hücreler ile komşuları arasındaki bağlantı şekli gösterilmiştir. Kohonen ağının eğitiminde sadece kazanan hücrenin ağırlığı değil , o hücre çevresinde seçilen yine belli bir komşuluk içindeki hücrelerin ağırlıkları da değiştirilir.
Öğrenme esnasında bu komşuluğun sınırları zamanla lineer olmayacak şekilde azaltılır. Bu yöntem sayesinde , giriş uzayındaki sınıfların içleri hemen hemen eşit uzaklıkta noktalar tarafından örneklenir.



Şekil 3.12 Kohonen ağı modeli

3.6.1.1 Kohonen ağının öğrenme algoritması

Aşağıda kohonen ağının öğrenme algoritmasının adımları gösterilmektedir. Ağda N adet giriş hücresi ve M adet çıkış hücresi bulunduğunu varsayalım:

Adım 1:N adet giriş ile M adet çıkış arasındaki ağırlık katsayılarını 0 ile 1 arası rastgele küçük değerlere ayarla ve bir başlangıç komşuluk değeri seç. İterasyon sayısını belirle.

Adım 2 :Ağa izleyici küme içinden rastgele bir giriş ver.

Adım 3 : Giriş vektörü ile ağırlıklar arsında aşağıda ifade edilen uzaklıkları hesapla.

$$D_j = \sum_{i=0}^{N-1} (X_i(t) - W_{ij}(t))^2 \qquad (3.27)$$

X_i , giriş vektörünün elemanlarını ; W_{ij} ise j. inci çıkışın ağırlık vektörünün elemanlarını temsil eder.

Adım 4 : Minimum uzaklığa sahip j^* çıkışını bul.

Adım 5 : j^* çıkışının ve komşularının ağırlıklarını aşağıdaki şekilde değiştir.

$$W_{ij}(t+1) = W_{ij}(t) + \eta(t) * (X_i(t) - W_{ij}(t)) \qquad (3.28)$$

Kazanç terimi olarak ifade edilen $\eta(t)$, 0 ile 1 arasında ve zamanla azalacak şekilde değer alır.

Adım 6 : İterasyon sayısını azalt. İterasyon sayısı 0 a eşit değil ise Adım 2 ye git , aksi takdirde algoritmadan çık.

3. 6. 2 Adaptif rezonans teorisi:

Şekil tanımada etkilidir. Ölçekteki değişimlere ve gürültüye karşı duyarlıdır.

4 UYGULAMA

4.1 Açıklamalar

Sistemde 10'a 10 luk desen alanı olan 4 sınıflı 100 nöron vardır. Amaç desen tanımaktır. Gelen desenin, hangi sınıfa ait olduğunu bulunmaya çalışılmıştır.

Genel tanımlamalar Patterns.Pas dosyasında (Patterns Ünitesinde) tanımlıdır. Sample.Pas dosyasında ise prosedürlerin genel olarak yaptıkları işler şöyledir:

InitWays: Nöronlar arasındaki yolların ağırlık (weight) değerleri sıfırlanır.

InitNorons: Bütün nöronların Girdi (Input) ve Çıktı (Output) değerleri sıfırlanır.

TrainSystem: Tanımlanmış sınıflara göre (Classes) hücreler arası yolların ağırlık değerleri hesaplanır.

CalculateNorons: Bir girdi verildikten sonra, hücrelerin çıktı değerleri hesaplanır. Bu prosedürde Hopfield algoritmasındaki formülü uygular. Bu algoritma şu şekildedir:

Hedef hücrenin çıkış değeri = hedef hücrenin çıkış değeri + Yolun ağırlığı * Uygulanan girdi (1 ya da –1)

HardLimitFunction: CalculateNorons çağrıldıktan sonra çıkan output l bir eşik değerinden geçirilir. Bu fonksiyonda bu eşik değer fonksiyonunu hücrelerin output değerlerine uygular.

```
[ Output >  0 ,  1
  Output <= 0 , -1 ]
```

SetInput: Sisteme değerlendirmesi için yeni veri verilir.

GetOutput: Sistem verilen girdiyi değerlendirdikten sonra, yine sistemin oluşturduğu çıktıyı almak için kullanılır.

GenerateRandomInput: Bir desen alınıp, belli oranda desen bozma işlemi burada yapılır. Bu oran Ratio isimli değişkende tutulur. Örneğin Random(4) ise 1/4 oranında bozma işlemi yapılır. Ve desenin bozulmuş hali geri döndürülür.

InitProgram: Bu işlem program başladığında bir kez yaptırılır. Noronlar, aralarında yollar ve yapılması gereken ilk ayarlamalar yapılır.

GiveMaxSimilar: Hopfield tam bir sonuç vermeyebilir. Bu prosedür çıkan sonucu, tam olarak bir sınıfa uydurmaya çalışır. Yaptığı iş; gelen desenin, düzgün desenler içinde en çok hangisine uyduğunu puan vererek araştırmaktır. En çok puanı alan kazanır.

Şimdi bu işlemi nasıl yaptığını açıklayalım:

GiveMaxSimilar prosedürü, elimizdeki desenle düzgün desenleri karşılaştırıp, hangisine daha çok benzediğini bulmaya çalışır demiştik. Bunu,direk karşılaştırma ile yapar. Programda Points adlı bir dizi var. Elimizdeki desene d1 diyelim, Classes'da düzgün desenler var. İlk

önce, d1 ile Classes[1] deseni karşılaştırılır. Yani d1[0]=classes[1][0], d1[1]=classes[1][1], d1[2]=classes[1][2] ..... diye döngüde karşılaştırılır. Eğer eşitse, points[1] bir arttırılır, değilse bir azaltılır. Karşılaştırılanlar, desenlerin pixelleridir.d1[0] ; d1 deseninin 1. pixeli classes[1][0] classes[1] deseninin 1.pixeli olur. Yani direk desenlerin pixelleri karşılaştırılır. ve ona göre points'e o desen ile ilgili puan verilir. Bu işlem bütün sınıflar için yapılır.(Yani; d1 classes[1] ile, d1 classes[2] ile, d1 classes[3] ile .... karşılaştırılır.) Kaç tane sınıf varsa devam edince sonuçta, elimizde points diye bir dizi oluşur. Points[1]'de birinci karşılaştırma sonucu oluşan puan var ,points[2]'de ikinci karşılaştırma sonucu .... Şimdi bizim hangi sınıfın daha çok puan aldığını bulmamız gerekiyor. Yani elimizde sayılardan oluşan bir dizi var. Bu dizi içindeki en büyük sayısal değeri bulmamız gerekiyor. Bunun için değişik algoritmalar var. Benim burada kullandığım ise, Bir sabite (j) küçük bir değer atamaktır. Bu küçük değer, dizideki elemanların alamayacağı kadar küçük bir değer olmalıdır. Burada -32000'in değerini hiçbir eleman alamıyor. Şimdi elimizde dizide olmayan en küçük sayı var (j). Ve tek tek karşılaştırıyoruz. Bir değer j'den büyükse, bu değeri j'ye atıyoruz. Sonra bu işleme devam ediyoruz. Döngü bitince, elimizde dizideki en büyük sayısal değer oluyor. Bu değer j'de kalıyor. Bu arada bu değerin hangi sınıfta olduğunu da k değişkenine atmış oluyoruz. Sonuçta, points dizisindeki en büyük değere sahip elemanı bulmuş oluyoruz. Ve bunu da output olarak veriyoruz.

```
Procedure GiveMaxSimilar(Inp:TPattern;Var Out:TPattern);
Var
i,j,k:Integer;
Points:Array [1..M] of Integer;
Begin
For i:=1 to M do Points[i]:=0;
// Points dizisi sıfırlanır.

For k:=1 to M do // M adet sınıf var.
For i:=1 to wY do
For j:=1 to wX do // wX*wY adet pixel var.
If Inp[(i-1)*wY+j]=Classes[k][(i-1)*wY+j] then inc(Points[k])
else Dec(Points[k]);
```

Burada inp ile classes karşılaştırılır. Pixeller aynıysa points'in ilgili elemanı 1 arttırılır, (inc(points[k])) değilse 1 azaltılır. Yani (dec(points[k])) yazılır.

j:=-32000;

Burada j'ye rastgele en küçük değer verilir.

For i:=1 to M do

If Points[i]>j then begin k:=i; j:=Points[i]; end;

(Burada dizideki en büyük değeri bulmuş oluruz. Değer j'de, hangi eleman olduğu ise k'da tutuyoruz.)

For i:=1 to wY do

For j:=1 to wX do Out[(i-1)*wY+j]:=Classes[k][(i-1)*wY+j];

End;

(Bulunan desen (k. desen) output olarak verilir.)

Test : Rastgele sınıflar seçip bunları bozar. Sonra sisteme bu bozulmuş veriyi verip doğru sonuca ulaşmaya çalışır.

ReadNewRatio : Yeni bozma oranını (Ratio) okur.

Bir test işlemi yapmak için;

SetInput(Ptr); Sisteme yeni veri verilir.

CalculateNorons; Hücreler arası yol değerlerine göre çıktılar (outputlar) hesaplanır.

HardLimitFunction; Tüm hücrelerin output değerlerine eşik değer fonksiyonu uygulanır.

GetOutput(Ptr); Son olarak sistemin ulaştığı veri (output) alınır.

Sistemin Train edilmesi (Eğitilmesi);

İlk olarak, InitNorons ve İnitWays prosedürleri çağrılır. Sonra InitW, InitX, InitGuzelA, InıtO gibi procedürler çağrılarak Classes' taki desenlere bu desenler yerleştirilir. Sonra da TrainSystem çağrılır ve sistem Classes'taki desenlere göre kendini yeniden düzenler.

Bazı Sabitler;

wX,wY : Desenin boyutları. Burada 10'a 10'luk bir desen alanı tanımlandı.

M : Desen (sınıf) adeti.

Ratio : Bozma oranı (1/Ratio).

wX*wY : Toplam nöron adeti. Burada 100.

wX*wY*wX*wY : Toplam yol adeti. Burada 10000.

#27 karakteri Escape tuşunun ascii kodudur.

repeat .... until readkey=#27; derken, aradaki döngünün esc tuşuna basılana kadar yapılacağını belirtmiş oluruz. Böylece kullanıcı, esc tuşunun haricinde bir tuşa basınca döngü devam eder, esc 'e basınca döngü biter ve çıkılır.

Hopfield sistemiyle ilgili olarak, şunu söyleyebiliriz. Bu sistem, bizim verdiğimiz bozuk deseni bir sınıfa sokmaz. Yani bu bozuk desen 3 nolu desendir demez. Yaptığı şey, aldığı eğitim doğrultusunda, yine aldığı bozuk deseni, düzgün hale getirmektir. Yani bir bakıma bozuk deseni doğru olana doğru yakınsatır. Ama kesinlikle şu sınıfa aittir demez. Hopfield, bize kendi çapında düzelttiği yeni bir desen verir.

Hopfield'da yeni değer bir önceki değerle karşılaştırılır demiştik. Bu şu anlama gelir. Sistem işi tek döngüyle (girdiyle) halletmez. Sisteme doğru sonuca ulaşana kadar input veriyoruz. Örneğin;

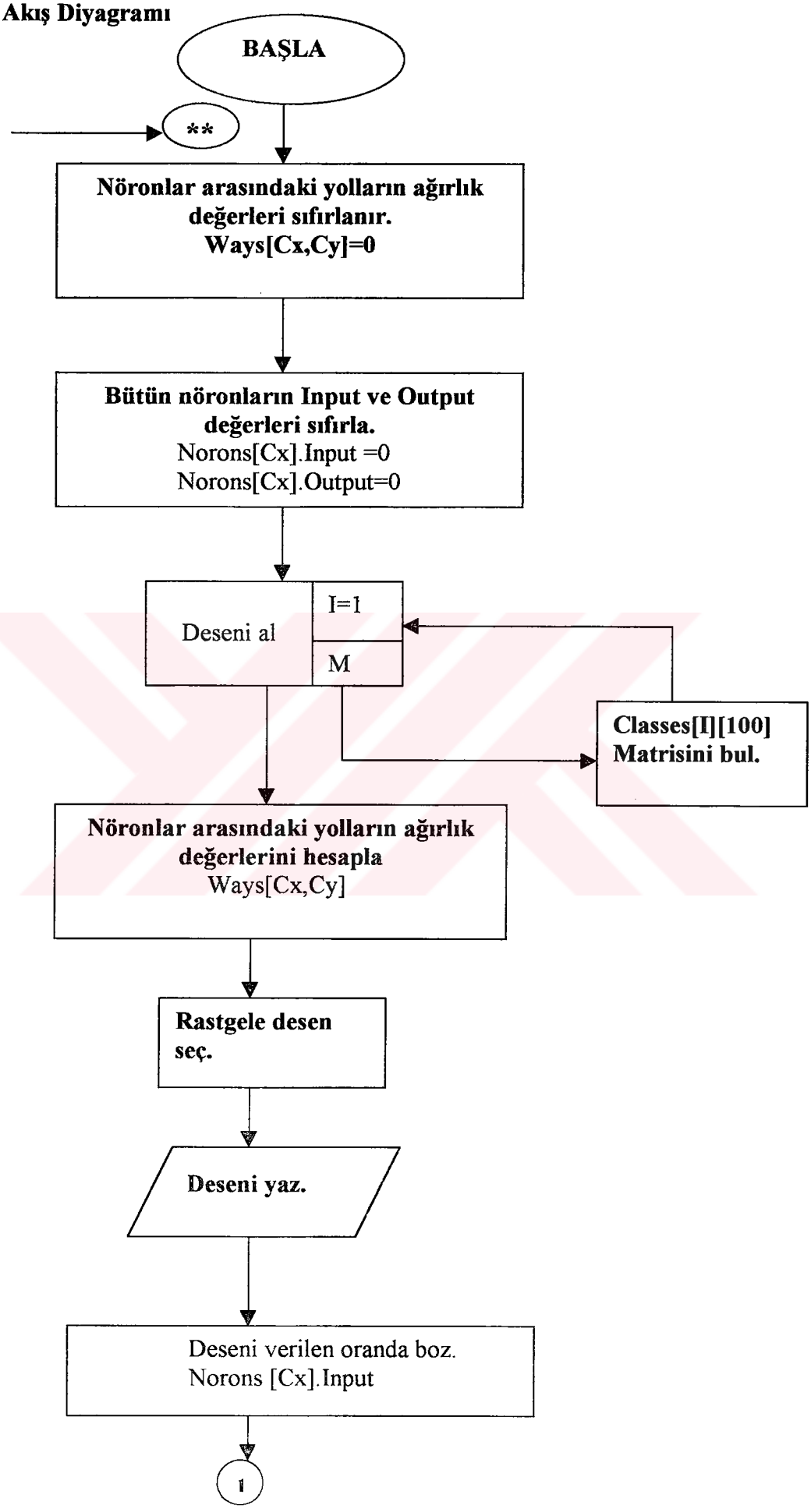
```
d1=bozukdesen;
önceki=d1;

repeat
SistemeGirdiVer(d1);  ( sisteme d1 deseni verildi.)
Calıstır;
CıktıyıAl(d1);  (sistemden yeni çıktının, doğruya daha yakın hali alındı ve d1'e konuldu.)
if önceki=d1 then çık=true
else
begin
çık=false;
önceki=d1; ( önceki çıktı değeri bundan sonra d1 olacak. d1 ise
(yani yeni çıkan değer) sisteme tekrar verilecek. Feedback.)
end;
until çık;

ulaşılacak değer = 12345
bozuk değer = 35124
```

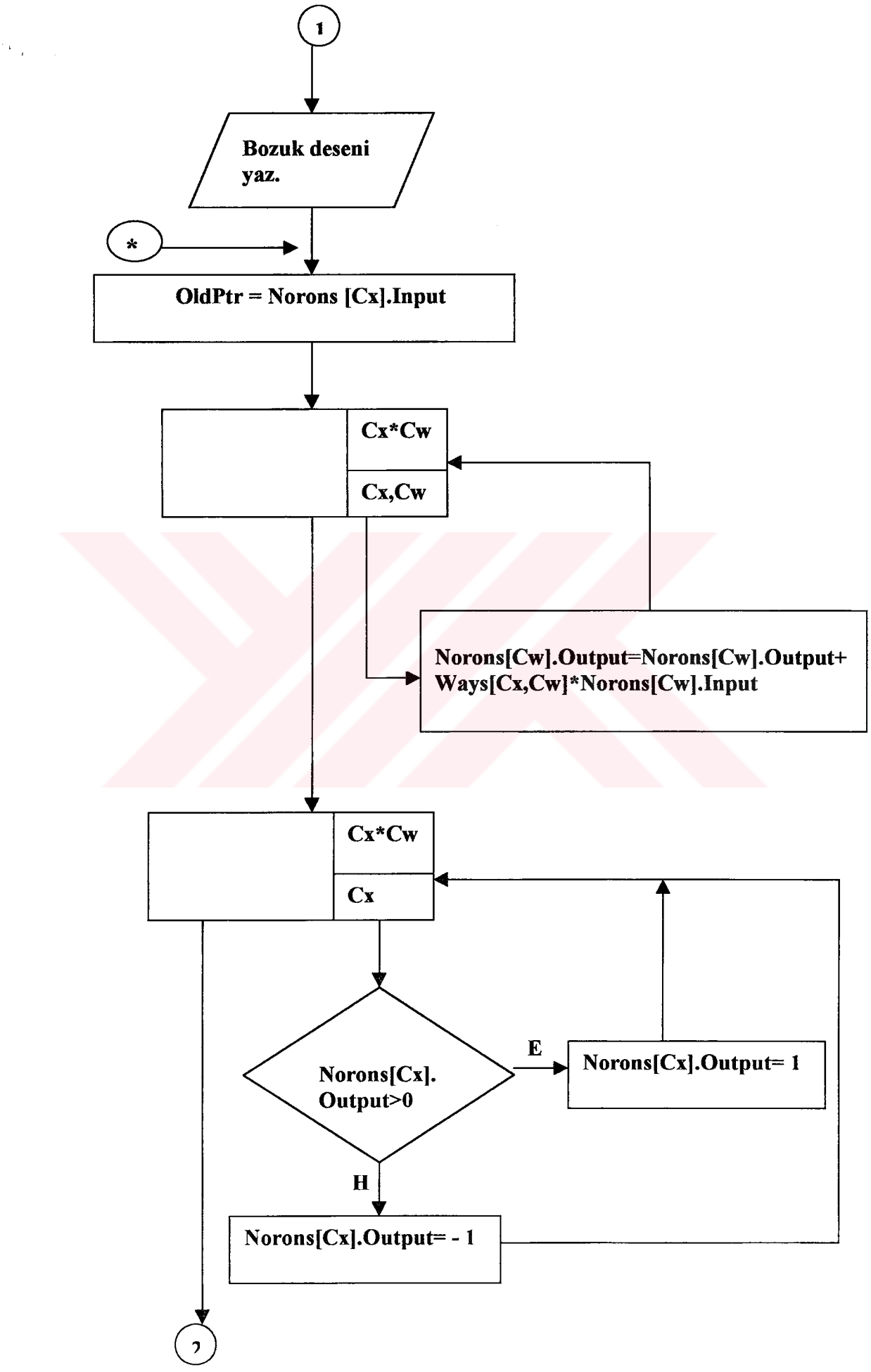
Giren Değer	Çıkan Değer	Durum
35124	21435	Farklı, devam et.
21435	12435	Farklı, devam et.
12435	12345	Farklı, devam et.
12345	12345	Eşit. Tamam.

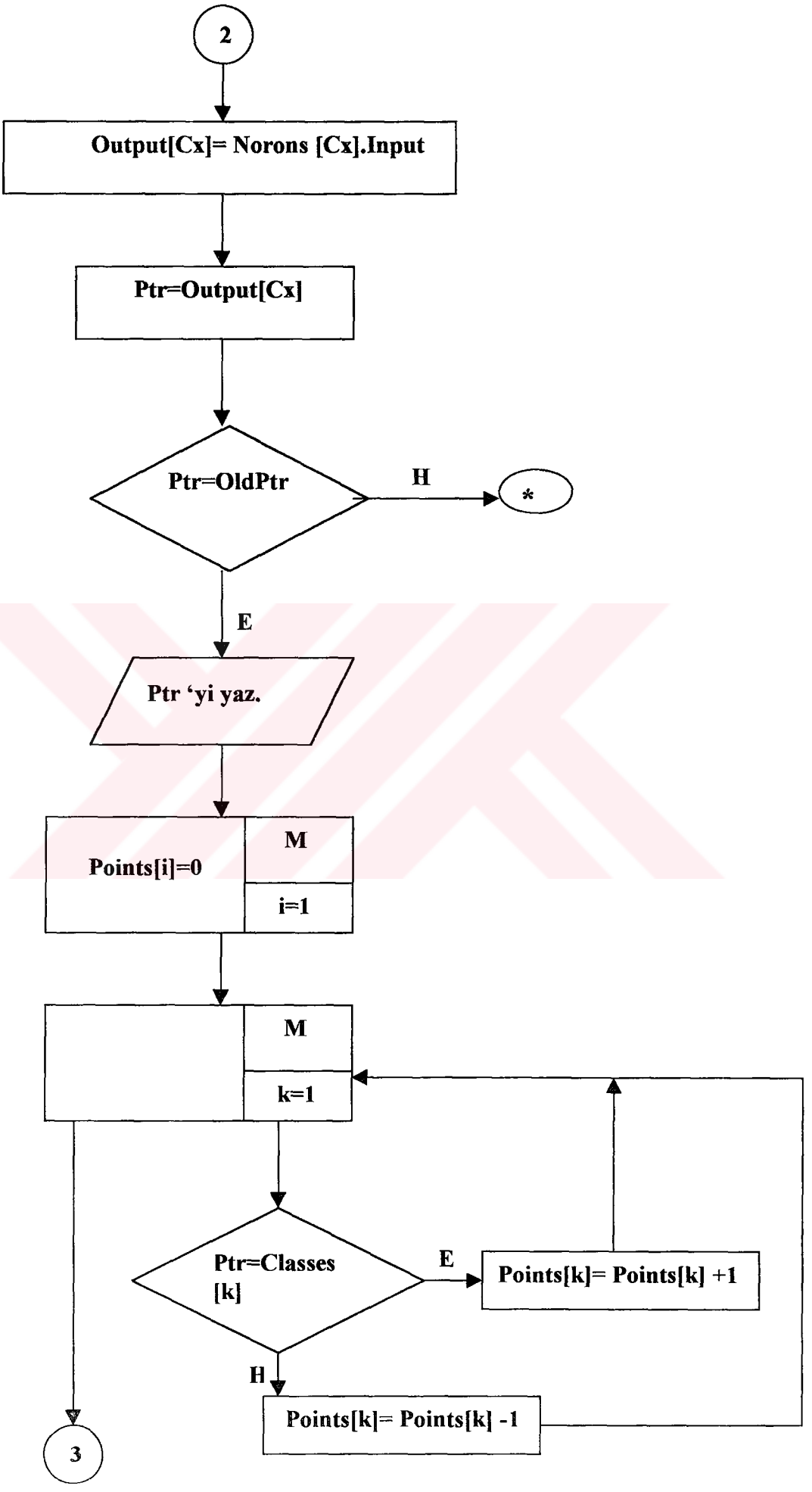
Hopfield, bozulmuş girdiyi doğru olana yakınsatabilesiye kadar işlemlerini devam ettirir. Sistemin verdiği output, bir önceki verdiği outputla aynı olunca artık daha fazla yakınsayamayacağı anlaşılır ve işlem biter. Sonra da çıkan sonucun hangi sınıfa girdiği GetMAxSimilar procedürü ile anlaşılır. Bir anlamda hopfield tam sonucu bulmaz, sadece girdiyi doğru olana yaklaştırır. Kısacası hopfield algoritmasının yaptığı iş, eğittiğimiz desenlerin (patternların) ağırlıklı ortalamasını bulup, bunları hücreler arası ağırlık katsayılarına paylaştırmak (bir bakıma ortalama değer oluşturmak) ve bizim verdiğimiz girdiyi (inputu) bu ağırlıktan geçirerek, doğru değere yakınsatmaktır.

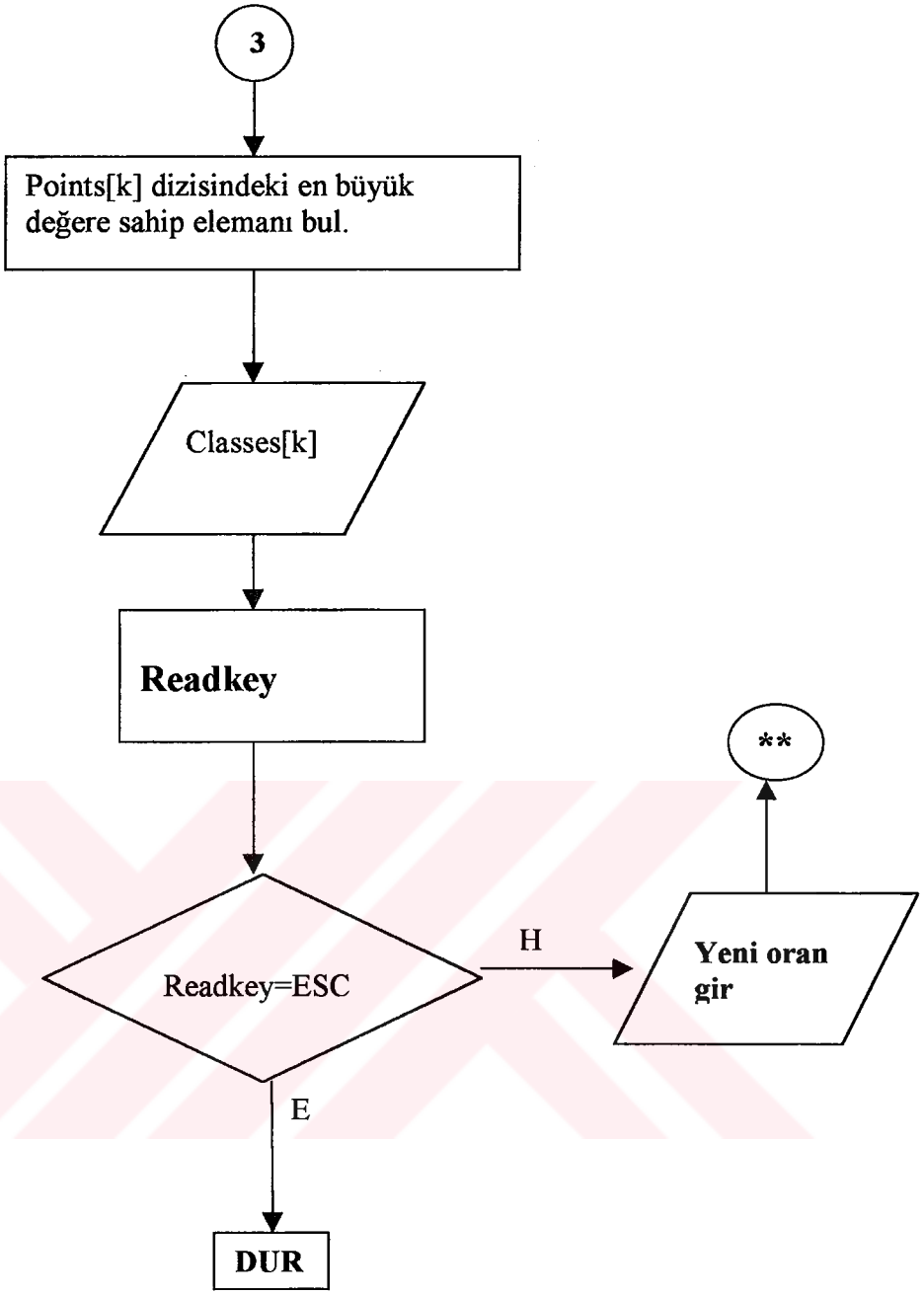
Bozma oranını 1/1 yaparsak, desenin 100% oranda bozulacağını belirtmiş oluruz. Bu da şeklin tam manasıyla tersine çevrileceğini gösterir.

4.2 Sistem Akış Diyagramı




1
Bozuk deseni yaz.
*
OldPtr = Norons [Cx].Input
Cx*Cw
Cx,Cw
Norons[Cw].Output=Norons[Cw].Output+ Ways[Cx,Cw]*Norons[Cw].Input
Cx*Cw
Cx
Norons[Cx]. Output>0
E
Norons[Cx].Output= 1
H
Norons[Cx].Output= - 1
2


2
Output[Cx]= Norons [Cx].Input
Ptr=Output[Cx]
Ptr=OldPtr
H
*
E
Ptr 'yi yaz.
Points[i]=0
M
i=1
M
k=1
Ptr=Classes [k]
E
Points[k]= Points[k] +1
H
Points[k]= Points[k] -1
3



Şekil 4.1 Sistem Akış Diyagramı

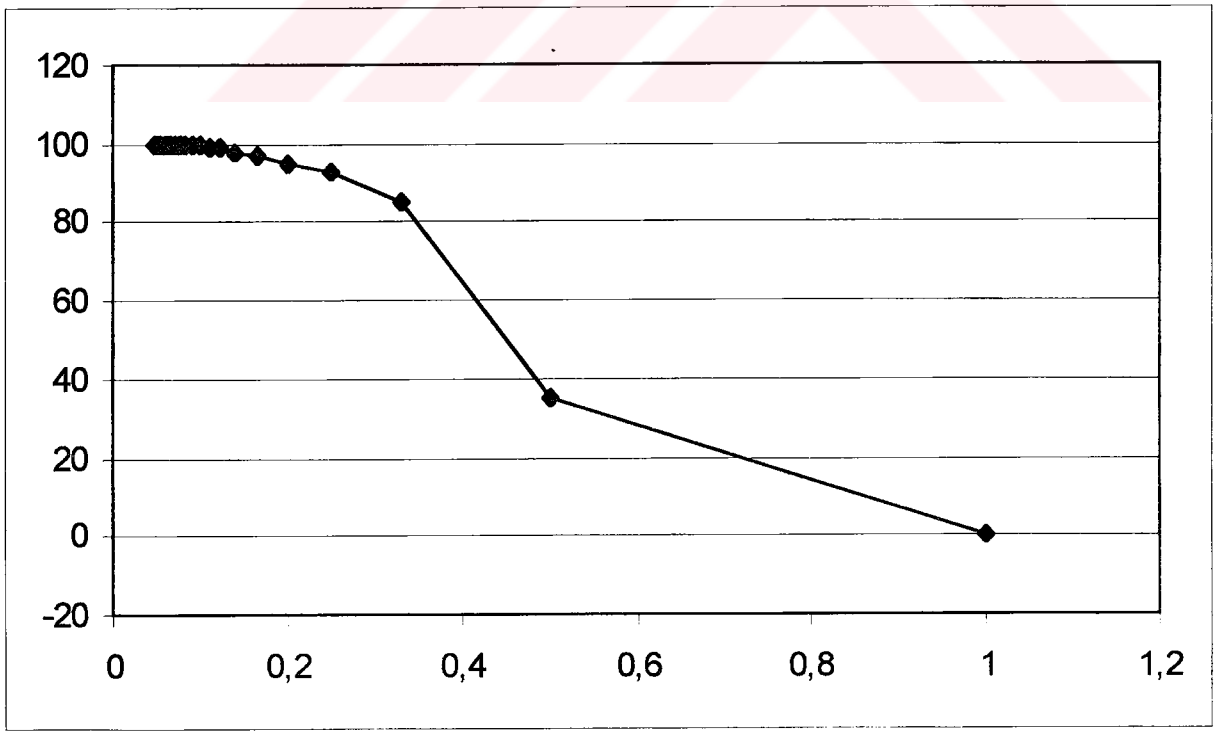
Program Ek 1' de ve programın çıktısı Ek2' de sunulmuştur.

Sample.Pas programı çalıştırılarak 100 deneme yapılmış, ve bu deneme sonucunda bozulma oranlarına karşılık gelen düzeltme oranları aşağıda gösterilmiştir. Bu değerlerin oluşturduğu düzeltme fonksiyonunun grafiği Şekilde 4.2 'de gösterilmiştir.

X: Bozulma Oranı

Y: Düzeltme Oranı

x	y
1,0	0
0,5	35
0,33	85
0,25	93
0,2	95
0,166	97
0,142	98
0,125	99
0,111	99
0,1	100
0,091	100
0,083	100
0,077	100
0,071	100
0,067	100
0,063	100
0,059	100
0,056	100
0,053	100
0,05	100



Şekil 4.2 Düzeltme Fonksiyonu

5. DÜZELTME FONKSİYONUNUN BULUNMASI

Bozma oranının bir fonksiyonu olarak düzeltme fonksiyonunun bulunması için SPSS 8.0 paket programının REGRESSION seçeneği kullanılmıştır. Yapılan denemeler sonunda düzeltme fonksiyonunun kübik bir eğri olduğu görülmüştür. SPSS paket programı çıktıları aşağıdaki gibidir.

Metod:	CUBIC
Bağımlı Değişken :	Y
Bağımsız Değişken :	X
Çoklu R:	0,99532
R kare :	0,9906
Düzeltilmiş R kare :	0,98891
Y' nin Standart Hatası :	2,70764

Yukarıda görüldüğü gibi R-kare, yani bağımlı değişkenin bağımsız değişken tarafından açıklanma oranı %99,06 bulunmuştur. Bu oranın çok yüksek olduğu görülmektedir. Fakat bulunan kübik fonksiyonun genel anlamlılığını ve parametrelerinin anlamlılığını istatistiki olarak test etmek için Varyans Analizi Tablosu hazırlanmıştır.

VARYANS ANALİZİ TABLOSU			
Hata Kaynağı	**Serbestlik Derecesi**	**Hata Kareleri Toplamı**	**Hata Kareleri Ortalaması**
Regresyondan	**3**	**12441,649**	**4147,2163**
Kalıntılardan	**16**	**117,301**	**7,3313**

İstatistiksel olarak , bulunan kübik modelin test edilmesi için F-testi uygulanmış ve F istatistiği hesaplanmıştır.

$F = \frac{4147,2163}{7,3313} = 565,68532$ bulunmuştur.

F Dağılımı tablosundan %99 anlam seviyesinde bulunan kritik değer 5,29 'dur.

$F_{kritik} < 565,68532$ olduğundan bulduğumuz model %99 olasılıkla doğrudur.

Denklemdeki katsayılar ve anlam seviyeleri aşağıdaki gibidir.

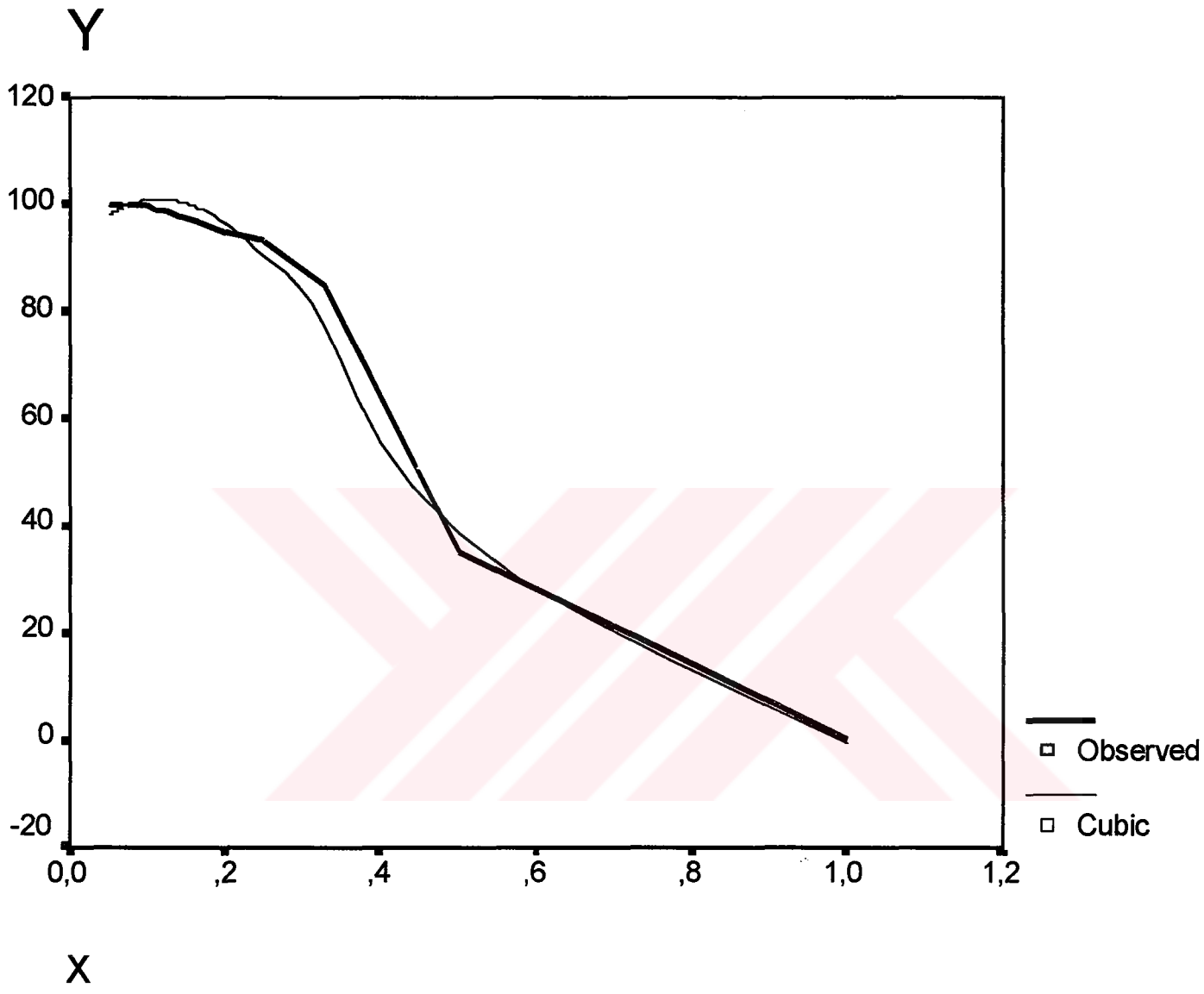
Değişkenler	Katsayılar	Katsayıların Standart Hatası	t değerleri	Anlam Seviyeleri
X	171,503242	29,336776	5,846	0,000
X^2	-846,080322	86,252639	-9,809	0,000
X^3	582,689875	60,142208	9,689	0,000
Regresyon Sabiti	91,694468	2,121899	43,213	0,000

Yukarıdaki tablodan görüldüğü gibi tüm katsayılar %99 güvenle , yapılan t testi sonucunda anlamlı bulunmuştur.

Analiz sonucunda bulunan kübik model şu şekildedir:

$Y = 91,694468 + 171,503242\ X - 846,080322\ X^2 + 582,689875\ X^3$

Gözlemlenen gerçek Y değerleri ve X değerleri bulunan modelde yerine konarak bulunan tahmini Y değerleri aşağıdaki grafikte gösterilmiştir.



Şekil 5.1 Gözlemlenen gerçek Y değerleriyle, analiz sonucu bulunan kübik fonksiyonun karşılaştırılması

Aşağıdaki tabloda ;

X: Bozma oranı

Y: Düzeltme oranı

$\hat{Y}$: Modelde X değerleri yerine konarak tahmin edilen Y değeri

Hata: Y- $\hat{Y}$ değerlerini göstermektedir.

X	Y	$\hat{Y}$	Hata
1,0000	0	-,19274	,19274
,5000	35	38,76224	-3,76224
,3300	85	77,09252	7,90748
,2500	93	90,79479	2,20521
,2000	95	96,81342	-1,81342
,1660	97	99,51481	-2,51481
,1420	98	100,65597	-2,65597
,1250	99	101,05043	-2,05043
,1110	99	101,10368	-2,10368
,1000	100	100,96668	-,96668
,0910	100	100,73397	-,73397
,0830	100	100,43376	-,43376
,0770	100	100,14982	-,14982
,0710	100	99,81466	,18534
,0670	100	99,56238	,43762
,0630	100	99,28678	,71322
,0590	100	98,98763	1,01237
,0560	100	98,74767	1,25233
,0530	100	98,49425	1,50575
,0500	100	98,22727	1,77273

6. YOĞUNLUK FONKSİYONUNUN BULUNMASI

y=g(x) denkleminde , rastgele değişkenin $f_y(y)$ yoğunluğunu, x'in $f_x(x)$ yoğunluğuna göre belirleyelim. x'in sürekli tipte ve g(x) 'in sürekli olduğunu ve herhangi bir aralığın üzerinde bir sabite eşit olmadığını varsayalım. Bu, verilen y için,

$$y=g(x) \tag{6.1}$$

denkleminin en fazla $x_1, x_2, \ldots$ köklerinin sayılabilir bir sayıya sahip olduğu anlamına gelir. Aşağıdaki teoremin; karışık tipteki rastgele değişkenler için de geçerli olduğunu ve yukarıdakini sağlamayan g(x) için, (6.4) ' daki belirli y için, (6.1) 'ün yalnızca x_n 'in çözümlerinin sayılabilir bir sayıya sahip olduğunu ve $F_x(x)$ 'in x_n noktalarında türetilebilir olduğunu gösterdiğine dikkat edelim. Böylece, $f_y(y)$ 'yi belirlemek için (6.4), önceden tartışılan metodlarla birleştirilebilir.

6.1 Temel Teorem

Verilen bir y değeri için $f_y(y)$ 'yi bulmak amacıyla, x için

$$y=g(x) \tag{6.2}$$

denklemini y'ye göre çözeriz. Eğer $x_1,\ldots,x_n,\ldots$ lerin hepsi reel kökler ve

$$y=g(x_1)=\ldots=g(x_n)=\ldots \tag{6.3}$$

ise,

$g'(x)=\dfrac{dg(x)}{dx}$ olduğunda

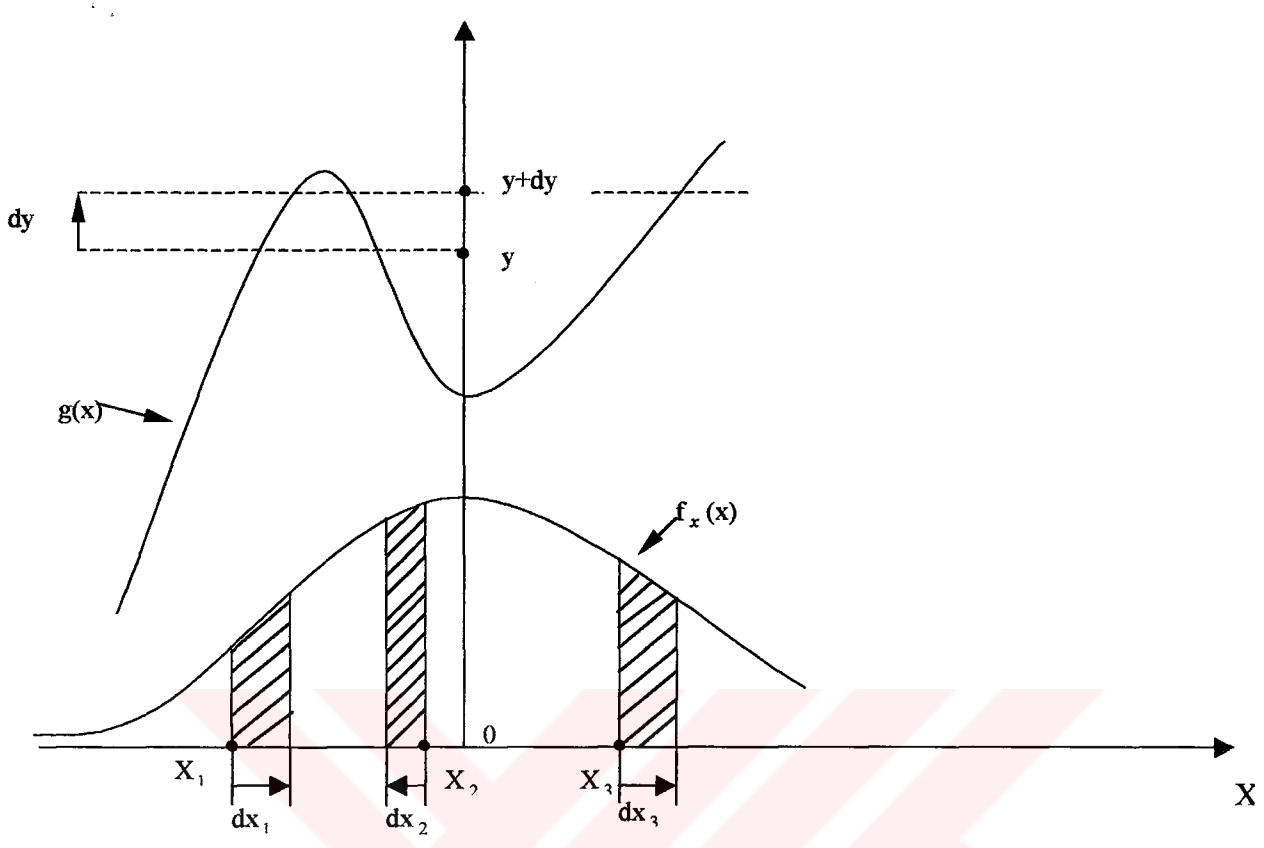
$$f_y(y)=\frac{f_x(x_1)}{|g'(x_1)|}+\ldots+\frac{f_x(x_n)}{|g'(x_n)|}+\ldots \tag{6.4}$$

olur.

Açıkça; $x_1,\ldots,x_n,\ldots$ sayıları y 'ye bağlıdır. Eğer, belirli bir y için y=g(x) denklemi hiçbir reel köke sahip değilse, $f_y(y)=0$ 'dır.

İSPAT: Genelleştirmeden kaçınmak amacıyla, verilen y için y=g(x) denkleminin Şekil 6.1 deki gibi x_1, x_2, x_3 şeklinde 3 kökü olduğunu varsayalım. O zaman

$f_y(y)dy=P\{y<y<y+dy\}$, $dy>0$ olur. (6.5)



Şekil 6.1 y=g(x) denkleminin kökleri

Böylece, $f_y(y)$'yi belirlemek amacıyla $y < g(x) < y + dy$ olacak şekilde x 'in bütün değerlerini bulmak yeterlidir. Şekil 6.1 'den görüldüğü gibi, $dx_1 > 0, dx_2 < 0, dx_3 > 0$ olduğunda $x_1 < x < x_1 + dx_1$, $x_2 + dx_2 < x < x_2$, $x_3 < x < x_3 + dx_3$ için yukarıdakiler doğrudur. Böylece

$P\{y < y < y+dy\}$

$= P\{x_1 < x < x_1 + dx_1\} + P\{x_2 + dx_2 < x < x_2\} + P\{x_3 < x < x_3 + dx_3\}$ (6.6)

dır.

Bu toplam Şekil 6.1'de taranmıştır.

Ayrıca

$$P\{x_1 < x < x_1 + dx_1\} = f_x(x_1)dx_1$$

$$P\{x_2 + dx_2 < x < x_2\} = f_x(x_2)dx_2$$

$$P\{x_3 < x < x_3 + dx_3\} = f_x(x_3)dx_3 \qquad (6.7)$$

dür.

Üstelik,

$$g'(x_i)dx_i = dy \qquad i = 1,2,3 \tag{6.8}$$

dir.

Böylece

$$f_y(y)dy = \frac{f_x(x_1)}{|g'(x_1)|}dy + \frac{f_x(x_2)}{|g'(x_2)|}dy + \frac{f_x(x_3)}{|g'(x_3)|}dy \tag{6.9}$$

olur. Ve bunu (6.4) izler. g(x) 'in herhangi bir benzer formu için de benzer şekilde uslamlandırılabilir.

6.2 Bozulma Oranının Düzgün Dağılım Olması Halinde Düzeltme Fonksiyonunun Beklenen Değerin Bulunması:

[1,1/20] aralığındaki bozulma oranı düzgün dağılım ise , f(x) düzeltme fonksiyonunun olasılık yoğunluk fonksiyonunun beklenen değerini bulalım.

$$f(x) = 582,7x^3 - 846,1x^2 + 171,5x + 91,7 \tag{6.10}$$

fonksiyonunun [1,1/20] aralığındaki bozulma oranı düzgün dağılım olduğundan dolayı

$$f_x(x) = \frac{1}{1 - 1/20} = \frac{20}{19} \tag{6.11}$$

olur.

Bu fonksiyonun olasılık yoğunluk fonksiyonu;

$$f_y(y) = \frac{f_x(x)}{|f'(x)|} = \frac{\frac{20}{19}}{|1748,1x^2 - 1692,2x + 171,5|} \tag{6.12}$$

olur.

Beklenen değerini bulmak için , f(x) denklemini y 'nin her değerine ayrı ayrı eşitleyip kökleri bulmamız gerekir. [1,1/20] aralığında bulduğumuz köklerle daha sonra $f_y(y)$ hesaplanacak ve buradan da beklenen değer bulunacaktır.

y= 0 için;

$582,7x^3 - 846,1x^2 + 171,5x + 91,7 = 0$ denkleminin [1,1/20] aralığındaki kökleri

$x_1 = 0,6817$, $x_2 = 1$

olarak bulunur.

$x_1 = 0,6817$ için olasılık yoğunluk fonksiyonu ;

$$f_y(y)=\frac{f_x(x)}{|f'(x)|}=\frac{\frac{20}{19}}{|1748,1(0,6817)^2-1692,2(0,6817)+171,5|}=\ 0,0062 \quad \text{ve} \qquad (6.13)$$

x_2=1 için olasılık yoğunluk fonksiyonu ;

$$f_y(y)=\frac{f_x(x)}{|f'(x)|}=\frac{\frac{20}{19}}{|1748,1-1692,2+171,5|}=\ 0,0046 \qquad (6.14)$$

olduğundan

y=0 için $f_y(y)=0,0062+0,0046=0,0108$ 'dir. (6.15)

y=35 için;

$582,7x^3-846,1x^2+171,5x+91,7=35$ denkleminin [1,1/20] aralığındaki kökü

x=0,515877 olarak bulunur.

Bu kökü alırsak, olasılık yoğunluk fonksiyonu;

$$f_y(y)=\frac{f_x(x)}{|f'(x)|}=\frac{\frac{20}{19}}{|1748,1(0,515877)^2-1692,2(0,515877)+171,5|}=0,0045 \qquad (6.16)$$

olduğundan

y=35 için $f_y(y)$=0,0045 'dir.

Diğerlerini de benzer şekilde çözersek;

y=85 için $582,7x^3-846,1x^2+171,5x+91,7=85$ denkleminin [1,1/20] aralığındaki kökü

x=0,287018 , $f_y(y)=0,0061$

y=93 için $582,7x^3-846,1x^2+171,5x+91,7=93$ denkleminin [1,1/20] aralığındaki kökü

x=0,233751, $f_y(y)$=0,0084

y=95 için $582,7x^3 - 846,1x^2 + 171,5x + 91,7 = 95$ denkleminin [1,1/20] aralığındaki kökü

x=0,217244 , $f_y(y)$=0,01

y=97 için $582,7x^3 - 846,1x^2 + 171,5x + 91,7 = 97$ denkleminin [1,1/20] aralığındaki kökü

x=0,198101, $f_y(y)$=0,01

y=98 için $582,7x^3 - 846,1x^2 + 171,5x + 91,7 = 98$ denkleminin [1,1/20] aralığındaki kökleri

x_1=0,0475033

x_2=0,186924, $f_y(y)$=0,023

y=99 için $582,7x^3 - 846,1x^2 + 171,5x + 91,7 = 99$ denkleminin [1,1/20] aralığındaki kökleri

x_1=0,0590914

x_2=0,173916, $f_y(y)$=0,024

y=99 için $582,7x^3 - 846,1x^2 + 171,5x + 91,7 = 99$ denkleminin [1,1/20] aralığındaki kökleri

x_1=0,0590914

x_2=0,173916 , $f_y(y)$=0,024

y=100 için $582,7x^3 - 846,1x^2 + 171,5x + 91,7 = 100$ denkleminin [1,1/20] aralığındaki kökleri

x_1=0,0741097

x_2=0,157486, $f_y(y)$=0,03 olarak bulunur.

Sonuç olarak;

y	$f_y(y)$
0	0,0108
35	0,0045
85	0,0061
93	0,0084
95	0,01
97	0,01

y	$f_y(y)$
98	0,023
99	0,024
99	0,024
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03
100	0,03

bulunmuş ve fonksiyonun beklenen değeri

$$E(y) = \Sigma y\, f_y(y)$$

$$= 0+35(0,0045)+85(0,0061)+93(0,0084)+95(0,01)+97(0,01)+98(0,0023)+$$
$$+99(0,024)+99(0,024)+100(0,03)+30$$

$$= 43,3832 \qquad (6.17)$$

olarak hesaplanmıştır.

7. SONUÇ

Uygulamanın sonucunda girilen bozulma oranlarına karşılık gelen düzeltme oranları bulunmuş ve SPSS 8.0 kullanılarak düzeltme fonksiyonu hesaplanmıştır. Bu düzeltme fonksiyonunun yoğunluk fonksiyonu bulunmuş ve son olarak da bu bozulma oranının düzgün dağılım olması halinde olasılık yoğunluk fonksiyonunun beklenen değeri hesaplanmıştır. Sonuç olarak beklenen değer E(y)=43,3832 bulunmuştur. Bu da sisteme verilen bozuk desenin sistemden çıktıktan sonra gerçek desene benzeme oranının ortalama %43,3832 olduğunu gösterir.

KAYNAKLAR

Simpson, P.K., (1990), Artificial Neural Systems, Pergaman Press, 1990.

Haykin, Simon, (1998), Neural Networks, A Comprehensive Foundation, Canada.

Yıldızdoğan, M., (1995), İstanbul Teknik Üniversitesi YLT, Yapay Sinir Ağı ile Sistem Tanıma.

Zngram, Ben, Hurst, A., Naveen T., Pattern Recognition Using Neural Networks.

Comango, A., Learning Algorithms in Neural Networks, December 12 1990, Newyork Columbia University Computer Science Department.

Sawhney, Nitin, Shaun, Abrahamson, March 14 1997, Neural Networks Introduction and Applications.

Chen, Kenny, (1998), Image Recognition System.

Chen, Kenny, (1998), Introduction to Neural Networks.

Swinger, Kevin, (1994),Appliying Neural Networks.

Gerson, David G., Neural Networks.

http: // www. geocities.com / Athens.

http: // www. mit.edu/pgifford/nn/neural.html,Moscachussetts Institute of Technology.

Hopfield, J.J., "Neurons with Graded Response have Collective Computational Properties Lile those of two-state Neurons", USA, May 1984, Vol 81,3088-3092.

Lippman, Richadr P., "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, April 1997, pp 4-22.

Wolters, Pratzel, (1999), Neural Networks and Heart Risk Analysis, The Abdussalam International Central of Theoretical Physics, Italy.

Aytaç, Prof. Dr. Mustafa, (1994), Matematiksel İstatistik, Uludağ Üniv. Basımevi.

Üstün, S.Vakkos, (1997), YSA kullanılarak Türkçedeki Sesli Harflerin Tanınması YL. Tezi, YTÜ.

Gülseçen, Yrd. Doç. Sevinç, (1999), Akıllı Karar Sistemleri, İstanbul Üniversitesi Fen Fakültesi Bilgisayar Bölümü Astronomi ve Uzay Bilimleri Bölümü

EK 1

EK1 Sample.Pas

```
Uses Crt,Patterns;

Var
 Ways:TWays;
 Ch:Char;
 Ptr:TPattern;

procedure InitWays;
Var
 Cx,Cy:Integer;
begin
 For Cx:=1 to wX*wY do
    For Cy:=1 to wX*wY-1 do
       Ways[Cx,Cy]:=0;
end;

procedure InitNorons;
Var
 Cx:Integer;
begin
 For Cx:=1 to wX*wY do
    begin
     Norons[Cx].Input:=0;
     Norons[Cx].Output:=0;
    end;
end;

procedure TrainSystem;
var
 i,j,WClass,Sum:Integer;
Begin
 For i:=1 to wX*wY do
  For j:=1 to wX*wY do
   begin
   Sum:=0;
    If i<>j then
    begin
     For WClass:=1 to M do
        Sum:=Sum+Classes[WClass][i]*Classes[WClass][j];
    end;
```

{ Burada her bir yolun değeri (weight) hesaplanır. (i. hücreden j. hücreye giden yolun değeri hesaplanır.) Örneğin 2. hücreden 5. hücreye giden yolun değerinin hesaplanması için; bütün sınıfların (M) i.(2) ve j.(5) hücresinin değerleri (ya 1 yada -1'dir) çarpılır ve Sum'a eklenir. Sonra da çıkan sonuç (sum) Ways[2(i),5(j)]'ye eşitlenir. Ways, sadece ağırlık değerlerini tutar. }

```
Ways[i,j]:=Sum;
  end;
end;

procedure CalculateNorons;
Var
 Cx,Cw:Integer;
begin
 For Cx:=1 to wX*wY do
    begin
     For Cw:=1 to wX*wY do
        begin
         If Cx<>Cw then
            Norons[Cw].Output:=Norons[Cw].Output+Ways[Cx,Cw]*Norons[Cx].Input;
        end;
    end;
```

{ Burada sisteme verilen bir veriyi inceliyor. Yaptığı iş; ilgili hücrenin outputunu, aradaki yolların ağırlık değerleri ile, giren (input) değerini çarparak elde ettiği değeri yine ilgili hücrenin output değerine ekleyerek elde etmektir. Bu işlemi bütün yollar için yapıyor. Bu aşamadan önce SetInput diyerek sisteme düzelteceği veriyi vermek gerekir.}

```
end;

procedure HardLimitFunction;
Var
 Cx:Integer;
begin
 For Cx:=1 to wX*wY do
    begin
     If Norons[Cx].OutPut>0 then Norons[Cx].Output:=1 else
       Norons[Cx].OutPut:=-1;
    end;
end;

procedure SetInput(Input:TPattern);
Var
 Cx:Integer;
begin
 For Cx:=1 to wX*wY do
    begin
     Norons[Cx].Input:=Input[Cx];
     Norons[Cx].Output:=0;
    end;
 { Burada sisteme input geliyor. Hücrelerin input değerleri ayarlanıyor ve yeni
  işlem için output değerleri sıfırlanıyor.
 }
end;
```

```
procedure GetOutput(var Output:TPattern);
Var
 Cx:Integer;
begin
 For Cx:=1 to wX*wY do
    Output[Cx]:=Norons[Cx].Output;
end;

procedure GenerateRandomInput(Orj:TPattern;Var Unknown:TPattern);
var
 k: integer;
begin
 Randomize;
 For k:=1 to wX*wY do UnKnown[k]:=Orj[k];
    For k:=1 to wX*wY do
       begin
        Orj[k]:=Unknown[k];
        if Random(Ratio)=0 then
          UnKnown[k]:=UnKnown[k]*-1;
    end;
end;

procedure InitProgram;
begin
 clrscr;
 Randomize;
 write('Initing Ways & Norons...');
 InitWays;
 InitNorons;
 writeln('Done');
 InitGuzelA(1);
 InitO(2);
 InitX(3);
 InitW(4);
 write('Training...');
 TrainSystem;
 writeln('Done.');
 writeln('To Exit press ESC...');
end;



Procedure GiveMaxSimilar(Inp:TPattern;Var Out:TPattern);
Var
 i,j,k:Integer;
 Points:Array [1..M] of Integer;
Begin
 For i:=1 to M do Points[i]:=0;
 For k:=1 to M do
  For i:=1 to wY do
```

```
  For j:=1 to wX do
   If Inp[(i-1)*wY+j]=Classes[k][(i-1)*wY+j] then inc(Points[k])
     else Dec(Points[k]);
 j:=-32000;
 For i:=1 to M do If Points[i]>j then begin k:=i; j:=Points[i]; end;
 For i:=1 to wY do
  For j:=1 to wX do Out[(i-1)*wY+j]:=Classes[k][(i-1)*wY+j];
End;

procedure Test;
Var
 OldPtr:TPattern;
 Equal:Boolean;
 Cx,Cy,Rnd:Integer;
begin
 Clrscr;
 Rnd:=Random(M)+1;
 WritePattern(18,2,Classes[Rnd]);
 GenerateRandomInput(Classes[Rnd],Ptr);
 { Rnd'de hangi sınıfın bozulacağı var. Random'la seçiliyor. Ve bozulmuş değer Ptr'ye
aktarılıyor.}
 WritePattern(32,2,Ptr);
 Gotoxy(1,3); Write('(1-a)');
 Gotoxy(1,4); Write('Seçilen harfin');
 Gotoxy(1,5); Write('düzgün hali.');
 Gotoxy(1,8); Write('(1-b)');
 Gotoxy(1,9); Write('Harfin');
 Gotoxy(1,10); Write('(desenin)');
 Gotoxy(1,11); Write('Bozulmus Hali');
 Gotoxy(22,13); Write('(a)');
 Gotoxy(36,13); Write('(b)');
 Equal:=False;
 Cy:=0;

 repeat
 OldPtr:=Ptr;
 {Başlıyor.}
 SetInput(Ptr);
 CalculateNorons;
 HardLimitFunction;
 GetOutput(Ptr)
{Bitiyor. Artık elimizde sistemin outputu var. Değer yine Ptr'de bulunuyor.}

Equal:=True;
 for cx:=1 to wX*wY do
    If Ptr[cx]<>OldPtr[cx] then
      begin
       Equal:=False;
       Break;
      end;
```

{ Sistemin bir önceki değerlendirmede verdiği sonuç ile şimdiki (yenisi) karşılaştırılıyor. Bu iki değer aynı olasıya kadar bu işlemler yineleniyor. Equal True ise OldPtr(bir önceki) ile Ptr(Yenisi) aynı demektir ve döngüden çıkıyor. Cy'de iterasyon miktarı saklanıyor.}

```
 inc(Cy);
 until Equal;
 Gotoxy(45,7); Write('(2)');
 Gotoxy(45,8); Write('Harfin (desenin)');
 Gotoxy(45,9); Write('Duzelmis Hali --->');
 Gotoxy(45,10); Write('(Hopfield"dan');
 Gotoxy(45,11); Write('geçmiş hali)');
 WritePattern(65,2,Ptr);

 GiveMaxSimilar(Ptr,Ptr);
 Gotoxy(45,18); Write('(3)');
 Gotoxy(45,19); Write('GetMaxSimilar');
 Gotoxy(45,20); Write('procedure"unden');
 Gotoxy(45,21); Write('geçtikten sonra -->');
 WritePattern(65,14,Ptr);

 gotoxy(1,16); Write(wX*wY,' Neurons created.');
 gotoxy(1,17); Write(wX*wY*wX*wY,' ways initialized.');
 gotoxy(1,18); Write('Toplam ',M,' sınıf var.');
 gotoxy(1,19); Write('1/',Ratio,' oranında bozulma.');
 gotoxy(1,20); Write('Toplam ',Cy,' iterasyondan sonra.');
 gotoxy(1,21); Write('Bozulma oranını değiştirmek');
 gotoxy(1,22); Write('       için r tuşuna basın.');
 gotoxy(1,23); Write('çıkmak için ESC tuşuna basın.');
end;

procedure ReadNewRatio;
Var
 St:String;
 Err,NewRatio:Integer;
begin
 Clrscr;
 Writeln('Lütfen yeni oranı giriniz (1-20) :');
 Readln(St);
 Val(St,NewRatio,Err);

 If (Err>0) or (NewRatio>20) or (NewRatio<1) then
   writeln('Yanlış giriş... İşlem iptal edildi.')
   else
   begin
    Ratio:=NewRatio;
    writeln('Bozulma oranı değiştirildi.');
   end;

 writeln('Devam etmek için bir tuşa basın.');
 readkey;
end;
begin
```

```
InitProgram;
repeat
  Test;
  Ch:=Readkey;
  If Ch='r' then ReadNewRatio;
 until Ch=#27;
end.
```

EK 2

EK2 Patterns.Pas

```
Unit Patterns;
interface
Uses Crt;
Const
 wX=10;
 wY=10;
 M=4;
 Filled='2';
 Ratio :Integer = 4;

Type
 TWays = Array [1..wX*wY,1..wX*wY] of Integer;
 { Burada hücreler arası yolların tanımlanır. Ways[1,2] deyince, 1. hücreden 2.hücreye giden
yolun ağırlık değerini ifade etmiş oluyoruz. wX*wY toplam hücre adetidir.}

 TNoron = record
        Input,Output:Integer;
 end;
 { Bir nöronun tanımlaması yapılıyor. }

 TCharPattern = Array [1..wX,1..wY] of Char;
 { Geçici bir değişken tipi.}

 TNorons  = Array [1..wX*wY] of TNoron;
 { Noronların tanımlanması. Norons[2]-> 2. nöron. Veya Norons[4].Output'da 4. nöronun
output'u.}

 TPattern = Array [1..wX*wY] of ShortInt;
 { Genel desen tanımı. wX*wY'lik bir desen tanımı. Değeri sadece 1 veya -1 olabilir. Doluysa
1 boşsa -1. }

 TClasses = Array [1..M] of TPattern;
 { Bütün sınıfların (harflerin) tutulduğu değişken tipidir. M; sınıf adetidir.. Bu dizi Tpattern
cinsinden tanımlanmıştır. Classes[2][5] deyince 2. sınıfın (harfin) 5. pixelini kastetmiş
oluyoruz.}

Var
 Classes:TClasses;
 Norons:TNorons;

{ Init'lerle harflerin tanımlamalarını Classes değişkenine aktarıyoruz. Yani harflerin
tanımlamalarını bu prosedürler ile yapıyoruz. WritePattern'da ilgili deseni ekrana
yazdırıyoruz. }
procedure InitO(WClass:Integer);
procedure InitX(WClass:Integer);
procedure InitGuzelA(WClass:Integer);
procedure InitW(WClass:Integer);
procedure WritePattern(x,y:Byte;Pattern:TPattern);
```

```
implementation

procedure InitO(WClass:Integer);
Var
 Pattern:TCharPattern;
 Cx,Cy:Integer;

begin
 Pattern[1] :='        ';
 Pattern[2] :=' 222222 ';
 Pattern[3] :='2      2';
 Pattern[4] :='2      2';
 Pattern[5] :='2      2';
 Pattern[6] :='2      2';
 Pattern[7] :='2      2';
 Pattern[8] :='2      2';
 Pattern[9] :=' 222222 ';
 Pattern[10]:='        ';
 For Cy:=1 to wY do
    For Cx:=1 to wX do
       begin
        If Pattern[Cy,Cx]=Filled then Classes[WClass][Cx+((Cy-1)*wY)]:=1
          else Classes[WClass][Cx+((Cy-1)*wY)]:=-1;
       end;
{ Desenin o karakterinin dolu yada boş olmasına göre Classes'taki değer update ediliyor.
Doluysa 1, boşsa -1 }
end;

procedure InitX(WClass:Integer);
Var
 Pattern:TCharPattern;
 Cx,Cy:Integer;
begin

 Pattern[1] :='        ';
 Pattern[2] :='2      2';
 Pattern[3] :='2      2';
 Pattern[4] :='2      2';
 Pattern[5] :=' 2    2 ';
 Pattern[6] :='  2  2  ';
 Pattern[7] :=' 2    2 ';
 Pattern[8] :='2      2';
 Pattern[9] :='2      2';
 Pattern[10]:='2      2';
```

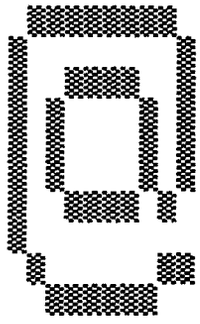
```
 For Cy:=1 to wY do
    For Cx:=1 to wX do
       begin
        If Pattern[Cy,Cx]=Filled then Classes[WClass][Cx+((Cy-1)*wY)]:=1
          else Classes[WClass][Cx+((Cy-1)*wY)]:=-1;
       end;
end;

procedure InitGuzelA(WClass:Integer);
Var
 Pattern:TCharPattern;
 Cx,Cy:Integer;
begin
 Pattern[1] :=' 22222222 ';
 Pattern[2] :='2       2';
 Pattern[3] :='2  2222  2';
 Pattern[4] :='2 2    2 2';
 Pattern[5] :='2 2    2 2';
 Pattern[6] :='2 2    2 2';
 Pattern[7] :='2  2222 2 ';
 Pattern[8] :='2       ';
 Pattern[9] :=' 2      22';
 Pattern[10]:=' 222222 ';
 For Cy:=1 to wY do
    For Cx:=1 to wX do
       begin
        If Pattern[Cy,Cx]=Filled then Classes[WClass][Cx+((Cy-1)*wY)]:=1
          else Classes[WClass][Cx+((Cy-1)*wY)]:=-1;
       end;
end;

procedure InitW(WClass:Integer);
Var
 Pattern:TCharPattern;
 Cx,Cy:Integer;
begin
 Pattern[1] :='       ';
 Pattern[2] :='       ';
 Pattern[3] :='       ';
 Pattern[4] :='       ';
 Pattern[5] :='       ';
 Pattern[6] :='2      2';
 Pattern[7] :='2   2  2';
 Pattern[8] :='2   2  2';
 Pattern[9] :=' 2 2 2 2';
 Pattern[10]:=' 22  22 ';
```

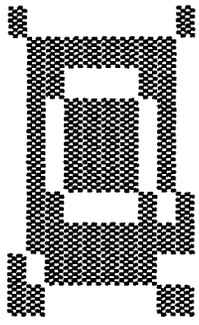
```
 For Cy:=1 to wY do
     For Cx:=1 to wX do
       begin
         If Pattern[Cy,Cx]=Filled then Classes[WClass][Cx+((Cy-1)*wY)]:=1
           else Classes[WClass][Cx+((Cy-1)*wY)]:=-1;
        end;
end;
procedure WritePattern(x,y:Byte;Pattern:TPattern);
Var
 Cx,Cy:Integer;
begin
 gotoxy(x,y);
 For Cy:=1 to wY do
    For Cx:=1 to wX do
       begin
        Gotoxy(x+Cx-1,y+Cy-1);
        If Pattern[Cx+((Cy-1)*wY)]=1 then write(Filled)
          else write(' ');
       end;
end;

begin
end.
```

EK 3

(1-a)
Seçilen harfin
düzgün hali.

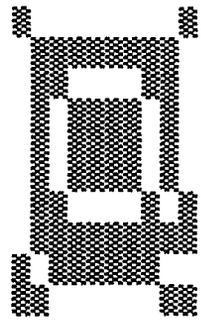
(1-b)
Harfin
(desenin)
Bozulmus Hali





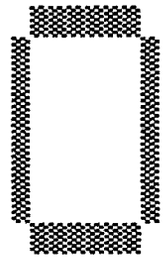
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



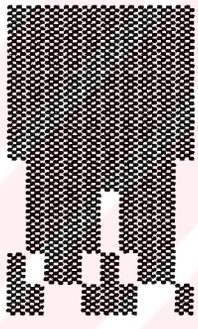
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/1 oranìnda bozulma.
Toplam 1 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



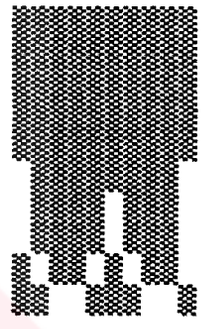
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



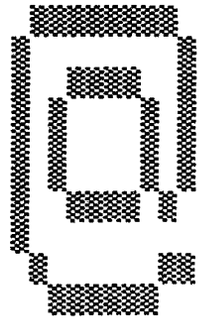
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



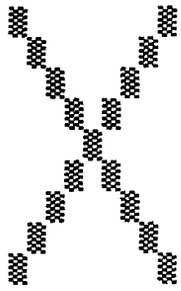
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/1 oranìnda bozulma.
Toplam 1 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

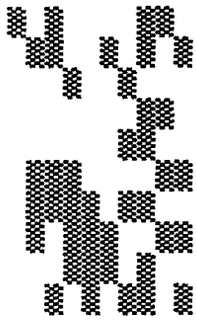


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

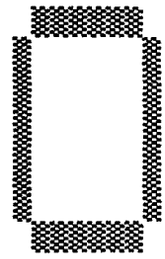


(a)



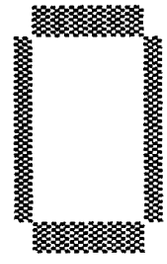
(b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/2 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

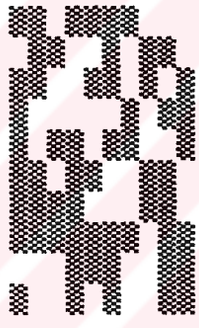
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

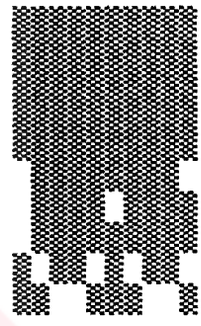
(1-b)
Harfin
(desenin)
Bozulmus Hali

(a)



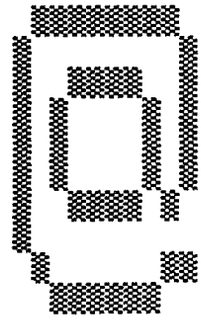
(b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



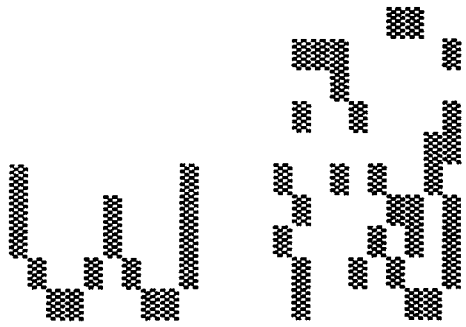
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/2 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



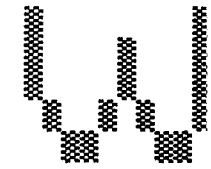
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



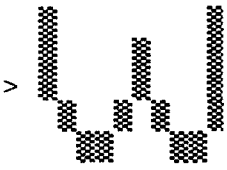
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



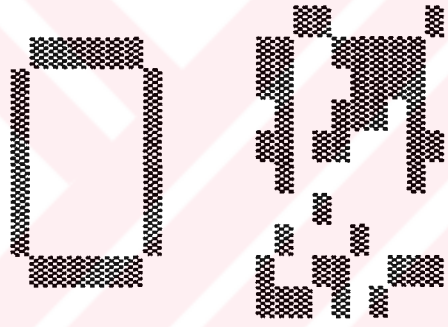
100 Neurons created.
10000 ways initialized.
Toplam 4 sınıf var.
1/3 oranında bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranını de°i*ş*tirmek
için r tu*ş*una basın.
Çıkmak için ESC tu*ş*una basın.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



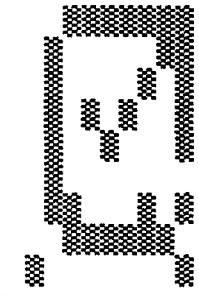
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



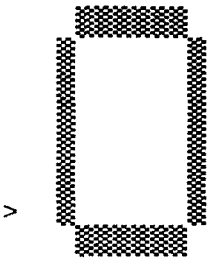
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



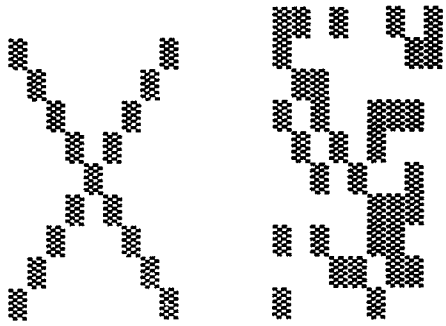
100 Neurons created.
10000 ways initialized.
Toplam 4 sınıf var.
1/3 oranında bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranını de°i*ş*tirmek
için r tu*ş*una basın.
Çıkmak için ESC tu*ş*una basın.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



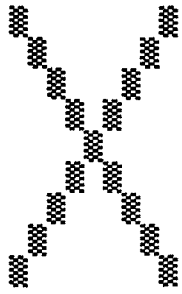
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



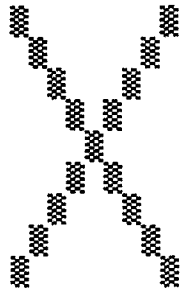
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



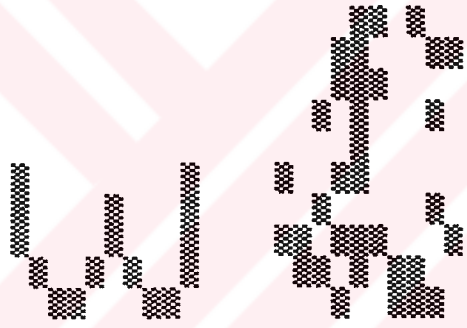
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/4 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



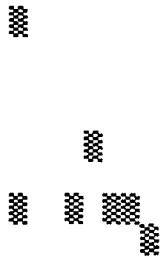
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)

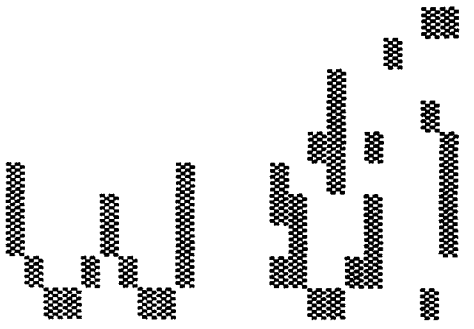


100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/4 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

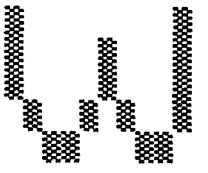
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



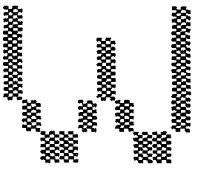
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



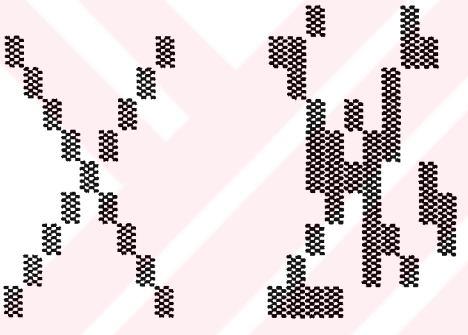
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/5 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



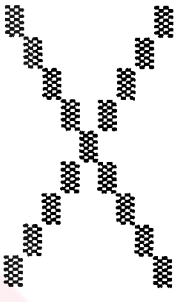
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



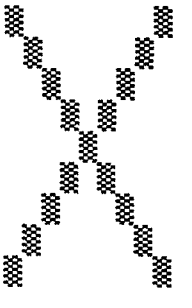
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/5 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

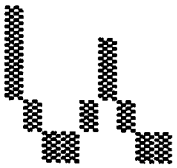


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

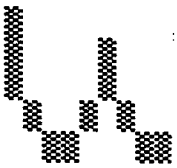
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



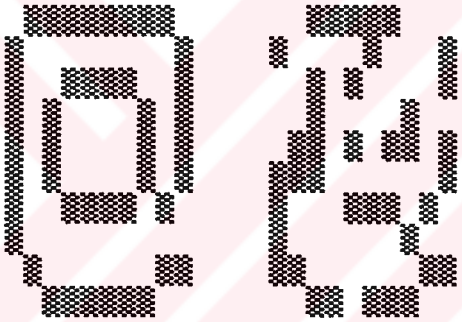
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/6 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



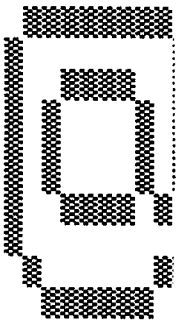
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



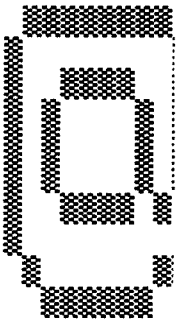
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



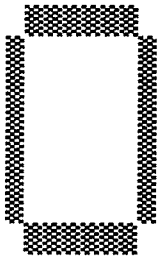
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/6 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

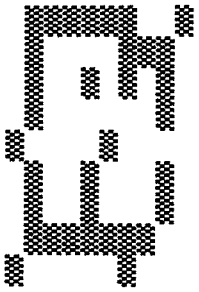
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

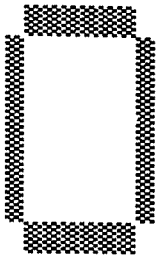
(1-b)
Harfin
(desenin)
Bozulmus Hali





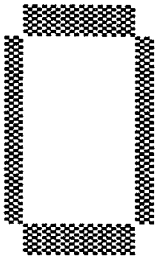
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



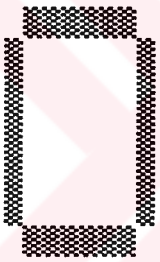
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/7 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

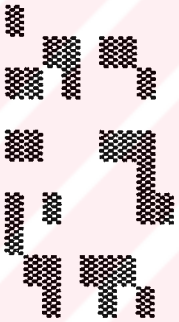
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

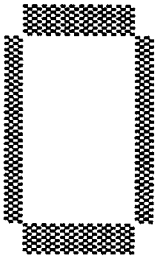
(1-b)
Harfin
(desenin)
Bozulmus Hali





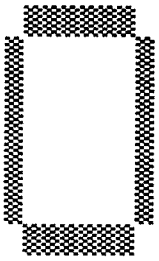
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/7 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

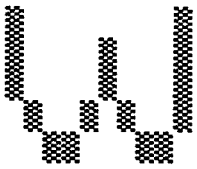


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

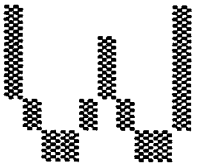
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



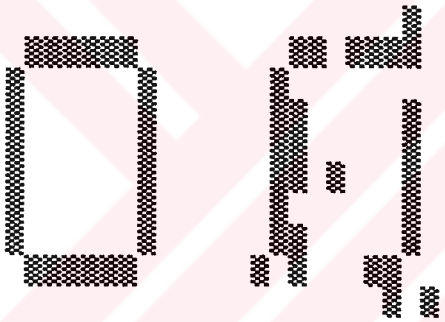
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/8 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



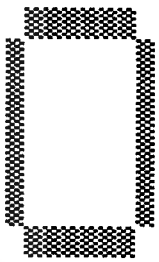
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



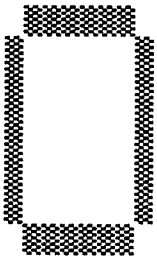
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



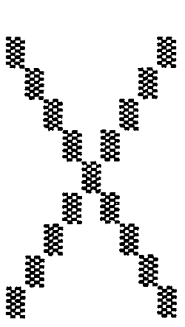
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/8 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

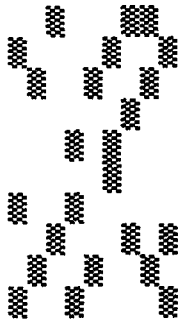
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

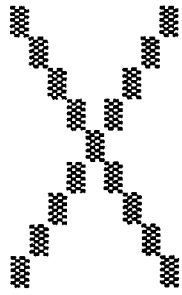
(1-b)
Harfin
(desenin)
Bozulmus Hali





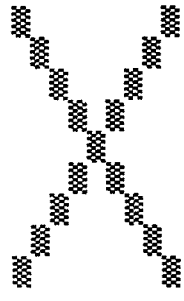
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ƒ* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/9 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ƒ*tirmek
için r tu*ƒ*una basìn.
Çìkmak için ESC tu*ƒ*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

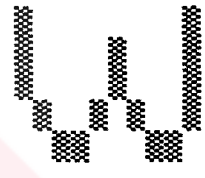


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

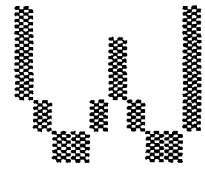
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ƒ* hali)



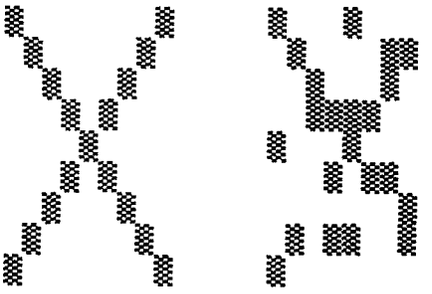
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/9 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ƒ*tirmek
için r tu*ƒ*una basìn.
Çìkmak için ESC tu*ƒ*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



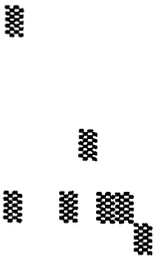
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/10 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

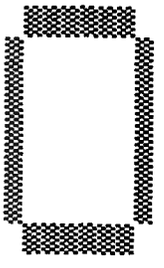
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

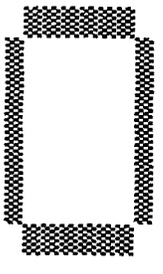
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



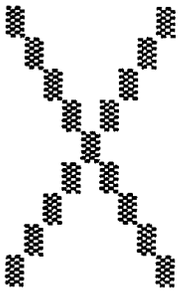
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/10 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

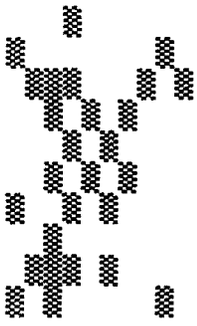
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

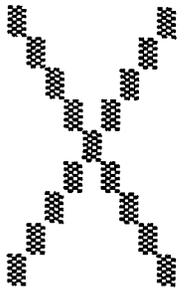
(1-b)
Harfin
(desenin)
Bozulmus Hali





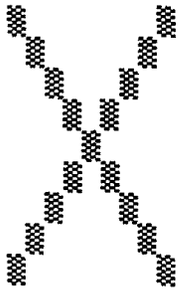
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/11 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

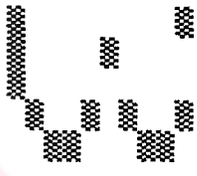


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

(a) (b)

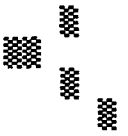
(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



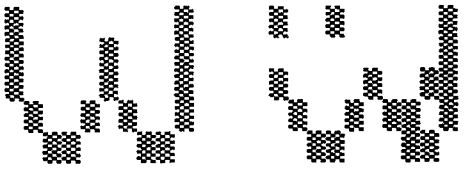
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/11 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

(1-a)
Seçilen harfin
düzgün hali.

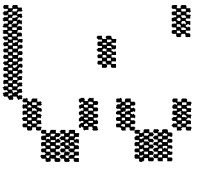


(1-b)
Harfin
(desenin)
Bozulmus Hali



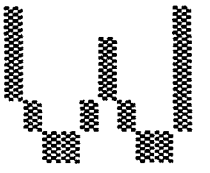
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/12 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

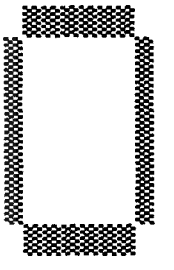


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

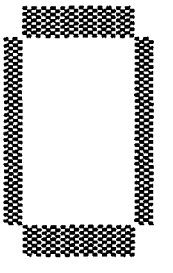
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



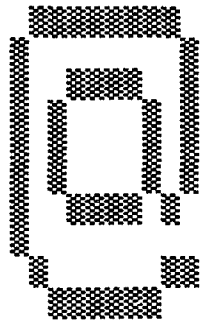
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/12 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

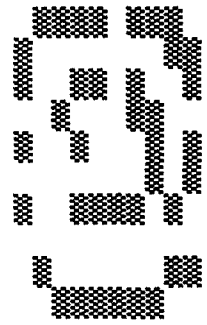


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

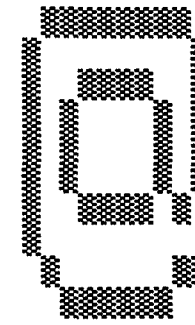


(a)



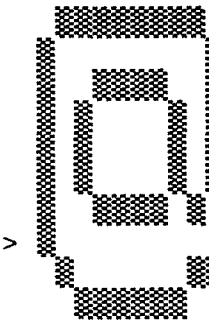
(b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



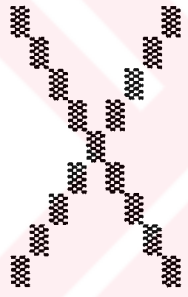
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/13 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

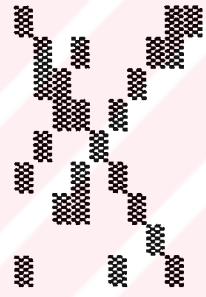


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

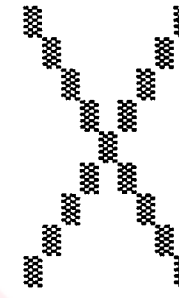


(a)



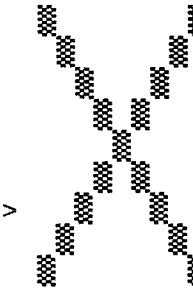
(b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



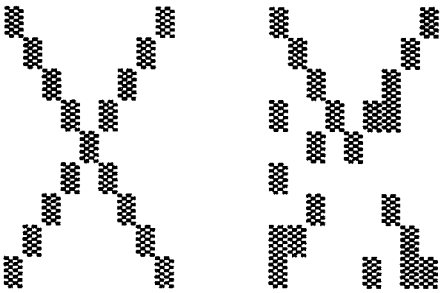
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/13 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



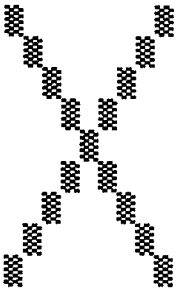
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



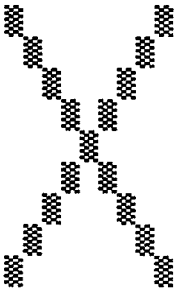
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/14 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

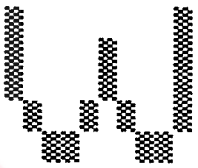


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

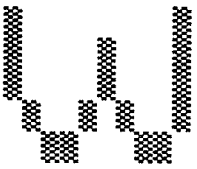
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



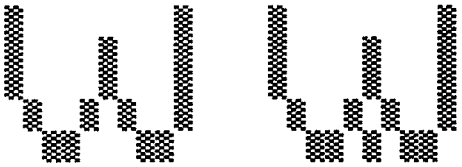
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/14 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



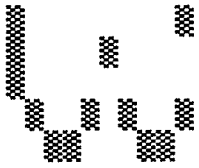
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)

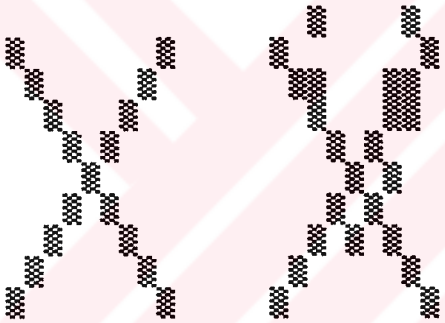


100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/15 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

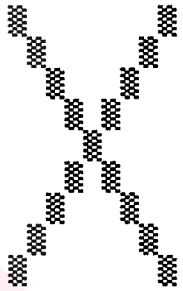
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



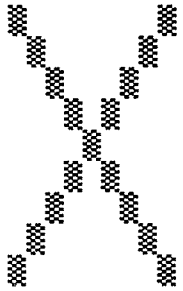
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/15 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

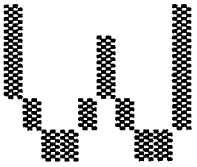


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

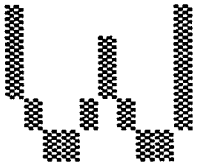
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



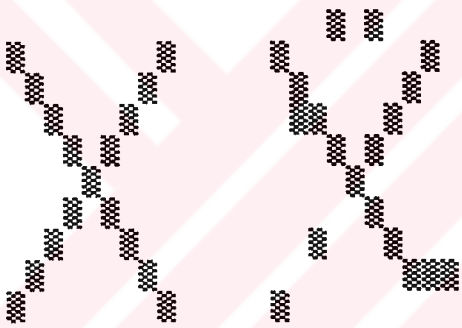
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/16 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



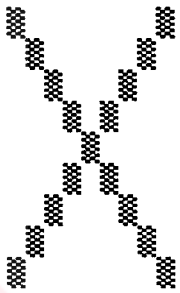
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



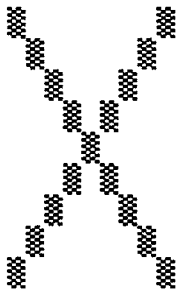
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



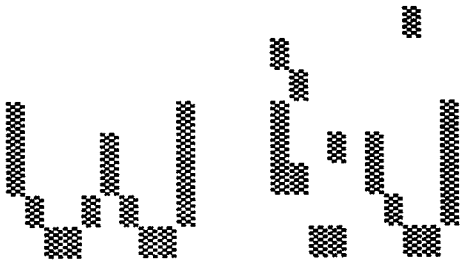
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/16 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



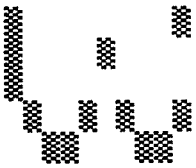
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



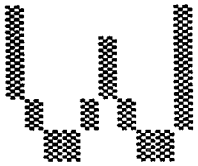
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/17 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

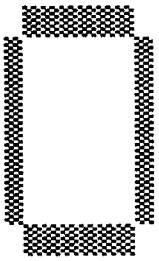


(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

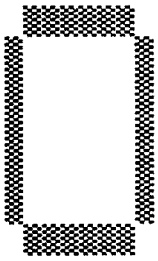
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



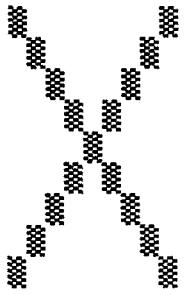
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/17 oranìnda bozulma.
Toplam 3 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

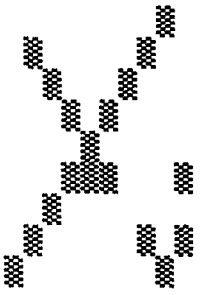
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

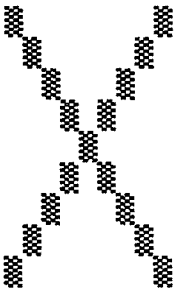
(1-b)
Harfin
(desenin)
Bozulmus Hali





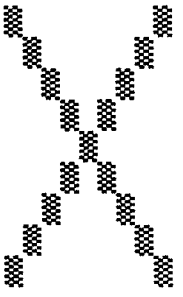
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/18 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

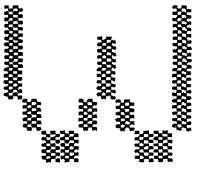
(1-b)
Harfin
(desenin)
Bozulmus Hali

(a) (b)

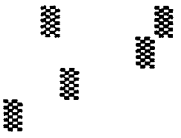
(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*ş* hali)

100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/18 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*ş*tirmek
için r tu*ş*una basìn.
Çìkmak için ESC tu*ş*una basìn.

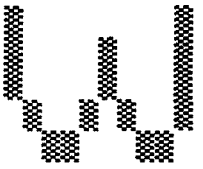
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->

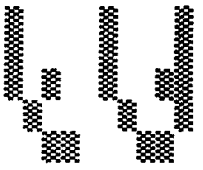


(1-a)
Seçilen harfin
düzgün hali.



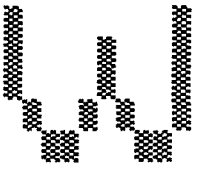
(1-b)
Harfin
(desenin)
Bozulmus Hali





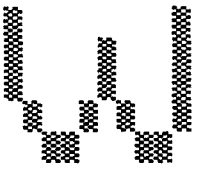
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)

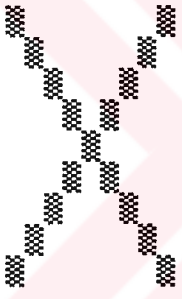


100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/19 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

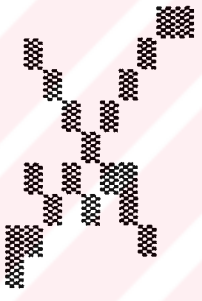
(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

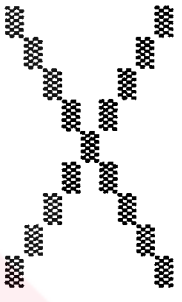


(1-b)
Harfin
(desenin)
Bozulmus Hali



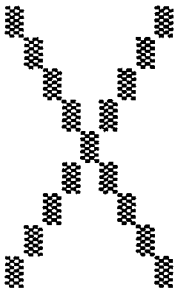
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmi*f* hali)



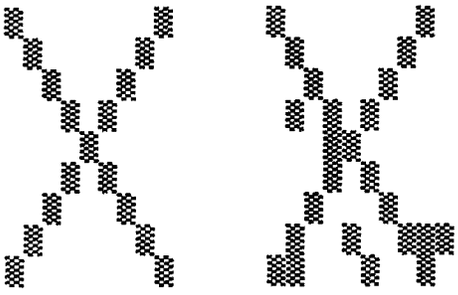
100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/19 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°i*f*tirmek
için r tu*f*una basìn.
Çìkmak için ESC tu*f*una basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



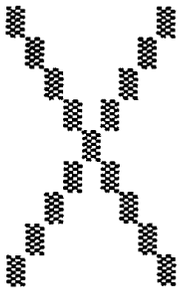
(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali



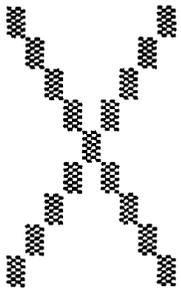
(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)



100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/20 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->



(1-a)
Seçilen harfin
düzgün hali.

(1-b)
Harfin
(desenin)
Bozulmus Hali

(a) (b)

(2)
Harfin (desenin)
Duzelmis Hali --->
(Hopfield'dan
geçmiƒ hali)

100 Neurons created.
10000 ways initialized.
Toplam 4 sìnìf var.
1/20 oranìnda bozulma.
Toplam 2 iterasyondan sonra.
Bozulma oranìnì de°iƒtirmek
için r tuƒuna basìn.
Çìkmak için ESC tuƒuna basìn.

(3)
GetMaxSimilar
procedure'unden
geçtikten sonra -->