

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Nöron Ağı, ve bir Uyg. Kro. Tan.

YÜKSEK LİSANS TEZİ

M. Elif Karşılıgil

1992

130-000
YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NÖRON AĞLARI VE BİR UYGULAMA:
KROMOZOM TANIMA

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜH. MİNE ELİF KARSLIGİL

TEZ YÖNETİCİSİ
Prof. Dr. MEHMET YAHYA KARSLIGİL

İSTANBUL 1991

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

R 368

61

Kot :

Alındığı Yer : ... Fen. Bil. Ens.

Tarih : ... 13.4.95

Fatura :

Fiyatı : ... 130.000.-

Ayniyat No : ... 1-4

Kayıt No : ... 51040

UDC :

Ek :

YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YILDIZ ÜNİVERSİTESİ
D.E. No 49899

ÖZET

ABSTRACT

1. TIBBİ GENETİK

1.1 Tıbbi Genetik Nedir ?

1.2 Sitogenetik Bilimi

1.2.1 Kromozom Morfolojileri

1.2.1.1 Kromozomlarda Sayısal Bozukluklar

1.2.2 Kromozomlar Nasıl Sınıflandırılır ?

1.3 Bilgisayarlı Genetik

KROMOZOM TANIMA

2. YAPAY NÖRON AĞLARI

2.1 Yapay Nöron Ağları Sistemlerinin Gelişimi

2.2 Bir Nöron Ağı Sistemi Nasıl Çalışır ?

2.2.1 Nöron Ağına Biyolojik Yapı

2.2.2 Yapay Nöron Ağına Yapı

2.3 Yapay Nöron Ağları Sistemlerinin Sınıflandırılması

2.3.1 Gözetim ile Çalışan Sistemler

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜH. MİNE ELİF KARSLIGİL



TEZ YÖNETİCİSİ

Prof. Dr. MEHMET YAHYA KARSLIGİL

İSTANBUL 1991

Genel İşlev İçindekiler	53
Genel İşlev İçindekiler	53
Genel İşlev İçindekiler	59
ÖZET	63
ABSTRACT	67
1. TIBBİ GENETİK	61
1.1 Tıbbi Genetik Nedir ?	61
1.2 Sitogenetik Bilimi	61
1.2.1 Kromozom Düzensizlikleri	63
1.2.1.1 Kromozomlarda Sayısal Düzensizlikler	63
1.2.1.2 Kromozomlarda Yapısal Düzensizlikler	63
1.2.2 Kromozomlar Nasıl Sınıflandırılır ?	76
1.3 Bilgisayar Kullanarak Kromozomların Sınıflandırılması	11
2. YAPAY NÖRON AĞLARI	12
2.1 Yapay Nöron Ağı Sistemlerine Giriş	12
2.2 Bir Nöron Ağı Sistemi Nasıl Çalışır	15
2.2.1 Nöron Ağının Biyolojik Yapısı	15
2.2.2 Yapay Nöron Ağının Yapısı	21
2.3 Yapay Nöron Ağı Sistemlerinin Sınıflandırılması	25
2.3.1 Öğretme İle Öğrenen Sistemler	25
2.3.1.1 Çoklu Seviyeli Ağlar	31
2.3.1.2 Back Propagation	32
2.3.1.3 Çok Seviyeli Ağ Uygulamaları	34
2.3.1.4 Hocalı Öğrenmenin Dezavantajları	38
2.3.2 Hocasız Öğrenme	40
2.3.2.1 Adaptif Resonans Teorisi	42
2.3.2.1.1 MAXNET	45
2.3.2.2 Hocasız Öğrenmenin Dezavantajları	47
3. GÖRÜNTÜ İŞLEME	
3.1 Giriş	48
3.2 Görüntü İşleme Algoritmaları	49
3.2.1 Point Proses	49
3.2.2 Sınıflandırma	51
3.2.3 Geometrik İşlemler	52

3.2.4	Frame İşlemleri	53
3.2.5	Alan İşlemleri	53
3.2.5.1	KÖŞE Belirleme	59
3.2.6	Kontur Belirleme	63

3.3	Sonuç	67
-----	-------	----

4.	ŞEKİL TANIMA	68
4.1	Giriş	68
4.1.1	Görüntü İşleme	68
4.1.1.1	Parazitlerin Temizlenmesi	68
4.1.1.2	Köşe Belirleme	68
4.1.1.3	Kontur Belirleme	69
4.1.1.4	Şekillerin Alanlarının Bulunması	69
4.2	Şekil Tanıma	71

KAYNAKLAR

ÖZGEÇMİŞ

İçerik

Özet

ÖZET

Sayısal hesapları insanların erişemeyeceği bir hızda ve doğrulukta yapabilen bilgisayaralar, insan zekasını ve düşüncesini gerektiren işlemleri yapmakta, şekli tanıma, algılamada ve algıladıklarını anlamlandırmakta ve en önemlisi, tecrübelerinden birşeyler öğrenmekte insanlara göre çok yetersiz kalırlar. Bu, insan beyni ile dijital bilgisayarların bilgi işleme yöntemi arasındaki büyük farklılıktan kaynaklanmaktadır. İnsan beyninin hesaplama ve değerlendirme hızı, birbirlerine karışık bir ağ yapısı ile bağlı çok sayıdaki nörondan gelmektedir. Bunu göz önüne alan araştırmacılar, insan beyninin temel bileşenleri olan nöronları inceleyerek, nöron ağlarını yapay olarak geliştirme çalışmalarına başlamışlardır.

İnsan beynine benzer sistemler geliştirmek için çalışmalar yapılmasının en önemli sebeplerinden biri beynin paralel çalışma özelliğidir. İnsan beynindeki nöronlar bilgisayarlardaki işlemcilerle karşılaştırılamayacak kadar basit yapıda fakat çok fazla sayıda ve paralel, yani aynı anda birden fazla işi yürüterek çalışan birimlerdir. Zeka, nöronların içinde ayrı ayrı değil, hücre ağının tümündedir. Bilinen pekçok yapay zeka uygulamasında en önemli sorun hızdır. Oysa bu uygulamaları paralel olarak gerçekleştirmek mümkün olduğunda bu çalışmalarda büyük gelişmeler elde edileceği gerçektir.

Bu projede sitogenetik yöntemlerle elde edilen 2 boyutlu kromozom resimleri bilgisayara aktarılarak, şekil tanıma yöntemleriyle kromozomları sınıflandıracak bir sistem geliştirilmiştir. Sınıflandırma işlemi, kendi kendine öğrenme yeteneğine sahip bir sistem olarak tasarlanmıştır.

BU sayede sistem her türlü kromozomu tanıyabilecek yetenekte bir Yapay Nöron ağı uygulaması olarak gerçekleştirilmiştir.

Bana araştırma zevkini aşılayan, beni bu çalışmaya yönlendiren ve projenin her aşamasında bilgi ve deneyimleriyle yol gösteren babam, hocam Prof. M. Yahya Karslıgil'e, tıbbi genetik konusunda bilgilerinden yararlandığım Prof. Asım Cenani ve asistanı Aslı Silahtaroglu'na, dökümantasyon sırasında yardımcı olan bütün arkadaşlarıma ve en önemlisi, beni yetiştiren çalışmalarım sırasında manen destekleyen aileme şükranlarımla..

One reason for studying brain-like models is their parallelism. In the brain, the individual processing elements, the nerve cells, are simple, but there are vast numbers of them. Intelligence, that much-prized property of brains, resides in the total network of cells, not the individual neuron. Many Artificial Intelligence programs run very slowly. If we can find ways to run AI programs in parallel, they will run in a reasonable amount of time.

Up to now neural networks have been studied mainly in the realm of science, subject for research, not development. There are some projects especially in Artificial Intelligence fields.

At this study, two dimensional chromosome pictures obtained by cytogenetic methods were transferred to the computer as digital data. A special system designed by the help of the ISTANBUL, 1991 strategies to analyze and classify the chromosomes. The system is specially designed to learn if

ABSTRACT

Traditional computing has always excelled at performing numerical calculations at speeds far exceeding human abilities. Computers have proved to be less capable in other areas where people excel, areas such as perception and learning from experience. Noting that the brain is an example of a machine that performs these tasks, researchs have been studying the behavior of networks of the brain's basic component, the neuron. Now researchers are finding that networks of artificial neuron like elements, called Neural Networks, may be capable of performing tasks such as learning and perception that until now were beyond the ability of computers.

One reason for studying brain-like models is their parallelism. In the brain, the individual processing elements, the nerve cell, are simple, but there are vast numbers of them. Intelligence, that much-prized property of brains, resides in the total network of cells, not the individual neuron. Many Artificial Intelligence programs run very slowly. If we can find ways to run AI programs in parallel, they will run in a reasonable amount of time.

Up the now neural networks have been studied mainly in the realm of science, subject for research, not development. There are some projects aspecially in Artificial Intelligence fields.

At this study, two dimentional cromosom pictures obtained by sytogenetic methods were transfered to the computer as digital data. A special system designed by the help of the image recognition strategies to analyse and classify the cromozoms. The system is specially designed to learn it

self and has the ability to recognize all kinds of
chromozoms using a Neural Network.

1.1.1.TIBBİ GENETİK NEDİR ?

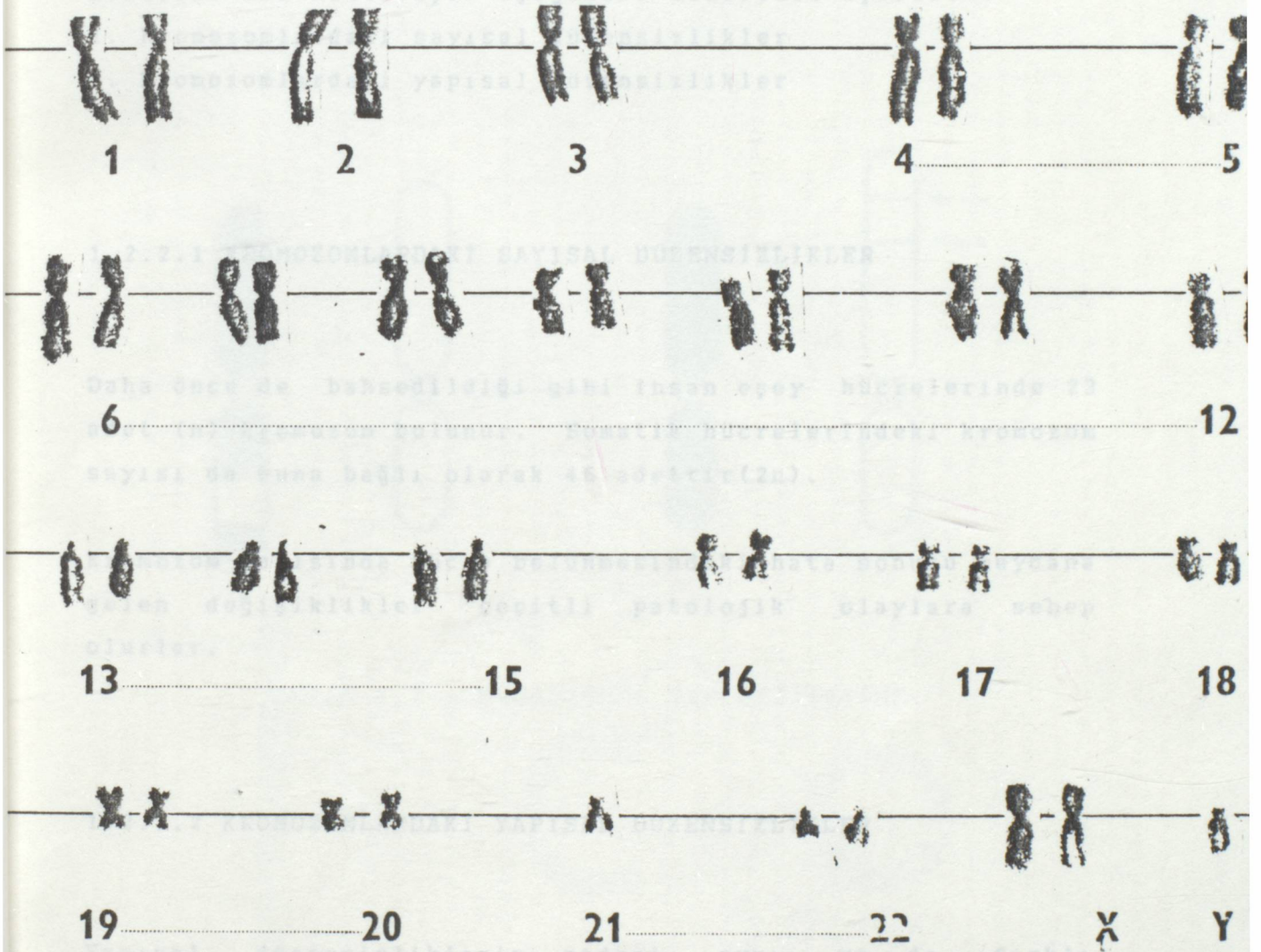
Basit bir tanımla, kalıtım ve değişimle ilgilenen bilim dalına genetik diyebiliriz. Kalıtsal madde, biyolojik sınırlılığı ve gücü ile organizmayı ortaya koyar. Fakat çevreyi oluşturan genlerle birbirine etki ederek organizmanın anatomik, fizyolojik ve diğer özelliklerini belirler. Bundan dolayı genetik yalnızca kalıtımı değil, değişimi de kapsar; bir önceki kuşaktan bir sonraki kuşağa neyin nasıl geçtiğini araştırır. Tıbbi Genetik çalışmalarının en önemli amaçlarından biri, genlerle kuşaktan kuşağa aktarılan hastalıkların tanısı ve kromozomal hastalıkların sonraki kuşaklara daha az zarar vermesini sağlamaktır.

1.2. SİTOGENETİK BİLİMİ

Kromozom kelimesi, Yunanca'da chromos(renkli) soma(cisim) kelimelerinin birleşmesinden oluşmuş, çeşitli boyaları tutma özelliklerinden dolayı renkli cisim anlamına gelen bir kelimedir. Kromozomlar canlıların tüm nükleuslu hücrelerinde bulunur ve organizmanın kalıtım bilgisini taşıyan DNA'yı içerir. Hücredeki kromozomlarla ilgilenen bilim dalına sitogenetik denir.

Her türün kendine özgü ve tek bir kromozom yapısı ve sayısı vardır Buna o türün karyotipi denir. İnsan kromozomları, cinsiyet belirleyen X ve Y kromozomları dahil 46 tanedir(Şekil 1.1). Bunlardan vücut hücrelerinin özelliklerini belirleyen 22 çifti(1-22) otozom olarak adlandırılırken 1 çifti de gonozom adını alır ve cinsiyeti

belirler. Normal bir diğide cinsiyeti belirleyen iki X kromozomu, erkekte bir X ve bir de Y kromozomu bulunur. İnsanda bulunan 23 çift kromozomda her çiftin biri anneden biri babadan geçer, yalnız erkekliği belirleyen Y kromozomu daima babadan geçer.



ŞEKİL 1.1 : 23 ÇİFT İNSAN KROMOZOMU

1.2.2 KROMOZOM DÜZENSİZLİKLERİ :

Yer değiştirilme bir kromozomdan kopan parça diğer bir kromozoma yerleşir. Bu hatalar büyük bir parça kaybına

Normalde 46 olan ($2n=46$) insan kromozomları kimi zaman hem sayı, hem şekil hem de yapı bakımından değişiklik gösterebilir ki bu durum kromozomal hastalıkların ortaya çıkmasına neden olur. Toplumdaki her 200 kişiden birisinde kendini gösterebilen kromozomal düzensizlikleri oluşturan kusurlar ana hatlarıyla aşağıdaki sınıflara ayrılır :

1. Kromozomlardaki sayısal düzensizlikler
2. Kromozomlardaki yapısal düzensizlikler

1.2.2.1 KROMOZOMLARDAKİ SAYISAL DÜZENSİZLİKLER

Daha önce de bahsedildiği gibi insan eşey hücrelerinde 23 adet (n) kromozom bulunur. Somatik hücrelerindeki kromozom sayısı da buna bağlı olarak 46 adettir ($2n$).

Kromozom sayısında hücre bölünmesindeki hata sonucu meydana gelen değişiklikler çeşitli patolojik olaylara sebep olurlar.

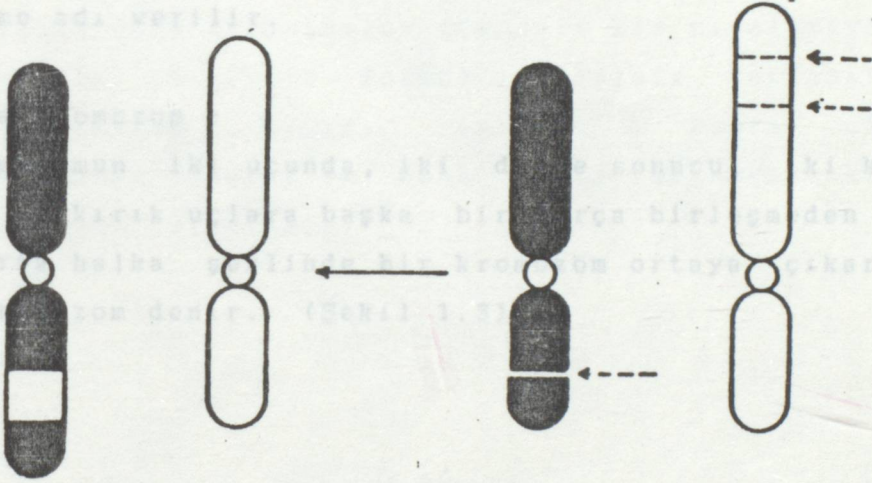
SERİ 1.2 : KROMOZOMDA YER DEĞİŞTİRME

1.2.2.2 KROMOZOMLARDAKİ YAPISAL DÜZENSİZLİKLER

Yapısal düzensizliklerin nedeni aynı ya da farklı kromozomlardaki kırılma ve yeniden düzenlenmelerdir. Bunların belli başlılarını inceleyelim:

a) Yer Değiştirme :

Yer değiştirmede bir kromozomdan kopan parça diğer bir kromozoma yerleşir. Bu kırılmalar büyük bir parça kaybına neden olmadığı için kişide fazla bir gen kaybı olmayacak ve böyle bir kromozom taşıyan kişi, genetik yapısında ve fenotipinde bir değişiklik göstermeyecektir. Önemli olan nokta, bu taşıyıcı kişinin çocuklarında kromozom dengesizliklerinin ortaya çıkacak olmasıdır. (Şekil 1.2)



ŞEKİL 1.2 : KROMOZOMDA YER DEĞİŞTİRME

ŞEKİL 1.3 : HALEA KROMOZOMUN OLUŞUMU

b) Eksilme:

Eksilme, bir kırılma sonucu, kromozomun küçük bir parçasının kopması demektir.

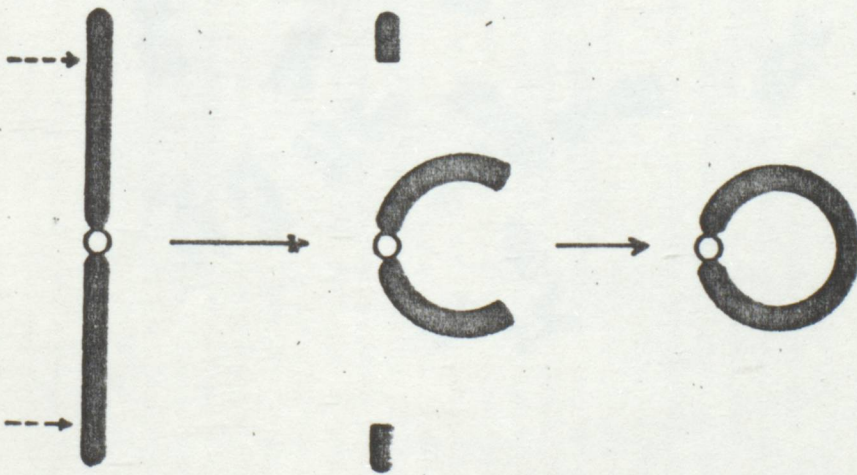
c) Artma : İki kromozomdan birinden çift darbe sonucu kopan parçanın diğerinden tek darbe sonucu kopan aralığa girerek kaynaşmasına denir.

d) Ters dönme :

Bir kromozoma iki darbenin gelmesi ve bunun sonucunda kopan parçanın, kaybolmadan yani eksilmeye uğramadan kendi eksenini çevresinde 180 derece dönerek yine eski yerine yapışmasına tersdönme adı verilir.

e) Halka kromozom :

Bir kromozomun iki ucunda, iki darbe sonucu, iki kırılma olur ve bu kırık uçlara başka bir parça birleşmeden iki uç kaynaşırsa halka şeklinde bir kromozom ortaya çıkar, buna halka kromozom denir. (Şekil 1.3).



ŞEKİL 1.3 : HALKA KROMOZOMUN OLUŞUMU

1.2.2 KROMOZOMLAR NASIL SINIFLANDIRILIRLAR ?

Günümüzde, çeşitli sitogenetik yöntemlerle elde edilen kromozomlar, geliştirilen yöntemlerle tek tek sınıflandırılabilen, hangi kromozomda ne tür bir yapısal ve sayısal değişiklik olduğu kesin olarak belirlenebilmektedir.

İnsan kromozomlarının sayı ve morfolojilerini, 1 damla insan kanından elde etmek mümkündür. İnsan kromozomları en kolay ve sık olarak, periferik kandaki lenfositlerden elde edilir. Elde edilen bu kromozom fotoğrafı (Şekil 1.4) kullanılarak önce kromozomlar standart bir sıralamaya tabii tutularak tüm bilgi bir formül içerisinde toplanır; önce toplam kromozom sayısı yazılır. Sonra cinsiyet kromozomlarının yapısı ve daha sonra da varsa kromozom anormallikleri belirlenir.

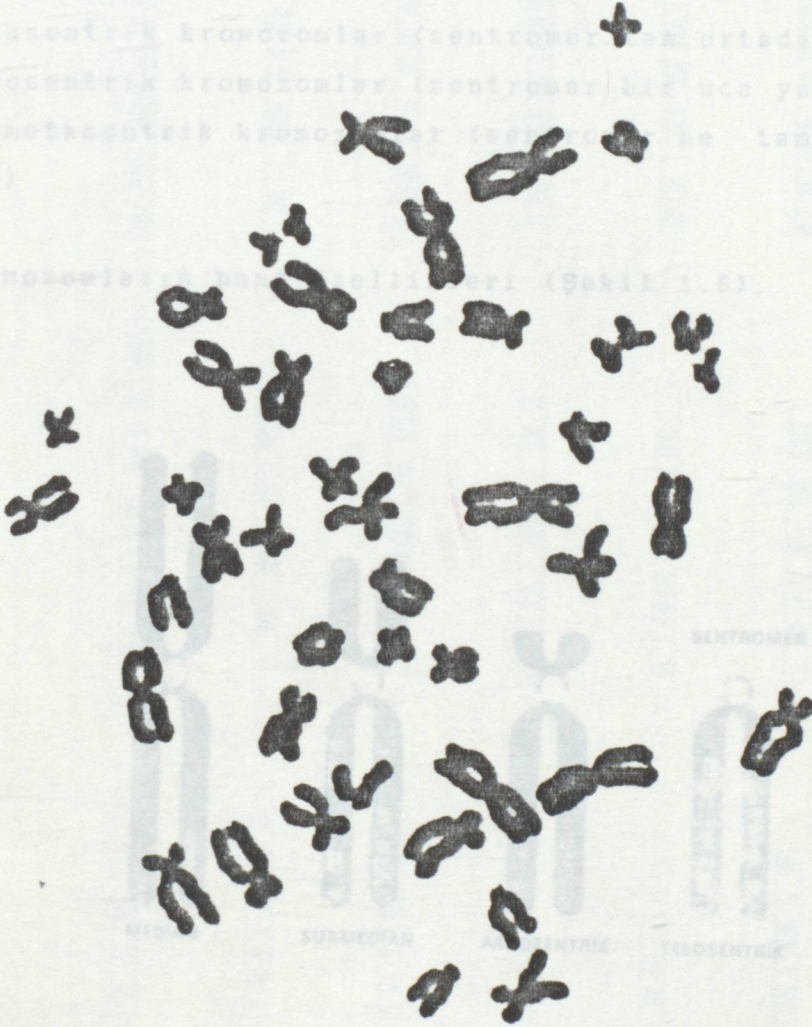
ŞEKİL 1.4 : KROMOZOM FOTOĞRAFI

1. Uzunluk; toplan ve ayrı ayrı kolların uzunlukları. Her kromozomun bir uzun, bir de kısa kolu vardır.

2. Sentromer pozisyonu. Kromozomun ortasındaki sentromer olarak işlevlendirilen dar kısımın yeri, her kromozom için sabittir. Sentromerin yerleşim bölgesine göre kromozomlar üç kısma ayrılır:

- a. Metacentrik kromozomlar (sentromer ortada)
- b. Akrocentrik kromozomlar (sentromer bir uca yakın)
- c. Submetacentrik kromozomlar (sentromer ne tam uca ne de ortada)

3. Kromozomların boyutları (Şekil 1.6)



ŞEKİL 1.5 : SENTROMER POZİSYONLARI

Bu yöntemler kullanılarak elde edilen bir kromozom yapısı Şekil 1.7'deki gibidir.

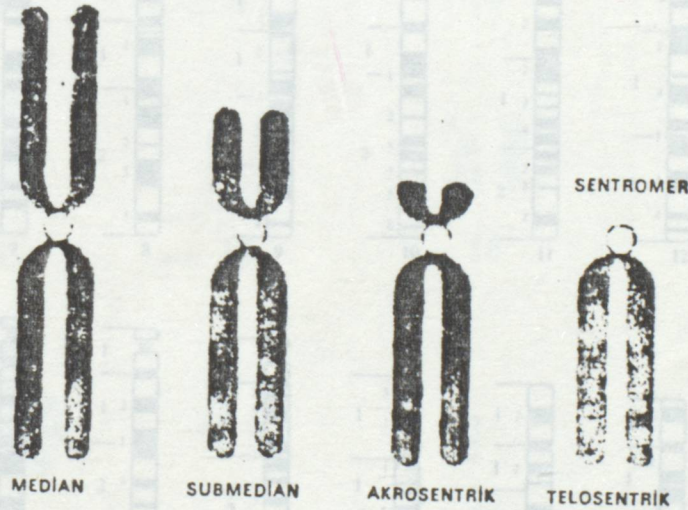
Temelde kromozomlar sınıflandırılırken şu kriterler dikkate alınır:

1. Uzunluk; toplam ve ayrı ayrı kolların uzunlukları. Her kromozomun bir uzun, bir de kısa kolu vardır.

2. Sentromer pozisyonu. Kromozomun ortasındaki sentromer olark isimlendirilen dar kısmın yeri, her kromozom için sabittir. Sentromerin yerleşim bölgesine göre kromozomlar üç kısma ayrılırlar :

- a. Metasentrik kromozomlar (sentromer tam ortada)
- b. Akrosentrik kromozomlar (sentromer bir uca yakın)
- c. Submetasentrik kromozomlar (sentromer ne tam uçta ne de ortada)

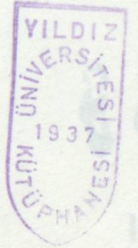
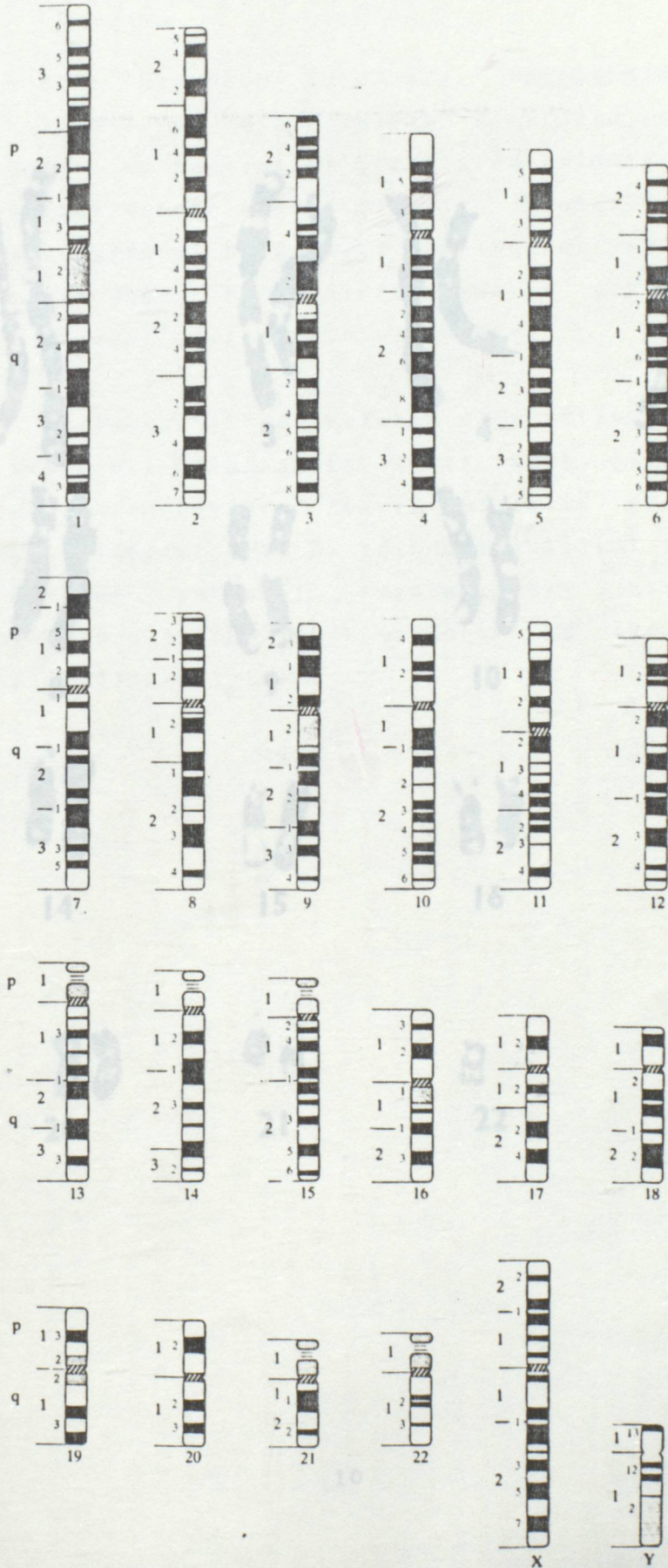
3. Kromozomların bant özellikleri (Şekil 1.6)



ŞEKİL 1.5 : SENTROMER POZİSYONLARI

Bu yöntemler kullanılılmasıyla sınıflandırılmış bir kromozom yapısı Şekil 1.7'deki gibidir.

ŞEKİL 1.6 : KROMOZOMLARIN BANT ÖZELLİKLERİ



ŞEKİL 1.7 : SINIFLARINA AYRILMIŞ KROMOZOMLAR



JMA C. 6191

1.3 BİLGİSAYAR KULLANARAK KROMOZOMLARI SINIFLANDIRMA :

Şekil 4 'deki kromozom fotoğrafı incelendiğinde, bu fotoğraf kullanılarak kromozomları sınıflandırmanın çok hassas ölçümler ve kontroller gerektiren oldukça zahmetli bir iş olacağını tahmin etmek kolaydır. Kromozom tanıma ve sınıflandırma işlemini bilgisayar kullanarak yapılabilirse daha hassas ve doğru bir sınıflandırmanın çok daha kısa sürede elde edilebilmesi mümkün olacaktır.

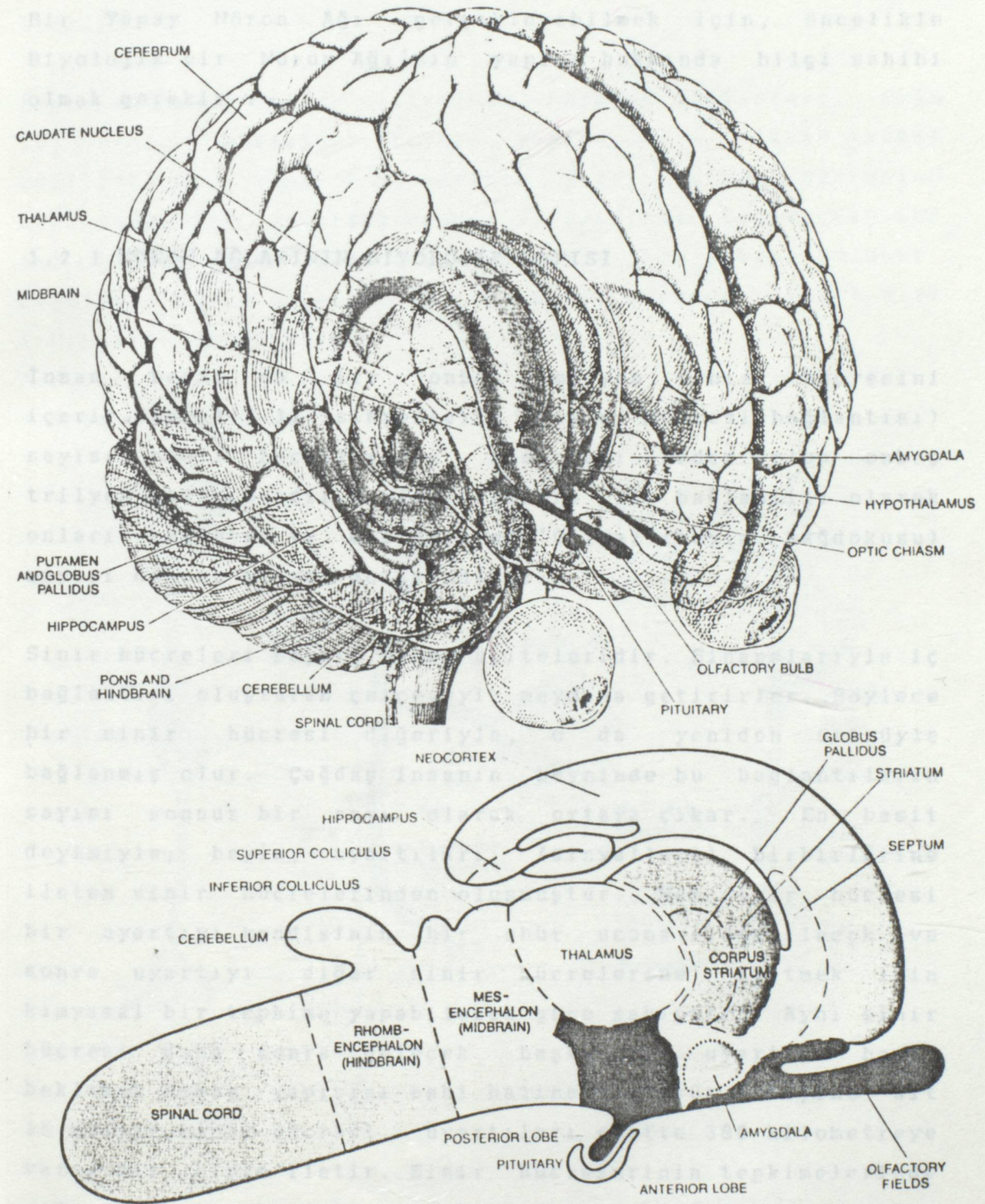
Bu projede, sitogenetik yöntemlerle elde edilen 2 boyutlu kromozom resimleri bilgisayara aktarılarak, şekil tanıma yöntemleriyle kromozomları sınıflandıracak bir sistem geliştirmeye çalışılmıştır. Sınıflandırma işlemi için kendi kendine öğrenme yeteneği kazandırılmış bir sistem kurulmasının gerekliliği düşünülmüş ve bu amaçla YAPAY NÖRON AĞLARI kullanılmıştır.

1.1 YAPAY NÖRON AĞI SİSTEMLERİNE GİRİŞ

Sayısal hesapları insanların erişemeyeceği hızda ve doğrulukta yapabilen bilgisayarlar, insan zekasını ve düşüncesini gerektiren işlemleri yapmakta, daha önceki deneyimlerden birşeyler öğrenme, konuşulan bir dili anlama, şekil tanıma, algılama ve algıladıklarını değerlendirmekte insanlara oranla çok daha yavaş ve yetersiz kalırlar. Bu, insan beyni ile dijital bilgisayarların bilgi işleme kapasitesi arasındaki büyük farktan kaynaklanmaktadır. Dijital bilgisayarlar akıllı işler yapmaları için de programlanabilirler. Fakat bu, fazla ve farklı sayıda bilgi işleme için çok karmaşık programlar gerektirir. İnsan zekasını ve düşüncesini gerektiren işlemleri yapabilen, algılama, yorum, karar verme gibi yeteneklere sahip bilgisayar sistemleri geliştirebilme üzerine olan çalışmalara Yapay Zeka çalışmaları denir. Bu çalışmalarda amaç insan beyninin bir simülasyonudur. Fakat bugüne kadar geliştirilen sistemler hız ve çalışma şekli bakımından henüz çok yetersiz bir seviyededir. İnsan beyninin hesaplama ve değerlendirme hızı birbirlerine karışık bir ağ ile bağlı olan çok sayıdaki nöronlardan gelmektedir. Bunu göz önüne alan araştırmacılar, beynin temel bileşenleri olan nöronları inceleyerek, Yapay Zeka çalışmalarını kolaylaştırabilmek amacıyla, Yapay Nöron Ağları geliştirme çalışmalarına başlamışlardır.

Bilgisayarların temel yapısı insan beyninden tamamıyla farklıdır. Bunun tek sebebi bilgisayarların yaşayan hücreler yerine, elektronik elemanlardan bir araya gelmesi değildir. Bilgisayarların bilgi değerlendirme ve işleme yöntemi ve kapasitesinden doğan temel farklılıklar da söz konusudur. İnsan beynindeki nöronlar, bilgisayardaki işlemcilerle karşılaştırılamayacak kadar basit yapıda fakat

ŞEKİL 2.1 : İNSAN BEYNI



1.2 BİR NÖRON AĞI SİSTEMİ NASIL ÇALIŞIR ?

Bir Yapay Nöron Ağı gerçekleyebilmek için, öncelikle Biyolojik bir Nöron Ağı'nın yapısı hakkında bilgi sahibi olmak gerekir :

1.2.1 NÖRON AĞLARININ BİYOLOJİK YAPISI

İnsan beyni on ile onbeş milyon sinir hücrelerini içerir. (Şekil 2.1) sinapsların (beyin hücresi bağlantısı) sayısı sinir hücrelerinin bin katı kadar olup onbeş trilyon dolayındadır. Sinir hücreleri ile bağlantılı olarak onları besleyen ve destekleyen 'Gliyal' (beyin bağdokusu) sayısı doksan milyar dolayındadır.

Sinir hücreleri beynin temel üniteleridir. Sinapslarıyla iç bağlantıyı oluşturan çerçeveyi meydana getirirler. Böylece bir sinir hücresi diğeriyle, o da yeniden öbürüyle bağlanmış olur. Çağdaş insanın beyinde bu bağlantıların sayısı sonsuz bir sayı olarak ortaya çıkar. En basit deyişle; beyin, uyartıları (sinyalleri) birbirlerine ileten sinir hücrelerinden oluşmuştur. Her sinir hücresi bir uyartıyı kendisinin bir öbür ucuna iletebilecek ve sonra uyartıyı diğer sinir hücrelerine iletmek için kimyasal bir tepkime yapabilecek güce sahiptir. Aynı sinir hücresi daha sonra gelecek başka bir uyartıya hazır beklemek üzere yapısını eski haline döndürür. Beyine ait 15 milyar sinir hücresi, uyartıları saatte 380 kilometreye varan bir hızla iletir. Sinir hücrelerinin tepkimelerinde uçlarıyla çıkardıkları kimyasal maddeler oldukça çeşitlidir. Her hücrenin bin dolayında bazı durumlarda

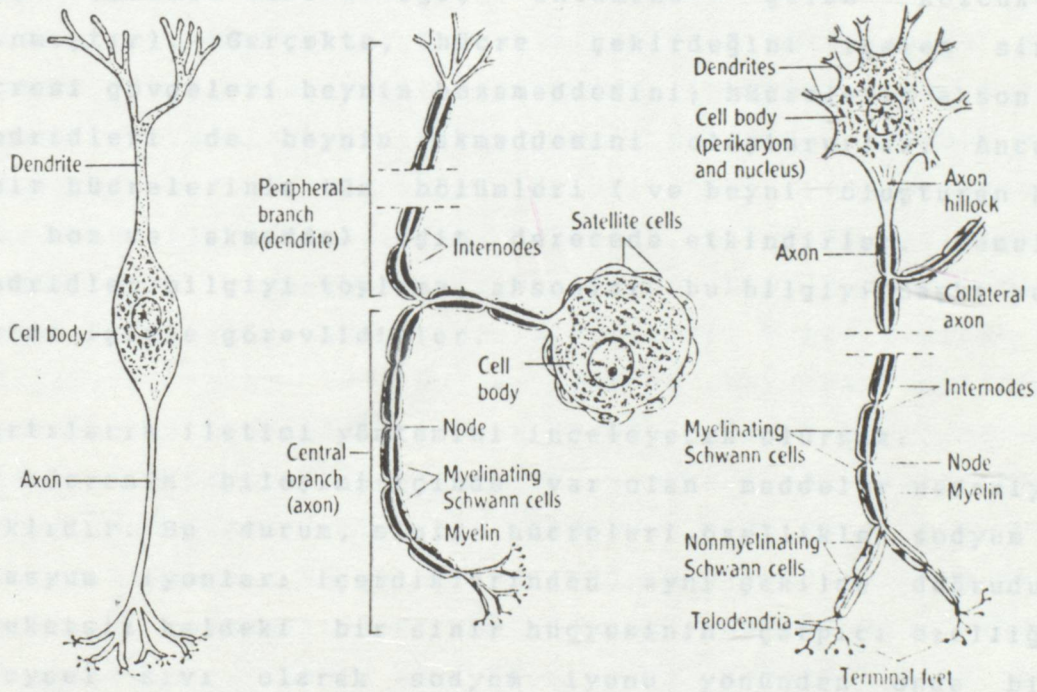
ombine yakın sinir ucu bulunmaktadır. Bu sistemi çalışır durumda tutabilmek dakikada elli milimetre oksijen gerektirir. Bu da tüm bedenin kullandığı oksijenin dörtte biridir.

Sinir hücresi, sinir sisteminin başlama noktasıdır. Çoğu kez sinir hücreleri beyin yapıtaşları olarak kabul edilirler. Her sinir hücresinin diğer sinir hücresine benzemeyen bir yuvarlağımsıdan piramidimsi biçime değişen ama hiçbir zaman tam tanımlanamayan bir şekli bulunur. Biyolojik nöron üç kısımdan oluşan bir hücredir: (Şekil 2.2)

* Dendrite :Kısa Uzantı

* Hücre

* Axon :Uzun Uzantı



ŞEKİL 2.2 NÖRON

Dendrite'ler hücreye gelen girişleri toplarlar. Dendrite tarafından alınan sinyaller hücrede birleştirilerek bir çıkış darbesi üretilip üretilmeyeceğine karar verilir. Eğer bir iş yapılacaksa üretilen çıkış darbesi axon'lar tarafından taşınarak diğer nöronlarla olan bağlantılara veya terminal organlara iletilir. Beyindeki korteksde her nöronun bir karşılığı vardır. Bir nöronun çıkışı, ona bağlı olan bütün nöronlara iletilir. Fakat korteks, işin yapılabilmesi için hangi nöronu harekete geçirecekse yalnızca ona komut gönderir.

iyon değiş tokuşu olması ve bunun da oluşturduğu korkunç

Sinir hücrelerinin hemen her zaman pek uzun ve pek ince uzantıları olan aksonları ile dallara ayrılan dendridleri bulunur (Dendrid sözcüğü benzerliği yönünden doğru olarak Eski Yunanca'daki ağaç anlamına gelen sözcükten alınmıştır). Gerçekte, hücre çekirdeğini içeren sinir hücresi gövdeleri beynin bozamaddesini; hücrelerin akson ve dendridleri de beynin akamaddesini oluştururlar. Ancak, sinir hücrelerinin tüm bölümleri (ve beyni oluşturan her iki boz ve akmadde) eşit derecede etkindirler. Temelde dendridler bilgiyi toplama, aksonlar bu bilgiyi başka yere iletme işiyle görevlidirler.

değişiklik olur ve bu da akson boyunca uyarılma

Uyarıların iletimi yöntemini inceleyecek olursak:

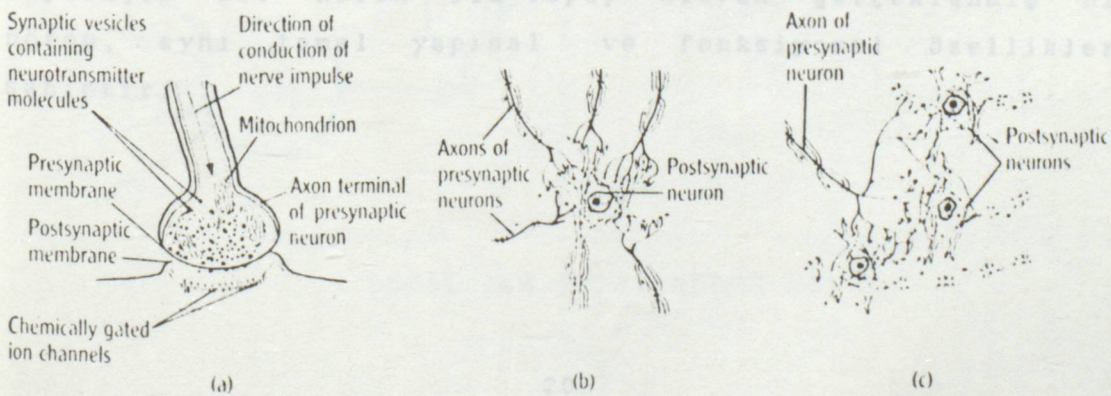
Her hücrenin bileşimi içinde var olan maddeler nedeniyle farklıdır. Bu durum, sinir hücreleri özellikle sodyum ve potasyum iyonları içerdiklerinden aynı şekilde doğrudur. Hareketsiz haldeki bir sinir hücresinin çarpıcı özelliği, yüzeysel sıvı olarak sodyum iyonu yönünden onda biri oranında zengin oluşu; aynı zamanda, hücrenin iç kesiminde potasyum iyonu olarak da on katı zengin oluşudur.

Hücre zarında kanallar bulunduğundan hücrede bir sızıntı olması kaçınılmazdır. Sodyum iyonları hücrenin içine doğru sızarken potasyum iyonları da dışa doğru sızacaklardır. Bu yüzden hücre zarında bu akışları dengelemek için sodyum -

potasyum adenosine trifosfat adıyla adlandırılan pompacıklar bulunmaktadır. Bu pompalar çok hünerli ve işlevleri çok önemli olmak üzere, Sürekli üç sodyum iyonunu iki potasyum iyonu ile değiş tokuş ederler. Her sinir hücresinin zarında hafif çıkıntılar şeklinde bir milyon dolayında pompacık vardır. Her pompacık saniyede 200 sodyum iyonunu 130 potasyum iyonu ile değiştirir (Olayı, insan beyni de bulunan milyarlarca sinir hücresi yönünden düşünmek oldukça güçtür. Çünkü, bu hücrelerin herbirinde var olan birer milyon dolayındaki pompa cıkta saniyede 330 iyon değiş tokuşu olması ve bunun da oluşturacağı korkunç sayının düşünülmesi gerekmektedir).

Bu olay bu şekilde durmadan tekrarlanır. Pompacıklar sodyum/potasyum dengesizliğini korumayı başarır ve bu yüzden her bir sinir hücresinin iç kısmını dışa göre negatif yetmiş milivolt kapasitede tutarlar. Bu durumu sürekli koruma; kişi etkin olsun olmasın, uyasun yada uyanık olsun, düş görsün ya da bilincini yitirmiş olsun, hemen sürekli denebilecek oksijen istemini açıklamaya yardım eder. Sonra sinir hücresine bir uyartı gelir. Başlangıçta, sinir hücresinin elektriksel kapasitesinde bir değişiklik olur ve bu da akson boyunca uyartının ilerlemesini hızlandırır. Buna paralel olarak hücre zarının geçirgenliğinde bir değişme ve sodyum iyonlarının hücre içine dökülüşüne olasılık sağlayan durumda bir bir farklılık oluşur. Hücre zarında sodyum akışına olanak veren kanallar bulunmaktadır. Bu durum da kendi kendini geliştiricidir; şöyle ki, iyon akımı çoğaldıkça bu seli arttırabilmek için yeni kanallar açılır. Bu durumda birdenbire, hücre zarı elektriksel yükünde negatiften pozitif doğru değişmeyle sodyum kanalları kapanır. Bu kez de diğer bir gurup kanal açılır ve potasyum iyonlarının dışarı akması sağlanır. Bu da başlangıçtaki negatif yetmiş milivoltluk elektriksel kapasitenin yeniden elde edilmesi

demektir. Bu uyarının iletimi sırasında, sinire başka bir uyarı etki yapamaz. Bu durumda sinir hücresi için, kesin olarak başka bir uyarı etki edemez açıklaması yapılabilir. Tüm bu fizyolojik, kimyasal ve elektriksel değişimler olup bitince ve sinir hücresi hareketsiz durumundaki kapasitesine dönüşünce, bir başka benzeri uyarı hücre boyunca yol alabilir. Bununla birlikte, en küçük sinir lifleri bile saniyenin yüzde biri kadar zamanda eski durumlarına dönerlerken, daha büyük lifler bu konuda on katı fazla hıza erişebilirler. Sodyum iyonlarının kanallarda hızla akması sonra pompacıklarla geriye çıkarılması ve yeniden hücrenin içine doğru hızlanmaları saniyede yüz defalık yinelenmelerle olur. Ancak işlem, tek bir hücrede çok fazla iyonun işe karışması ve tek bir beyinde de milyara varan hücrenin varoluş nedeni ile insanı pek fazla ürkütmemelidir. Hızlı ya da yavaş olsun, bir uyarının iletimi, sinir hücresinin aksonu boyunca gerçekleşir. Sinir hücresinin gövdesinde ayrıca dendridler bulunmaktadır. Gerçekte dendridlerin sayısı pek çoktur. Sinir sistemimizdeki bölümlerin sayıları, binlerle, milyonlarla ve bazan milyarlarla ifade olunur. Sinapslar, bir sinir hücresi ile diğeri arasındaki temas (birleşme) noktalarıdır. (Şekil 2.3) Bu noktalar bir akson ile bir dendrid ya da bir dendrid ile diğeri bir dendrid arasında olabilir. Yapılan araştırmalarla, bilgiyi bir sinir hücresinden diğeri iletenin, sinirsel iletken denilen kimyasal maddeler olduğu anlaşılmıştır.



1.2.3 YAPAY NÖRON ŞEKİL 2.3: NÖRON YAPISI

Sinaps köprüsünden geçiş zamanı elektriksel iletim ile ilgili değildir. Ve burada olaya karışan kimyasal maddeler yavaş, yavaş tanımlanmıştır. Başlangıçta bunların yalnızca iki madde olduğu; bilgileri güçlendirip artırmak ve sonra zayıflatıp bitirme için bunlardan birinin uyarma diğerinin de yasaklama işlevi yaptığı sanılmıştır. Oysa şu ana kadar otuz sinirsel iletken tanımlanmıştır.

Özetleyecek olursak; hücreler birer ateşleyici değildir. İçerideki sinirsel uyarıtları ileten kimyasal maddeler aracılığıyla evet hayırın ötesinde daha duyarlı bilgileri iletebilecek yeteneğe sahiptirler. Sinirsel uyarının iletimi ile bu uyarının alınması farklı şeylerdir. Tanımlamaya çalıştığımız gibi, Sinaps bir sinir hücresi ile diğeri arasındaki aralıktır. Bu sebepten ötürü sinapslar birinci hücrenin çıkardığı kimyasal maddeyi ikinci hücrenin bir haber, bir uyarı veya iletilecek bir bilgi olarak tanımlarına tanık olur. Her sinirsel uyarı yol boyunca biçim değiştirir. Olayın gerçekleştiği yer de sinapstir. bu nedenlerle sinir hücresinde olanlar; nasıl uyardığı ya da uyarının bittiği, şu veya bu kimyasal maddeyi çıkardığı, yol boyunca şu yada bunu kullandığı, hızlı ya da yavaş yol aldığı gibi bütün konular beyin çalışmasıyla temelden ilgilidir.

Biyolojik bir nöron ile Yapay olarak gerçekleştirilmiş bir nöron, aynı temel yapısal ve fonksiyonel özelliklere sahiptir.

ŞEKİL 2.4: YAPAY NÖRON AĞI

1.2.2 YAPAY NÖRON AĞININ YAPISI :

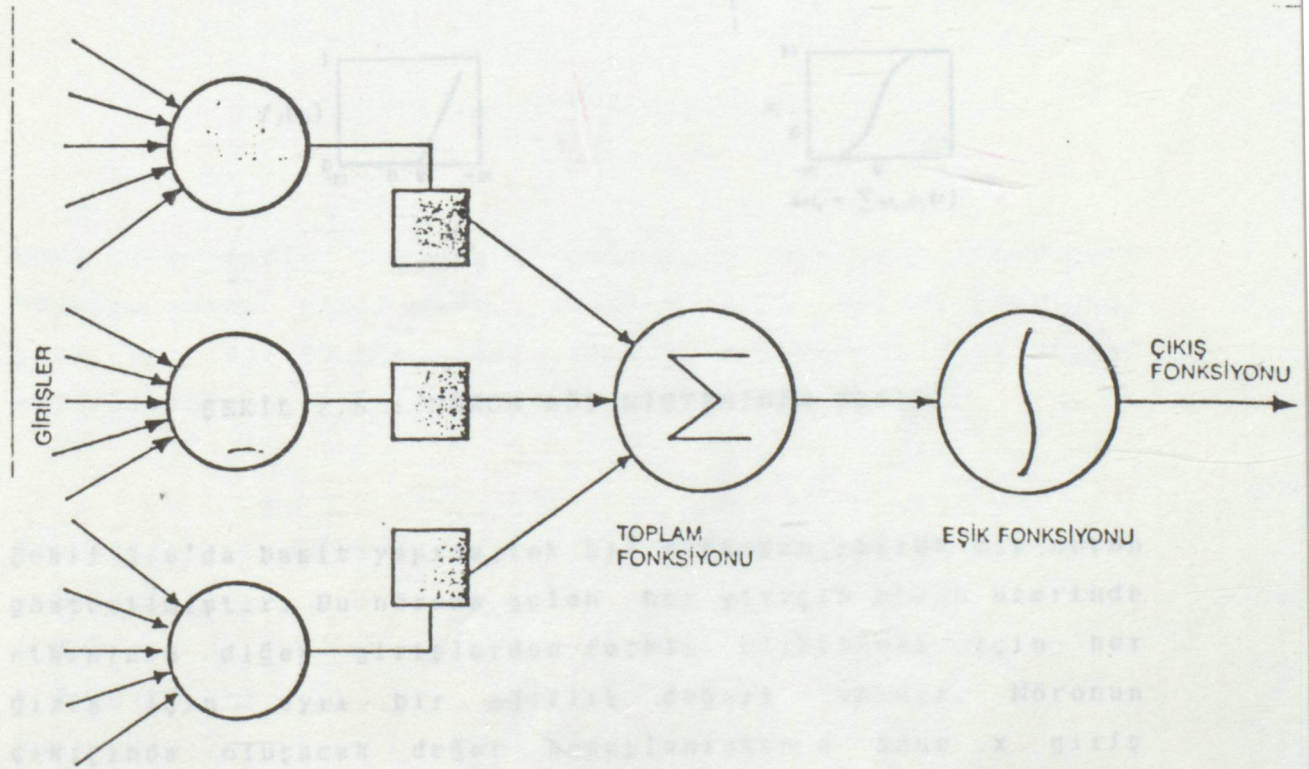
Bir Nöron Ağı Sistemindeki bütün üniteler dört önemli bileşene sahiptir: (Şekil 2.4)

* Giriş Bağlantıları: Üniteye diğer ünitelerden gelen etkilenimler

* Toplam Fonksiyonu: Farklı giriş aktivitelerini tek bir değer haline getiren fonksiyon.

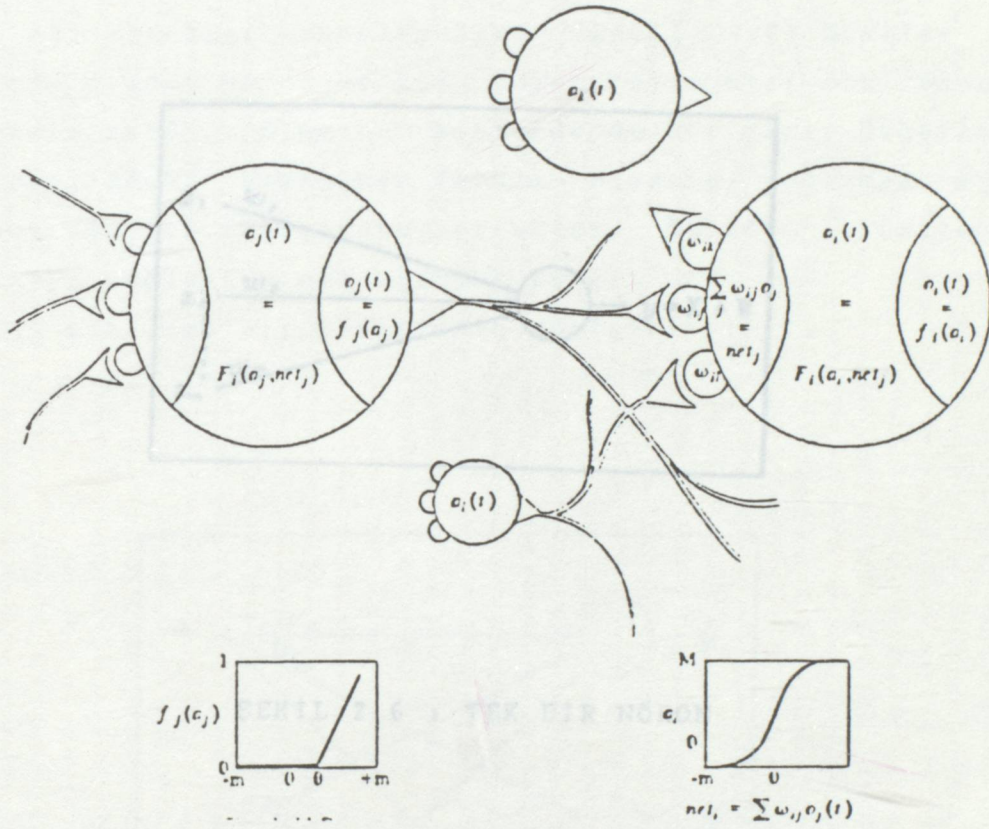
* Eşik Fonksiyonu: Giriş etkilenimlerinin toplam değerini çıkış etkinliğine çeviren fonksiyon. (Eğer toplam fonksiyonu ile elde edilen değer eşik değerinin altına düşerse çıkış değeri 0 olur.)

Çıkış Bağlantıları:



ŞEKİL 2.4 YAPAY NÖRON AĞI

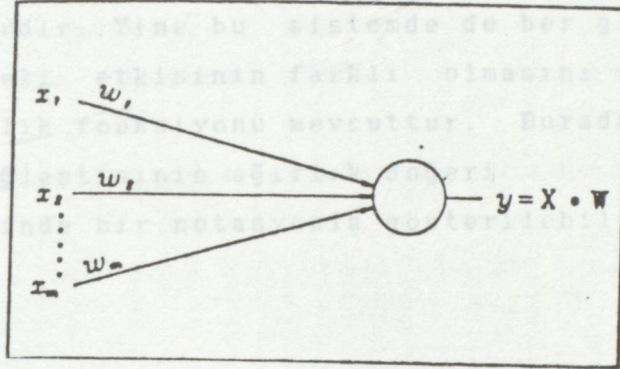
Birimin hesaplanan sonuç değerleridir. Bağlı olduğu diğer birimler için giriş değeridir. (Şekil 2.5)



ŞEKİL 2.5 : NÖRON AĞI SİSTEMİNİN YAPISI

Şekil 2.6'da basit yapıda, tek bir düğümden ibaret bir nöron gösterilmiştir. Bu nörona gelen her girişin nöron üzerinde etkisinin diğer girişlerden farklı olabilmesi için her giriş için ayrı bir ağırlık değeri vardır. Nöronun çıkışında oluşacak değer hesaplanırken n tane x giriş değeri ve her giriş için n tane w ağırlık değerinin çarpımlarının toplamı alınır :

$$y(i) = \sum_{j=1}^n w(j) * x(i)$$



ŞEKİL 2.6 : TEK BİR NÖRON

Eğer elde edilen toplam, nöronun eşik seviyesi değerinden büyükse nöron aktif olur, değilse pasif kalır. Görüldüğü gibi bu çalışma sistemi insan beynindeki biyolojik nöronların çalışma sisteminin bir simülasyonudur.

Biyolojik nöronlar arasında bağlantı değeri temelde iki şekildedir:

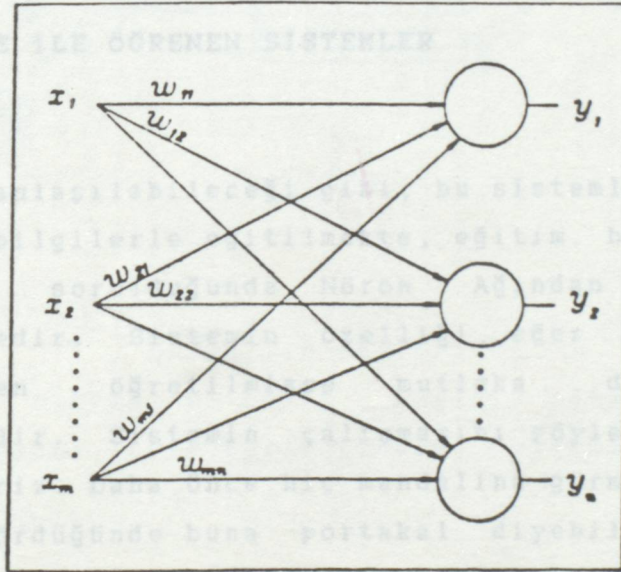
- * Uyarıcı
- * Baskılayıcı

Buna göre,

örneğin bir kas üzerindeki değer uyarıcı konumda ise kasılmaya, baskılayıcı konumda ise gevşemeye sebep olur.

Yapay Nöron Ağında bu modlar, ağırlıklara pozitif veya negatif değerler verilerek belirlenmiştir. Pozitif ağırlıklar uyarıcı, negatif ağırlıklar ise baskılayıcıdır.

Şimdi birden fazla nöronun birleşmesinden oluşan basit bir nöron ağı modelini inceleyelim. (Şekil 2.7) Şekilde ağ yapısında n adet nöron ve her nörona ayrı ayrı etki eden m adet giriş vardır. Yine bu sistemde de her giriş değerinin nöron üzerindeki etkisinin farklı olmasını sağlamak için $m \times n$ adet ağırlık fonksiyonu mevcuttur. Burada i . ünitenin j . üniteye bağlantısının ağırlık değeri w_{ij} şeklinde bir notasyonla gösterilebilir.



ŞEKİL 2.7

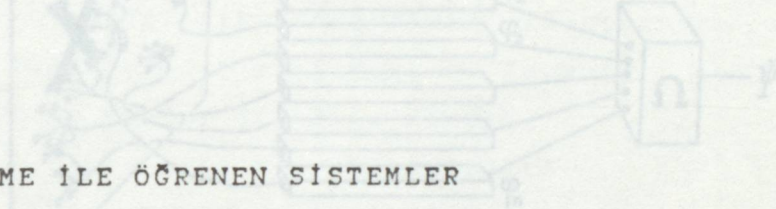
1.3 YAPAY NÖRON AĞI SİSTEMLERİNİN SINIFLANDIRILMASI

Yapay Nöron Ağı sistemlerinin en önemli özelliği sistemlerin öğrenme yeteneğine sahip olmasıdır.

Günümüze kadar yapılan Nöron Ağı Çalışmaları incelendiğinde, Yapay Nöron Ağı Sistemlerinin temelde iki sınıfa ayrıldığı görülür. Bunlar :

1. Öğretme ile öğrenen sistemler (Supervised Neural Networks)
2. Kendi kendine öğrenen sistemler (Unsupervised Neural Networks)

Şimdi bu sistemleri daha ayrıntılı olarak inceleyelim :



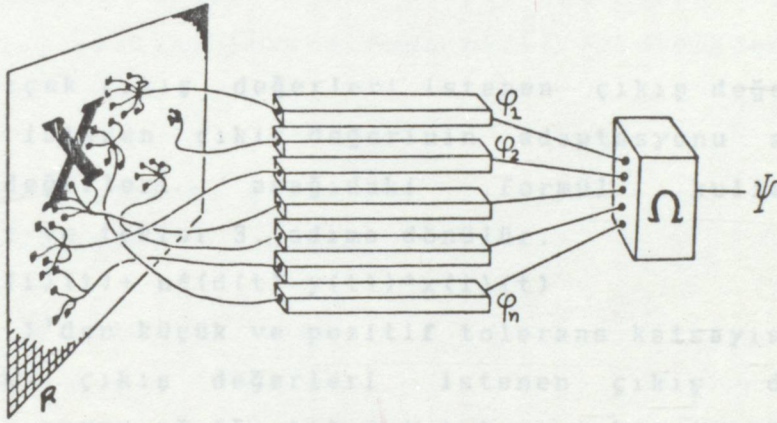
1.3.1 ÖĞRETME İLE ÖĞRENEN SİSTEMLER

İsminden de anlaşılabilir olduğu gibi, bu sistemlerde nöron ağı önce örnek bilgilerle eğitilmekte, eğitim bittikten sonra bu bilgiler sorulduğunda Nöron Ağından doğru cevap alınabilmektedir. Sistemin özelliği, eğer sorulan örnek daha önceden öğretilmişse mutlaka doğru cevabın alınabilmesidir. Sistemin çalışmasını şöyle bir örnekle açıklayabiliriz. Daha önce hiç mandalina görmemiş bir çocuk mandalina gördüğünde buna portakal diyebilir. Çocuğa bu meyvenin mandalina değil portakal olduğu söylendikten sonra belki çocuk bir daha mandalina gördüğünde tekrar portakal diyecektir ama meyvenin gerçekte mandalina olduğu çocuk öğrenene kadar tekrarlandığında artık çocuk mandalinayı gördüğünde tanıyabilecektir.

Şimdi öğretme ile öğrenen Nöron Ağı sistemlerinden en çok bilinen birkaçını inceleyelim :

PERCEPTRON

Bu alanda geliştirilen ilk model olarak kabul edilen perceptron, verilen giriş bilgisinin mevcut iki sınıftan hangisine eşit olabileceğini bulacak şekilde eğitilen en basit Nöron Ağı yapısıdır (Şekil 2.8).



ŞEKİL 2.8 PERCEPTRON

Matematiksel olarak y çıkış değeri ağıdaki girişlerin ağırlıklı toplamı eşik seviyesinden büyükse 1, değilse 0 olacaktır.

Açıklaarsak, ağda bulunan tek düğüm, giriş bilgilerinin ağırlıklı toplamını hesaplayarak bu toplamı eşik seviyesi değeri ile karşılaştırır. Eğer toplam eşik seviyesini aşıyorsa örnek bir sınıfa, değilse diğer sınıfa aittir.

Perceptron'un çalışma algoritmasını şöyle açıklayabiliriz :

1. Önce ağırlık değerleri ($w(i)(0)$, $0 \leq i \leq N-1$) ve eşik

seviyesi (E) random değerlerle belirlenir.

2. Nöron ağına öğretilmesi gereken bilgi yani, giriş değerleri ve bu giriş değerleri için istenen çıkış değerleri verilir. (Girişler:- $x(0), x(1), \dots, x(n)$ ve çıkış $d(t)$)

3. İlk adımda random olarak belirlenen ağırlık değerleri, eşik seviyesi ve mevcut giriş değerleri kullanılarak gerçek çıkış değeri belirlenir:
$$y(t) = f\left(\sum (w(i)(t) * x(i)(t)) - E\right)$$

4. Eğer gerçek çıkış değerleri istenen çıkış değerlerine uymuyorsa, istenen çıkış değerinin adaptasyonu amacıyla ağırlık değerleri aşağıdaki formül kullanılarak değiştirilir ve tekrar 3. adıma dönülür.

$$w(i)(t+1) = w(i)(t) + n * (d(t) - y(t)) * x(i)(t)$$

(Burada n , 1'den küçük ve pozitif tolerans katsayısıdır.)

Eğer gerçek çıkış değerleri istenen çıkış değerine uyuyorsa, o zaman ağ öğretilmesi istenen bilgiyi öğrenmiş demektir.

Önce OR probleminin çözümü için sağlanması gereken şartları belirleyelim :

Bilindiği gibi OR'un çalışma şekli aşağıdaki gibidir :

Girişler	Çıkış
0 0	0
0 1	1
1 0	1
1 1	1

Buna göre E eşik seviyesi için, istenen çıkış değerlerinin eldesi için şu şartlar sağlanmalıdır:

çözümlenir: $0 * w(1) + 0 * w(2) < E$ gibi, perceptron
 sadece $0 * w(1) + 1 * w(2) > E$ ile ilgili birbirinden
 ayrılabilecek $1 * w(1) + 0 * w(2) > E$ için problemleri
 çözümlenir. $1 * w(1) + 1 * w(2) > E$ ya daha fazla sınıfa
 ihtiyacı bulunan problemleri çözmek için yapılması gereken,
 Yani OR devresini öğrenen bir nöron ağına öğrenme işleminin
 tamamlanabilmesi için ağırlık değerlerinin her ikisinin
 pozitif olması yeterlidir.

Toplama, çarpma gibi basit problemleri çözmekte
 kullanılabilecek olan perceptron, daha karmaşık problemleri
 çözmekte yetersiz kalır. Örnek olarak bu nöron ağına XOR
 devresinin çalışmasını perceptrona nasıl öğretebileceğimizi
 inceleyelim :

XOR'un yapısı aşağıdaki gibidir :

Girişler Çıkış

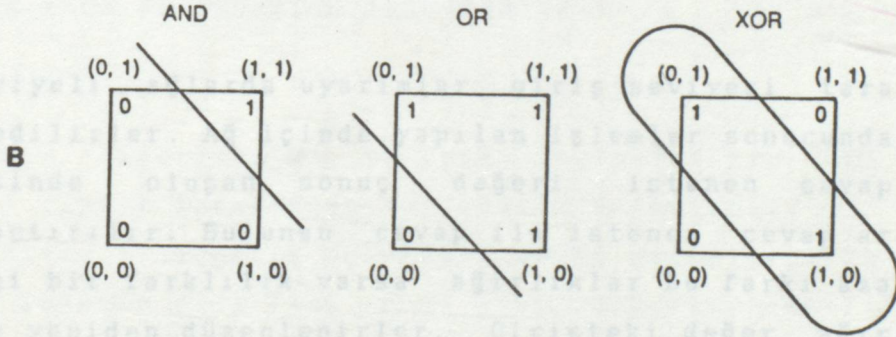
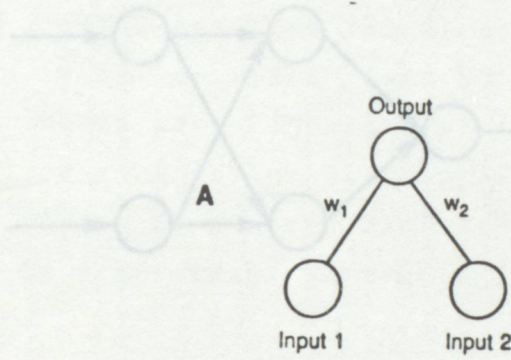
0	0	0
0	1	1
1	0	1
1	1	0

Bu durumda istenen çıkış değerlerinin eldesi için aşağıdaki
 şartlar sağlanmalıdır :

$0 * w(1) + 0 * w(2) < E$	-->	$0 < E$
$0 * w(1) + 1 * w(2) > E$	-->	$w(2) > E$
$1 * w(1) + 0 * w(2) > E$	-->	$w(1) > E$
$1 * w(1) + 1 * w(2) < E$	-->	$w(1) + w(2) < E$

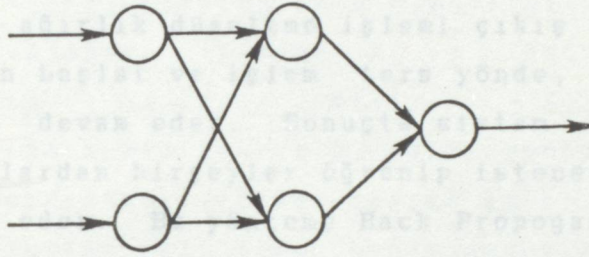
Rahatça görülebileceği gibi $w(1)$ ve $w(2)$ değerleri ayrı
 ayrı E 'den büyükken aynı değerlerin toplamının $(w(1) + w(2))$
 E 'den küçük olması şartını sağlamak imkansızdır.
 Şekil'2.9'da iki girişli bir çıkışlı bir nöron için AND ,
 OR ve XOR problemlerinin koordinat sisteminde durumu

gösterilmiştir. Burada da görülebileceği gibi, perceptron sadece iki sınıfa ayrılabilen (tek bir çizgiyle birbirinden ayrılabilen sınıflar) değerler için problemleri çözebilmekte, fakat XOR gibi 3 veya daha fazla sınıfa ihtiyaç duyulan problemleri çözmek için yapılması gereken, Nöron Ağına yeni seviyeler eklemektir.



1.3.2.1 ÇOKLU SEVİYELİ AĞLAR

Çoklu seviyeli ağlar, giriş ve çıkış seviyeleri arasında birden fazla seviyenin kullanıldığı Nöron Ağı Sistemleridir. Gizli seviye (hidden layer) olarak isimlendirilen bu seviyelerde, düğümleri aracısız giriş alamayan ve aracısız çıkış veremeyen üniteler vardır. (Şekil 2.10)



ŞEKİL 2.10

1.3.2.2 BACK PROPAGATION ALGORİTHMASI

İki seviyeli ağlarda uyarımlar giriş seviyesi tarafından kabul edilirler. Ağ içinde yapılan işlemler sonucunda çıkış seviyesinde oluşan sonuç değeri istenen cevap ile karşılaştırılır. Bulunan cevap ile istenen cevap arasında herhangi bir farklılık varsa ağırlıklar bu farkı azaltacak şekilde yeniden düzenlenirler. Girişteki değer, ağırlıklar uygun noktaya ulaşana kadar değişmez. Üç seviyeli ağlarda ise düzen çok daha karmaşıktır. Uyarımlar giriş seviyesinden alınırlar. Ağ içinde seviyeler arasında filitrelenirler. Hesaplanan çıkışlar istenilen cevaplarla karşılaştırılarak sonuçta gerekirse hata sinyali verilir. Hata sinyali, gizli birimlerden çıkış birime olan ağırlıkları değiştirmekte kullanılırlar. Fakat bunu

yaparken giriş seviyesinden gizli seviyeye gelen ağırlıkların değiştirilip değiştirilmeyeceğini düşünmek gerekir. Gizli birimlerden ne tür bir çıkış istendiği bilinemeyeceği için gizli birimlerin çıkışında hata sinyali verilmesi kolay bir iş değildir. Bunun yerine, her gizli birimin çıkış biriminin hatalarına etkisine değer biçilmelidir. Bu her hatalı birim için, gizli birime bağlı olan çıkış birimlerinin hata sinyallerinin ağırlıkları toplamı alınarak yapılır. Multigizli seviyeli sistemlerde her seviyenin hata sinyalleri bir önceki seviyenin düzeltilmiş sinyallerinden 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 yaparak, ama bu hatalardan birşeyler öğrenip isteneni bulana kadar işleme devam eder. Bu yöntemle Back Propagation Algoritması denir. Şimdi bu algoritmayı adım adım inceleyelim :

Şekil 7.11'de back propagation algoritmasıyla XOR için çalışacak şekilde eğitilmiş bir Nöron Ağı'nın eğitim sonrası

1.3.1.2.1 BACK PROPOGATION ALGORİTMASI

1. Ağırlık değerleri(w) ve eşik seviyelerinin (E) ilk değerleri random fonksiyonu kullanarak belirlenir.
2. $x(i)$ ($i=0..n-1$) giriş değerleri ve bu girişler için elde edilmesi istenen $d(j)$ ($j=0..m-1$) çıkış değerleri verilir.
3. Sigmoid*linner olmayan fonksiyon :

$$-(a-E)$$

$$f(a)=1/(1+e)$$

kullanılarak gerçek çıkış değerleri :

$$y(j)= f((w(i)(t)*x(i)(t) - E)$$

hesaplanır.

4. Eğer gerçek çıkış değerleri istenen çıkış değerlerine uymuyorsa rekürsif olarak çıkış düğümlerinden geriye doğru

ilk seviyeye ulaşılan kadar gidilerek ağırlıklar aşağıdaki şekilde düzenlenir :

$$w(i)(j)(t+1) = w(i)(j)(t) + n * o(j) * x(i)'$$

Burada $w(i)(j)$, t anında i. düğümden j. düğüme olan ağırlık değeridir. n tolerans katsayısı, o ise j. düğümün hata katsayısıdır. Aşağıdaki şekilde hesaplanır :

Eğer düğüm çıkış düğümü ise :

$$o = y(j) * (1 - y(j)) * (d(j) * y(j))$$

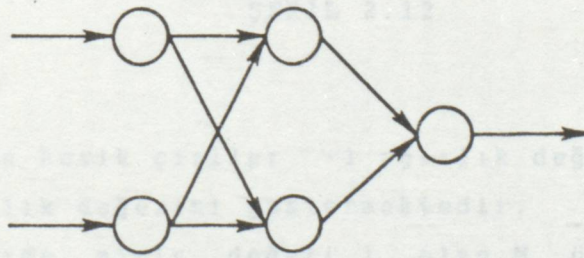
Burada $d(j)$ j. düğüm için istenen çıkış değeri, $y(j)$ ise hesaplamalar sonucu elde edilen gerçek çıkış değeridir.

Eğer düğüm bir gizli seviye düğümü ise :

$$o(j) = x(j) * (1 - x(j)) * (\text{toplam}(o(k)w(j)(k)))$$

olarak hesaplanır. Burada k, j. düğüme gelen bütün bağlantıların sayısını verir. Ağırlık değerleri hesaplandıktan sonra tekrar 2. adıma dönülerek gerçek çıkış değerleri hesaplanır. İşleme gerçek çıkış değerleri istenen çıkış değerlerine eşit olana kadar devam edilir.

Şekil 2.11'de back propogation algoritmasıyla XOR için çalışacak şekilde eğitilmiş bir Nöron Ağının eğitim sonrası ağırlık değerleri görülmektedir.

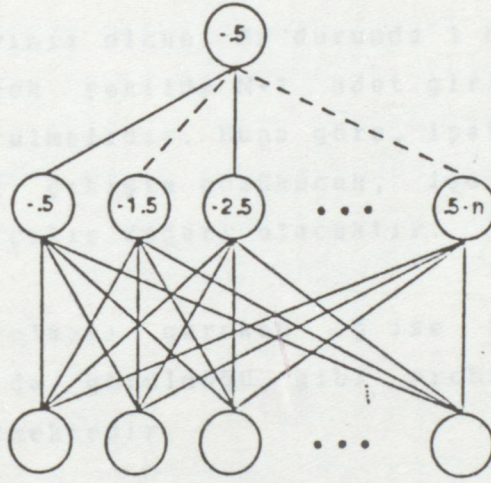


ŞEKİL 2.11

1.3.1.3 ÇOK SEVİYELİ AĞ ÖRNEKLERİ

PARITY Problemi :

Parity probleminde girişteki 1'lerin sayısı tek ise çıkışta 1, değilse 0 üretecek bir Nöron Ağı oluşturulmalıdır. Aslında dikkat edilirse XOR problemi için oluşturulan ağıın da iki girişli bir parity ağı olduğu görülür. Bu ağı oluşturmak için, N giriş değeri için en az N adet gizli ünite kullanılmalıdır. (Şekil 2.12)



ŞEKİL 2.12

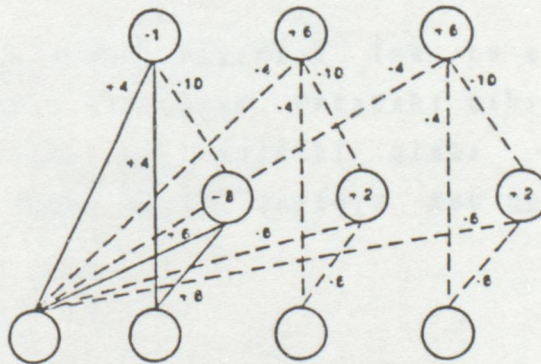
Şekil 2.12'de kesik çiziler -1 ağırlık değerini, diğerleri ise +1 ağırlık değerini göstermektedir. Ağıın çalışmasını incelediğimizde, giriş değeri 1 olan M ünite için M adet gizli ünitenin aktif hale geçtiğini görebiliriz. Gizli ünitelerden çıkış ünitesine gelen toplam sonuç değeri, girişteki 1'lerin sayısı tek iser 1, çift ise 0 olacaktır. 4 giriş değeri için bu yapının tablosunu şu şekilde oluşturabiliriz.

Aktif Giriş Ünitesi Sayısı	Gizli Ünite de Değerler	Çıkış Değeri
0	1111	0
1	1011	1
2	1010	0
3	0010	1
4	0000	0

NEGATİFLİK Problemi

N dijitlik bir sayımız olsun. Bu durumda 1 dijit de sayının işaretini gösterecek şekilde N+1 adet giriş ünitesi olan bir ağ yapısı kurulmalıdır. Buna göre, işaret biti 0 iken giriş olduğu gibi çıkışta gözükcek, işaret biti 1 ise girişin tümleyeni çıkış değeri olacaktır.

Bu yapı için kurulması gereken ağ ise Şekil 2.13'deki gibidir. Burada da görüldüğü gibi, problem 3 XOR ağı kullanılarak çözülmektedir.



ŞEKİL 2.13

TOPLAMA Problemi:

İki bitlik iki binary sayı arasında toplama işlemi yapabilen bir ağ modeli kurmayı deneyelim. Sayılar için 4 giriş ünitesi, çıkış değerini göstermek üzere de 5 çıkış ünitesi gerekecektir. Şekil 2.14'de de görüldüğü gibi, ağın iki gizli ünitesi vardır. Bu üniteler, toplama sırasında oluşacak eldeler için kullanılır. Şekildeki giriş ünitelerinden 1. ve 3. ünite sayıların yüksek anlamlı bitleri için, 2. ve 4. bitler ise alçak anlamlı bitleri için kullanılmaktadır.

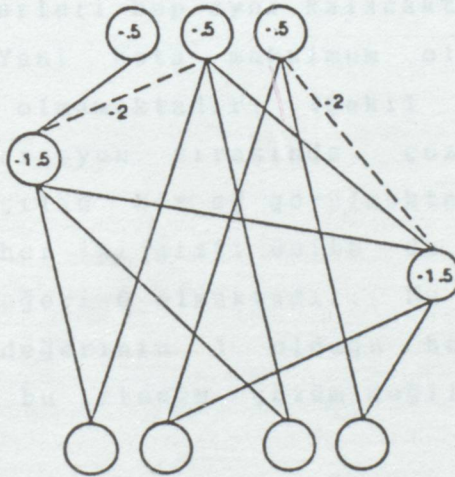
Buna göre, iki alçak anlamlı bit aktif ise (değerleri 1 ise), sağdaki gizli ünite aktif hale geçecektir. Her iki yüksek anlamlı bit aktifken veya yüksek anlamlı bitlerden birisi ve diğer gizli ünite aktifken de sağdaki gizli ünite aktif hale gelecektir. Burada baskılayıcı ağırlık değerleri -2, diğer bağlantıların ağırlık değerleri ise +1'dir.

Eğer her iki alçak anlamlı bit aktif olursa, sağdaki gizli ünite aktif hale geleceğinden en alçak anlamlı çıkış ünitesi 0 olacaktır.

Eğer sadece bitlerden birisi 1 ise bu çıkış ünitesinde görülen değer de 1 olacaktır. Ortadaki çıkış ünitesi yüksek anlamlı iki bite ve soldaki gizli üniteye pozitif ağırlıklarla, diğer gizli üniteye ise negatif ağırlıklarla bağlıdır.

Buna göre, Pozitif bağlantılı üç üniteden sadece bir tanesi aktifken, ikinci gizli ünite aktif hale geçemez ve ortadaki

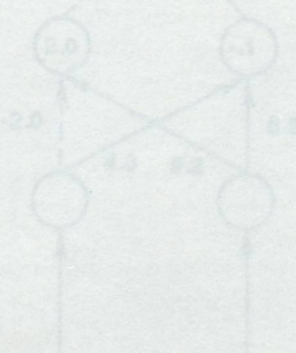
çıkış ünitesi aktif olur. Pozitif bağlantılı ünitelerin iki tanesi aktif hale geçtiğinde gizli ünite aktif olur ve çıkış değeri 0 olur. Üç ünite birden aktif konumda olursa, çıkış değeri 1 olur. Yüksek anlamlı çıkış ünitesi ise, ancak soldaki gizli ünite aktifken, yani elde oluştuğunda aktif olacaktır. Genel olarak, m bitlik iki binary sayıyı toplamak için $2*m$ adet giriş ünitesi, m adet gizli ünite, $m+1$ adet çıkış ünitesi gereklidir.



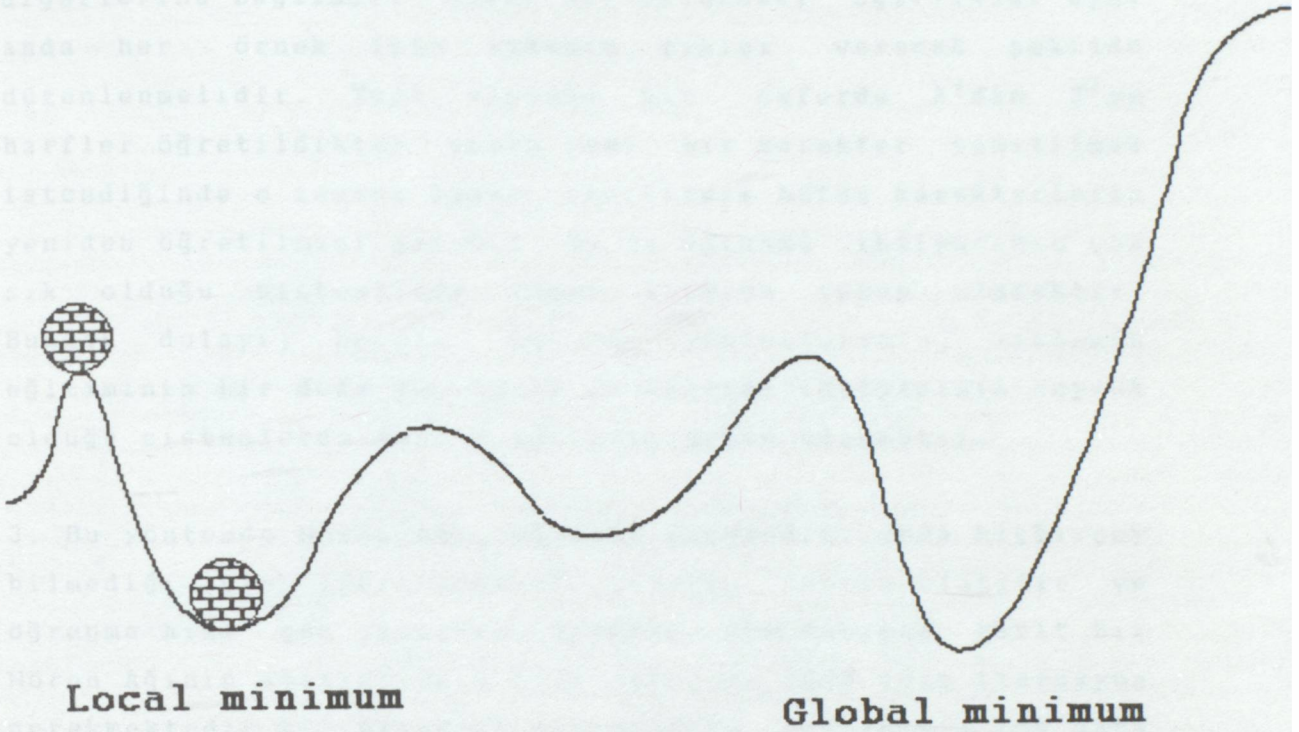
1.3.1.4 HOCALI ÖĞRENMENİN DEZAVANTAJLARI :

Backpropagation algoritması en çok kullanılan ve sonuç elde edilen hocalı öğrenme metodlarından biridir fakat pratikte bu yöntemin ve genel olarak bütün hocalı öğrenme yöntemlerinin bazı dezavantajları vardır. Hocalı öğrenme yöntemleri kullanılmak istendiğinde önce bu dezavantajları göz önünde bulundurmak yararlı olacaktır.

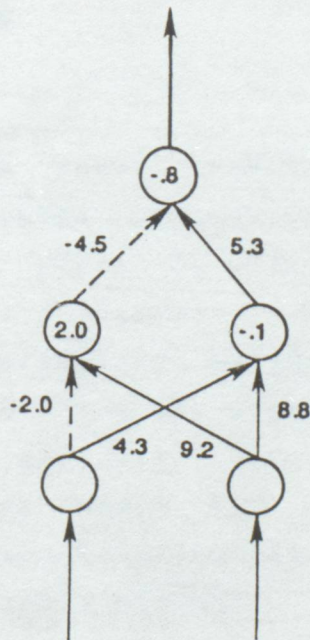
1. Hocalı öğrenmenin en büyük dezavantajı sistemin her zaman istenen sonuca ulaşamayışındır. Başlangıçta bütün ağırlık değerleri ve eşik seviyeleri random fonksiyonla belirlendiği için, eğer bütün ağırlıkların başlangıç değeri eşitse ve çözüm eşit olmayan ağırlıklarla sağlanıyorsa sistem, öğretileni asla öğrenemeyecektir. Çünkü çıkış ünitesine bağlı bütün gizli seviyeler aynı hata sinyalinin üretecektir. Ağırlıkların değişmesi de hata sinyallerine bağlı olduğu için gizli ünitelerden çıkış ünitelerine doğru olan ağırlık değerleri hep aynı kalacaktır. Bu duruma lokal minimum denir. Yani hata maksimum olmasına rağmen bunu küçültmek mümkün olmamaktadır. (Şekil 2.15) Şekil 2.16'da bu şekilde iterasyon sırasında çözümsüz kalmış XOR öğretilmeye çalışılan bir ağ görülmektedir. Sağdaki giriş aktif olduğunda her iki gizli ünite de aktif olmakta ve bu nedenle çıkış değeri 0 olmaktadır. Bu durumda, sistem sağ ünitenin giriş değerinin 1 olduğu her durumda 0 çıkış değerini alır ki bu istenen çözüm değildir.



ŞEKİL 2.15 : LOCAL MINIMUM



ŞEKİL 2.16



2. Öğrenme işleminde öğretilen her örneğin durumu diğerlerine bağlıdır. Nöron ağı üstündeki ağırlıklar aynı anda her örnek için istenen çıkışı verecek şekilde düzenlenmelidir. Yani sisteme bir seferde A'dan Z'ye harfler öğretildikten sonra yeni bir karakter tanıtılmak istendiğinde o zamana kadar tanıtılmış bütün karakterlerin yeniden öğretilmesi gerekir. Bu da öğrenme ihtiyacının çok sık olduğu sistemlerde zaman kaybına sebep olacaktır. Bundan dolayı, hocalı öğrenme yöntemlerinin, sistemin eğitiminin bir defa yapıldığı ve öğrenme ihtiyacının seyrek olduğu sistemlerde tercih edilmesi uygun olacaktır.

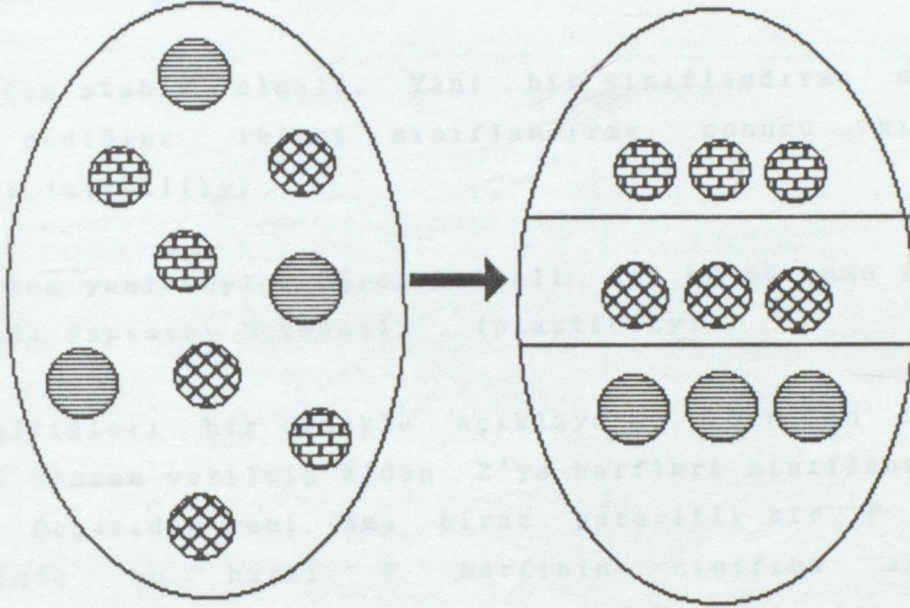
3. Bu yöntemde Nöron Ağı, eğitime başlandığı anda hiçbirşey bilmediği için iterasyonlarla istenen sonuca ulaşılır ve öğrenme hızı çok yavaştır. Örneğin XOR tanıyan basit bir Nöron Ağının eğitimi için bile ortalama 5000 adım iterasyon gerekmektedir ki, karmaşık sistemlerin eğitiminin çok daha uzun zaman gerektireceği açıktır.

Sınıflandırma

1.3.2 HOCASIZ ÖĞRENME

Bu yöntemde sistem kendi kendine öğrenme yeteneğine sahiptir. Bunu, tanıması istenen örnekleri özelliklerine göre sınıflandırarak yapar. Bu aslında insanların da daha önce görmedikleri birşeyi kendi kendilerine, tecrübelerinden yararlanarak anlamlandırmaya çalışmasına benzer. Örneğin daha önce hiç görmediğimiz türde bir kuş gördüğümüzde, kanat, gaga v.s. gibi özelliklerinden dolayı ismini bilmesek bile bunun bir kuş olduğunu anlarız. yaptığımız iş, hayvanı bilgilerimize göre uygun olduğunu düşündüğümüz sınıfa dahil etmektir.

Hocasız öğrenme metodunda, ilk gelen örnek bir sınıfa yerleştirilir. Sonra gelen örneğin özellikleri farklı ise yeni bir sınıfa değilse aynı sınıfa alınır. (Şekil 2.17)



Sınıflandırma

ŞEKİL 2.17

1.3.2.1 ADAPTİF REZONANS TEORİSİ

Sınıflandırarak tanıma için kullanılan çeşitli yöntemler vardır . Bunların içinde en çok kullanılan yöntemlerden birisi Adaptif Resonans Teorisidir. Adaptif Resonans Teorisinin ana prensipleri şunlardır:

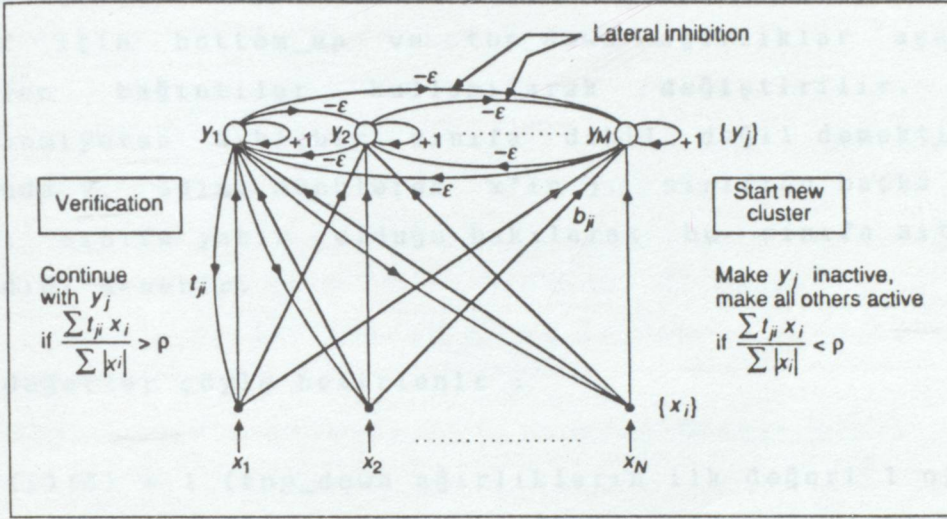
1. Sistem stable olmalı. Yani bir sınıflandırma sonucu beyaz dediğine ikinci sınıflandırma sonucu kırmızı dememeli.(stability)
2. Sistem yeni şeyler öğrenebilmeli ama bu öğrenme işlemi sırasında yapısını bozmamalı . (plasticity)

Bu özellikleri bir örnekle açıklayalım. Örneğin sistem hatasız olarak verilmiş A'dan Z'ye harfleri sınıflandırmış olsun. Dışarıdan yeni ama biraz parazitli bir F harfi geldiğinde bu harfi F harfinin sınıfına alırken adaptasyonla F sınıfının bilgilerini değiştiriyorsa, bozuk F harfleri geldikçe sistem gittikçe esas F harfinden uzaklaşacak ve en sonunda belki de sisteme F harfi gösterildiğinde F harfini F harfi için başlangıçta açtığı sınıfa yerleştiremeyebilecektir. Bunu önlemek için stability-plasticity dengesini iyi ayaralamak gerekir.

Adaptif Rezonans Teorisinde bu denge ayarlandığı takdirde sınıflandırırken esas şekil bozulmamış olur.

Şimdi bu teorinin algoritmasını inceleyelim.

Adaptif Resonans Teoride stability-plastisity dengesini sağlamak amacıyla sistem önce ileriye doğru sonra geriye doğru çalıştırılmakta ve bunun için iki ayrı ağırlık grubu(bottom_up ve top_down) kullanılmaktadır .(Şekil 2.18)



ŞEKİL 2.18

1. Sisteme giriş bilgisi olarak x örneği verilir.
2. Bottom_up proses ile ağırlıklı toplamalar belirlenir.

$$y(j) = (b(j)(i) * x(i))$$

3. MAXNET yöntemiyle y değerleri arasından en büyük değeri alan $y(j)$ belirlenir.

4. Top_down proses ile x in gerçekten j . sınıfa dahil olup olmadığına bakılır. Bunun için top_down ağırlıklar kullanılarak ağırlıklı toplam değeri hesaplanır :

$$MAXNET(t(j)(i) * x(i))$$

Eğer daha önceden belirlenmiş v vigilance parametresi (tolerans parametresi) için:

$$1.2.1.1.1 MAXNET$$

$$(t(j)(i)*x(i))/(\sum (x)) > v$$

ise x gerçekten j . sınıfa dahil demektir. Bu durumda j . sınıf için bottom_up ve top_down ağırlıklar aşağıdaki verilen bağıntılar kullanılarak değiştirilir. Şart sağlanmıyorsa x hiçbir sınıfa dahil değil demektir. Bu durumda 2. adıma dönülerek x 'in j . sınıftan başka en çok hangi sınıfa yakın olduğu bakılarak bu sınıfa ait olup olmadığı denenir. göre x 'in Hopping Distance'ı en yüksek sınıfa yerleştirilmelidir. Bu ifadeyi aşağıdaki bağıntılar

ilk değerler şöyle belirlenir :

$$t(j)(i)(0) = 1 \text{ (top_down ağırlıkların ilk değeri 1 olsun)}$$

$$b(j)(i)(0) = 1 / (1 + N) \text{ (bottom_up ağırlıkların ilk değeri)}$$

Bu ağırlıkların yenilenmesi sırasında ise şu bağıntılar kullanılır :

$$t(j)(i)(n+1) = t(j)(i)(n) * x(i)$$

$$b(j)(i)(n+1) = t(j)(i)(n)*x(i)/(0.5+(\sum_{i=1}^n \sum_{j=1}^m t(i)(j)(n)*x(i))$$

Görüldüğü gibi $t(j)(i)$ ve $b(j)(i)$ 'nin kullanım amaçları birbirinden farklıdır. $b(j)(i)$ x 'in en büyük ihtimalle hangi sınıfa dahil olabileceğini belirlemekte kullanılır. $t(j)(i)$ ise, x 'in gerçekten bu sınıfa ait olup olmadığını anlamak için kullanılır.

Sınıflandırma duyarlılığı vigilance parametresinine bağlıdır. Vigilance parametresi yüksek alınırsa sınıflandırma daha seçici olacaktır.

Şimdi sınıflandırma için kullanılan yöntemlerden biri olan

MAXNET'i inceleyelim :

1.3.2.1.1 MAXNET

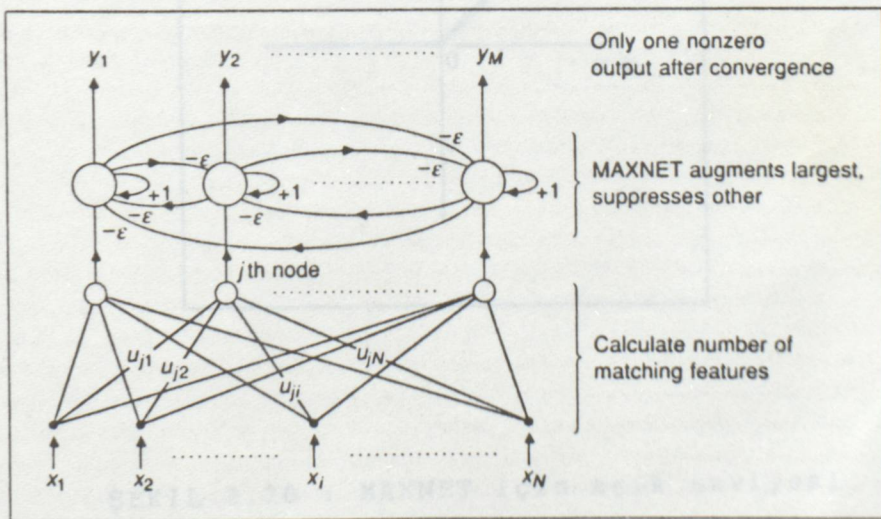
x ikili düzende sayı değeri alabilen örneği ; c mevcut sınıfları ; u da bu sınıflardaki örnekleri gösterebilir. MAXNET, x giriş bilgisinin mevcut sınıflardan birine mi yoksa yeni bir sınıfa mı ait olduğunu bulmak için kullanılan yöntemdir. Bunun için yapılan işlem, x ile c sınıfları arasındaki Hamming Distance'lara bakarak elde edilen değerlere göre x 'i Hamming Distance'ın en küçük olduğu sınıfa yerleştirmektir. Bu ifadeyi aşağıdaki şekilde gösterebiliriz :

iff Hamming Distance($u(j), x$) < Hamming Distance($u(k), x$)
for all $k=1, 2, \dots, m$ $k \neq j \implies x \in c(j)$

N : örnekteki özellik sayısını gösteriyorsa, x girişi ile $u(j)$ arasındaki Hamming Distance :

$$\text{Hamming Distance}(x, u(j)) = N - \sum_{j=1}^N \sum_{i=1}^N (u(j)(i), x(i))$$

olacaktır. MAXNET'in mimarisini gösteren yapı Şekil 2.19'deki gibidir :



MAXNET'te $t(j)(k)$, j . düğümünden k . düğüme bağlantı ağırlığı olsun:

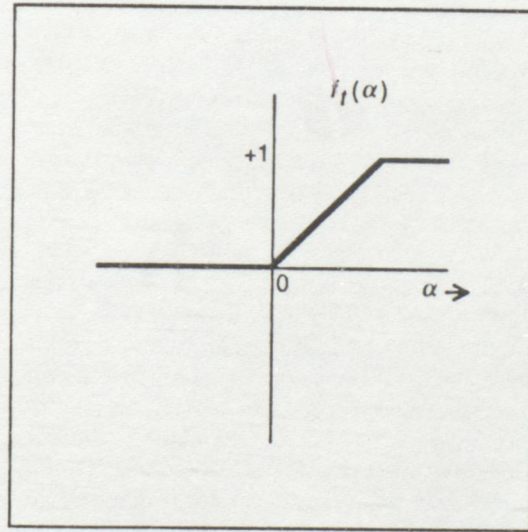
$$t(j)(k) = 1 \text{ if } j=k$$

$$t(j)(k) = -E \text{ if } j \neq k \text{ and } E < (1/M)$$

$t+1$ zamanında j . düğümün çıkış değeri :

$$M(j)(t+1) = f(t) * (M(j)(t) - E * (M(k)(t))) \quad j, k = 1..M$$

Buna göre $c > 0$ olduğu sürece $M(j)(t+1) > 0$ olacak, $c < 0$ durumu için bütün $M(j)$ çıkışları negatif değer alacaktır. Böylece, sadece diğerlerine göre en büyük olan çıkış değeri arttırılacak, diğer bütün çıkışlar bastırılacaktır (rich gets richer, contrast enhancement metodu). Bu yapının eşik seviyesi aktivite fonksiyonu şekil 2.20'deki gibidir.



ŞEKİL 2.20 : MAXNET için eşik seviyesi

HOCASIZ ÖĞRENMENİN DEZAVANTAJLARI :

Hocasız öğrenme yeteneği kazandırılmış nöron ağlarında, sistemin kendi kendine öğrenebilmesi bir üstünlük olmakla birlikte baştan sona yardımsız sınıflandırmanın bazı dezavantajları da vardır. Bu dezavantajlardan birincisi yukarıda da bahsedildiği gibi yeni gelen ve eski sınıflara dahil edilebilecek örneklerin sınıfın yeniden düzenlenmesi sonucu doğru değerleri bozabileceğidir. Yani sistem ilk sınıflandırılmada 'D' harfi dediğine bir kaç sınıflandırma sonra 'O' diyebilir. Bunun için sistemi yeni şeyler öğrenebilecek ama bu öğrenme işlemi sırasında yapısını bozmayacak şekilde kurmak gerekir. İkinci bir problem de sistem her gördüğü şekli bir sınıfa dahil etmek isteyeceğinden parazitleri de sınıflandırabilecek olmasıdır.

3.1 GİRİŞ

Bir görüntüyü anlamlandırma işlemi iki aşamada olur :

- * Görüntü İşleme

- * Görüntü Tanıma

Görüntü İşlemede amaç, ham görüntüden anlamlı bir bilgi elde edilmesini sağlamaktır. Görüntü Tanımada ise elde edilen bu bilgiler değerlendirilir. Early Processing olarak da isimlendirilen görüntü işlemeden sonra işlemde geçmiş bu görüntüden şekil tanıma yararlanır.

En basit bir Görüntü İşleme Sistemi :

- * Görüntü sağlayan bir cihaz,

- * Görüntü bellek

- * Bu belleğe erişen bir bilgisayar

- * Bellekte oluşan sonuçları gösteren bir ekrandan ibarettir.

Bu çalışmada, scanner aracılığıyla alınan gri tonunda resimler üzerinde çalışılmıştır. Scannerdan alınan resimler bellekte TIF(TAG Image File) dediğimiz bir formda saklanır. Görüntü işleme algoritmalarını uygulayabilmek için öncelikle bu formatı çözmek gerekir. Bir gri ton fonksiyonunu, görüntü olarak isimlendirebileceğimiz iki boyutlu bir dizi olarak düşünürsek, bu görüntünün en küçük birimi pixel(picture elements) olarak isimlendirilir. Her pixel'in sayısal değerini gray level'ı belirler. Format çözümü ile Tif file'dan resmin satır ve sütun miktarı, kaç gri seviyeden oluştuğu, file içinde pixeller halinde saklanan data'nın başlangıç yeri ve uzunluğu gibi bilgiler öğrenilir.(EK 2)

3.2 GÖRÜNTÜ İŞLEME ALGORİTMALARI

3.2.1 POINT PROCESS:

Bu algoritmada, görüntünün her noktasındaki pixellerin yeni değeri, pixelin sadece kendi eski değerine bakılarak hesaplanır. Hesaplama sonucu görüntü daha çok parlaklaştırılabilir veya koyulaştırılabilir. Örneğin her pixelin değeri 40 arttırılarak görüntünün parlaklığı arttırılabilir.

Pixel değeri ile birlikte pixelin adresi de verilirse gölgeleme veya şekli yumuşatma (smoothing) sağlanmış olur. Örneğin, şeklin ortası daha parlak, kenarları ise daha koyu hale getirilebilir. Point Process algoritmalarına yardımcı olarak, INTENSITY HISTOGRAM'ları oluşturulabilir. Bu, şeklin renk tonlarının dağılımı ile ilgili bir diagramdır ve point process ile ilgili parametrelerin hesaplanmasında kullanılır. Intensity'nin, görüntünün bir noktasındaki ışık parlaklığı olduğu düşünülürse, Intensity Histogram ile bir görüntü bölgesinde her pixelin kaç defa kullanıldığı hesaplanır. Bu diagramın hesaplanması için mümkün her pixel değerini karşılayabilecek bir dizi tanımlanır. Eğer 8 bitlik pixel değeri kullanılıyorsa, bunun karşılığı 256 farklı gri tonudur. Dolayısıyla 256 elemanlı bir dizi yeterli olacaktır. Her pixelin değerine bakılarak, dizide kendi değerine karşılık düşen gözün değeri bir arttırılır. Bu hesaplama sırasında, her seferinde yalnızca bir pixelin değerine bakıldığı için, görüntü üzerinde hiçbir değişiklik yapılmamasına rağmen, konu Point Process sınıfına girer. Bu histogram, görüntünün kontrastının değiştirilmesinde kullanılabilir. Belli bir eşik değeri seçildikten sonra 0 no'lu histogram değerinden başlayarak bu eşik değerini ilk aşan histogram noktası tesbit edilir. Bu işlem 255'den

geriye doğru bir kez daha tekrarlanınca elde edilen bu iki değer Point Process algoritmasında parametre olarak kullanılabilir. Bu iki değer, gerçekte şekildeki en çok pixel miktarını içeren en dar banttır. Bu işlem görüntüde bazı değerlerin istek dışı olarak kaybolmasına sebep olabilir. Bu iki histogram değerinin arasında kalan detaylar ortaya çıkarken bunun dışında kalanlar ise kaybolur. Böylece eşik değerlerinin her şekle göre farklı tesbit edilmesi zorunluluğu ortaya çıkar. Bunun yanında, algoritmaların hangi sırayla uygulanması gerektiği de belirlenmelidir. Çünkü bazı algoritmalar detayları yok etmekte, bazıları ise arttırmaktadır. Dolayısıyla, algoritmalar amaca en uygun ve ana şekli en az bozacak sırayla seçilmelidir.

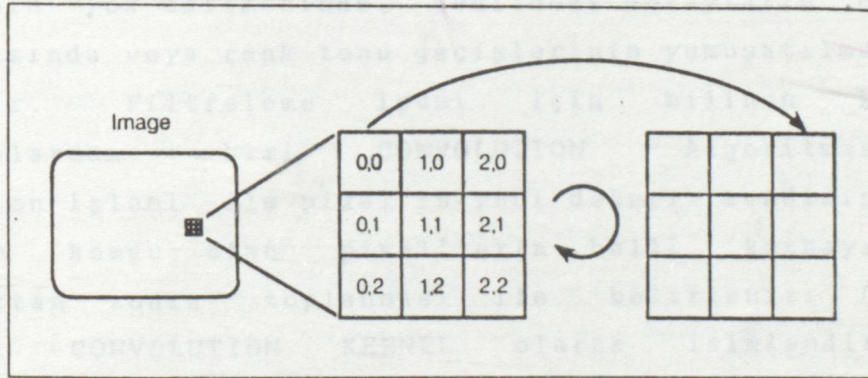
Point Process'in uygulamalarından birisi de PSEUDOCOLORING'dir. Bu uygulamada, gri tonlardan oluşan bir şekilde, belli bir aralıktaki pixeller gri tonlarından farklı olarak kırmızı, yeşil veya mavi renklerle gösterilirler. Örneğin pixel değeri 100 ile 200 arasında değişen bölgelerin renklerinin yeşil yapılması bu tür bir uygulamadır. Bu bölgelerin dışında kalan bölgeler, değiştirilmeden gri tonlarıyla ifade edilirler. Point Process ile en az 256×256 pixelin işleneceği düşünülecek olursa, bu her pixel için en az iki bellek erişimi, bir çarpma ve bir de toplama anlamına gelir. Bu tür hesaplamalar için LUT'dan yararlanılarak her pixel için ayrı ayrı işlem yapılması önlenmiş olur. Bunun için her pixele karşılık düşen sonuç değeri hesaplanarak LUT'a yerleştirilir. Gerktiğinde hesaplanacak pixel adres olarak kullanılarak tablonun o adresinden sonuç değeri bulunur.

3.2.2 SINIFLANDIRMA

Görüntü elemanlarını sınıflandırmak için görüntü elemanlarını belirleyici özellikleri tanımlamak gerekir. En basit tanımlama ile, bir eleman aynı değerlere sahip, birbirine bağlı pixeller grubudur. Eğer bir pixel kendisine 0,90,180 veya 270 dereceden komşu olan pixeller ile aynı değerde ise o grubun elemanıdır. Bu sınıflandırma işlemi için bütün pixellere 0-255 arası sayısal değerler verilir. Belli sayıdan fazla pixelden oluşan görüntü elemanları anlamlı kabul edilirler. Belirlenen bir eşik seviyesinden az pixelden oluşan görüntü elemanları ise parazit veya hata olarak düşünülüp dikkate alınmazlar.

3.2.3 GEOMETRİK İŞLEMLER

Geometrik işlem algoritmaları pixellerin yerlerini değiştirmek için kullanılırlar. Genelde bakış açısını değiştirmek, kameralardaki distorsiyonu düzeltmek veya istenen bölgelerin boyutlarını değiştirmek için kullanılırlar. Tipik bir geometrik işlem, görüntüyü döndürme, büyütme, daraltma, genişletme ve düzenleme ile değiştirerek tanınmasını kolaylaştırmaya çalışır. Yapılan iş, (x,y) koordinatlarındaki pixelin (x',y') koordinatlarına taşınmasıdır. Örneğin her yatay görüntü dikey olarak veya başka bir açı ile yeni yerine yerleştirilebilir. Fakat bunun için yapılacak bütün işlemler bir buffer'da yapılmalıdır. Aksi takdirde, her yeni işlem bir öncekinden etkileneceği için doğru sonuç sağlanamayacaktır. Şekil 3.1 de 90 derecelik dönme geometrik işlemi ile $(0,0)$ noktasının yer değiştirip yeni yerinin $(2,0)$ noktası olacağı gösterilmiştir. Burada $x'=(Y\text{SIZE}-1)-y$ ve $y'=x$ dir.



ŞEKİL 3.1 : GEOMETRİK İŞLEM

3.2.4 FRAME İŞLEMLERİ

Birden fazla görüntü kullanan algoritmalar frame işlemleri olarak adlandırılırlar. Örneğin bir görüntüden başka bir görüntü arasında çıkarma işlemi yapmak

bu tür bir iştir. Elde edilen sonuçtaki farklılıklar iki görüntüyü karşılaştırmak için kullanılabilir. Bu sonuçtan yararlanarak karşılaştırılan görüntüdeki eksik kısımlar veya parazitten dolayı eklenen hatalı fazlalıklar tesbit edilebilir. Yani Frame işlemler, görüntü kalitesini arttırmak için de kullanılabilir.

3.2.5 ALAN İŞLEMLERİ

Bu işlemlerde pixellerin değerleri, kendilerine komşu olan pixellerin değerleri de hesaba katılarak değiştirilir. Bu algoritma daha çok, istenmeyen veya tekrarlanan pixellerin filtrelenmesinde kullanılır. Bu filtreleme türü, örneğin A/D dönüşümü sırasında oluşan istenmeyen algılama hatalarının yok edilmesinde, şekildeki detayların ortaya çıkarılmasında veya renk tonu geçişlerinin yumuşatılmasında kullanılır. Filtreleme işlemi için bilinen klasik algoritmalarından biri CONVOLUTION Algoritmasıdır. Convolution işlemi ile pixel'in yeni değeri kendisinin ve kendisine komşu olan pixel'lerin belli katsayılarla çarpıldıktan sonra toplanması ile belirlenir. Çarpım katsayısı CONVOLUTION KERNEL olarak isimlendirilir. Örneğin, 3x3'lük bir pixel matrisi düşünelim. Bu durumda, bir pixel'in yeni değerinin hesabı için kendisi ve kendisine en yakın sekiz komşusu hesaba katılacaktır. $x=0,1,2$ için hesaplanacak pixel $p(x,y)$ kernel noktası $k(x,y)$ ise:

$$p(x,y) = \text{Toplam}(k(m,n)*p(x+m,y+n))$$

Örneğin (1,1) noktasındaki pixelin yeni değeri:

$$\begin{aligned} p(1,1) = & p(0,0)*k(0,0) + p(1,0)*k(1,0) + p(2,0)*k(2,0) \\ & + p(0,1)*k(0,1) + p(1,1)*k(1,1) + p(2,1)*k(2,1) \\ & + p(0,2)*k(0,2) + p(1,2)*k(1,2) + p(2,2)*k(2,2) \end{aligned}$$

veya kısaca :

$$p(1,1) = \text{Toplam}(k(m*n)*p(m*n)) \text{ dir.}$$

Tüm şeklin filtrelenebilmesi için yukarıdaki işlemin şekildeki her pixele tek tek uygulanması gerekir. (Şekil 3.2) Konvolüsyon işleminin uygulanacağı alanın boyutu $X*Y$ ve kernelin boyutu $m*n$ olarak düşünülürse, $X*Y*n*m$ defa çarpma ve toplama işlemi yapılır. Örneğin $256*256$ 'lık bir görüntü için $3*3$ 'lük bir kernel kullanılırsa 589824 kez çarpma ve toplama işlemi yapılması gerekir. Bu işlemler oldukça uzun zaman alır.

Kernel matrisi tüm görüntü üzerinde dolaştırıldığında, kenarda kalan bölgelerin yeni değerleri algoritmanın özelliğinden dolayı hesaplanamayacaktır. Örneğin $3*3$ 'lük bir Kernel için, bütün görüntünün kenarlarında 1 pixel genişliğinde bir bantın değeri hesaplanamayacaktır. Bu bantın genişliği Kernel boyutunun yarısı kadardır. Bu bant ihmal edilebileceği gibi, anlamlı değeri olan en yakın komşusunun değeri de verilebilir.

Kernel değerine bağlı olarak, konvolüsyon sonucu 0-255 sınırının dışına taşabilir. Görüntülemek için uygun olmadığından bu değerlerin düzeltilmesi gerekir. Bu durumda

point process kullanılarak belli bir öteleme yapılabilir veya kaybolan değerler sınır değerlerine eşit alınabilir. Negatif sonuçların değerlendirilebilmesi için, işaret bitinin de korunması gerekir. Buna göre, 8 bit pixel'ler ve 3x3 kernel için toplam ve işaret biti için 12 bite gerek olacaktır.

Kernel'in mutlaka kare matris olması gerekmez. Örneğin bir görüntüdeki sadece dikey kenarları ortaya çıkarıp, diğer bölgeleri yok etmek için aşağıdaki kernel kullanılabilir:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Üniform bir görüntü alanında, bütün pixel değerlerinin aynı olduğu bir bölgeye yukarıdaki kernel uygulandığında pixel'lerin çıkış değeri sıfır olacaktır.

Yatay kenarları kuvvetlendirip belirginleştirmek ve diğer ayrıntıları filtrelemek istendiğinde ise aşağıdaki kernel kullanılabilir:

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Daha büyük kerneller kullanılarak, bir görüntüdeki benzer modeller yakalanabilir. Böyle kerneller genellikle TEMPLATE olarak isimlendirilir.

Bir görüntüde frekans, aynı tondaki pixellerin arka arkaya tekrarlanmış yoğunluğudur. Frekansın belli bir bandını belirlemek ve seçmek isteniyorsa bunu sağlayan bir kernel yazılabilir. Örneğin, ani parlaklık değişimlerini tesbit için bir yüksek frekans filtresi kullanılabilir. Bunun için kullanılacak kernel aşağıdaki gibidir:

-1	-1	-1
-1	8	-1
-1	-1	-1

Bu kernel LAPLACIEN filtresi olarak isimlendirilir. Bu kernelin uygulandığı görüntülerde, yüksek frekansa sahip kenarlar, yani ani parlaklık değişiklikleri ortaya çıkarılır; diğer alçak frekanslar ise ortadan kalkar. Bundan dolayı bu kernel KENAR BELİRLEYİCİ olarak adlandırılır.

Eğer Laplacien kernelinin merkez değeri olan 8'i 9 yaparsak, yani Kernelin yeni hali aşağıdaki gibi olursa:

-1	-1	-1
-1	9	-1
-1	-1	-1

Bu filtre de ani parlaklık değişikliklerini belirler. Farklı olarak, frekansın değişmediği bölgelerin pixel değerleri aynı kalır. Bu kernel, frekansın yüksek olduğu bölgeleri seçerek belirginleştirdiği için sonuçta oluşan kernel'in daha keskin hatları olur.

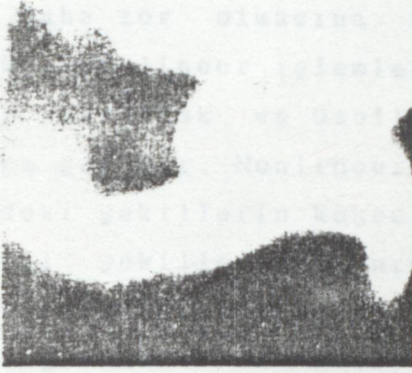
Uygun kerneller kullanılarak alçak frekans filtreleri de elde edilebilir. Bu tür filtreler kullanıldığında orijinal

şekil bulanıklaşır. Hatalar kaybolur. Kullanılabilecek kernel:

1/10	1/10	1/10
1/10	2/10	1/10
1/10	1/10	1/10

Şekil 3.3'de çeşitli alçak geçirgen kerneller uygulanmış bir resim görülmektedir.

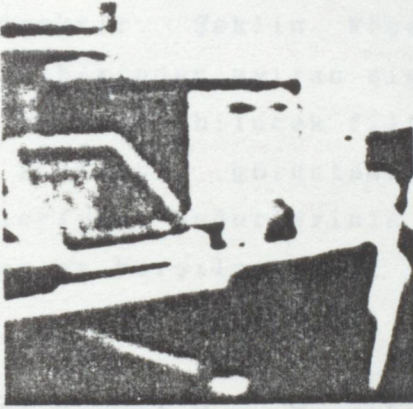
ŞEKİL 3.3 : KERNELLE GÖRÜNTÜ DEĞİŞİKLİKLERİ



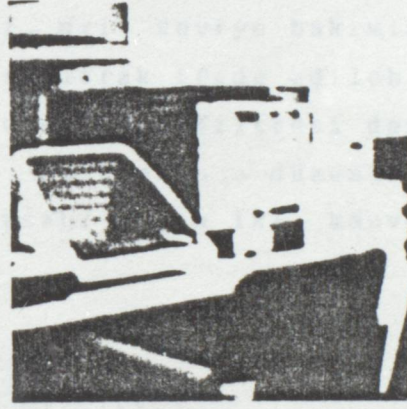
(a)



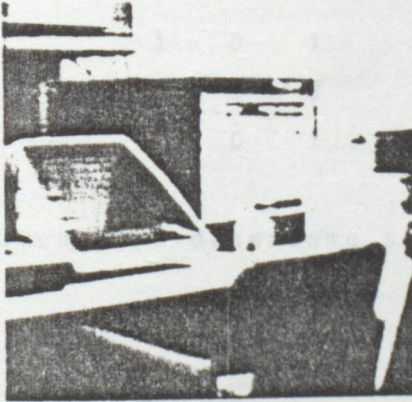
(b)



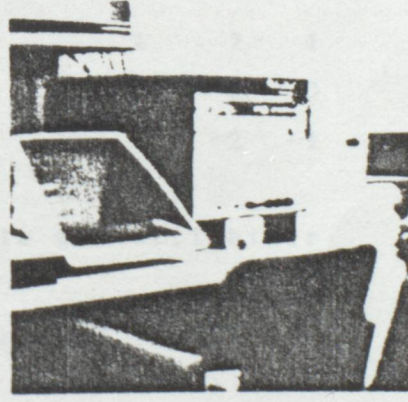
(c)



(d)



(e)



(f)

3.2.5 KÖŞE BELİRLEME, LİNEER OLAMAYAN ALAN İŞLEMLERİ

Lineer bir işlem olması sebebiyle konvolüsyonun kullanımı ve analizi kolaydır. Nonlinner işlemler ise konvolüsyon'dan biraz daha zor olmasına rağmen çok daha kullanışlı ve üstündür. Nonlineer işlemlerde görüntü elemanlarındaki hata oranını yakalamak ve özellikleri belirlemek için çok daha az işlem gerekir. Nonlineer alan işleme genel olarak resim üzerindeki şekillerin köşesini belirlemede kullanılır.

Resimdeki şekillerin tanınıp anlamlandırılabilmesi için öncelikle resim üzerindeki yerlerinin belirlenmesi gerekir. Bunu sağlamanın en kolay yolu şekilleri fon renkten belirgin olarak ayıracak şekilde köşe değerlerini belirlemektir. Şeklin köşesi, gri seviye bakımından iki alanı birbirinden ayıran sınır olarak ifade edilebilir. Bu amaçla kullanılabilecek filtreye Sobel filtresi denir.

Sobel filtresi görüntüdeki köşelerin düzenlenmesi ve kuvvetlerinin değerlerinin hesabı için iki konvolüsyonun sonuçlarını karşılaştırır.

Eğer iki kernel X ve Y aşağıdaki gibi ise:

$$\begin{array}{rcccl} & -1 & 0 & 1 & & 1 & 2 & 1 \\ X : & -2 & 0 & 2 & Y : & 0 & 0 & 0 \\ & -1 & 0 & 1 & & -1 & -2 & -1 \end{array}$$

köşe kuvveti ve düzenleme şöyle gösterilebilir:

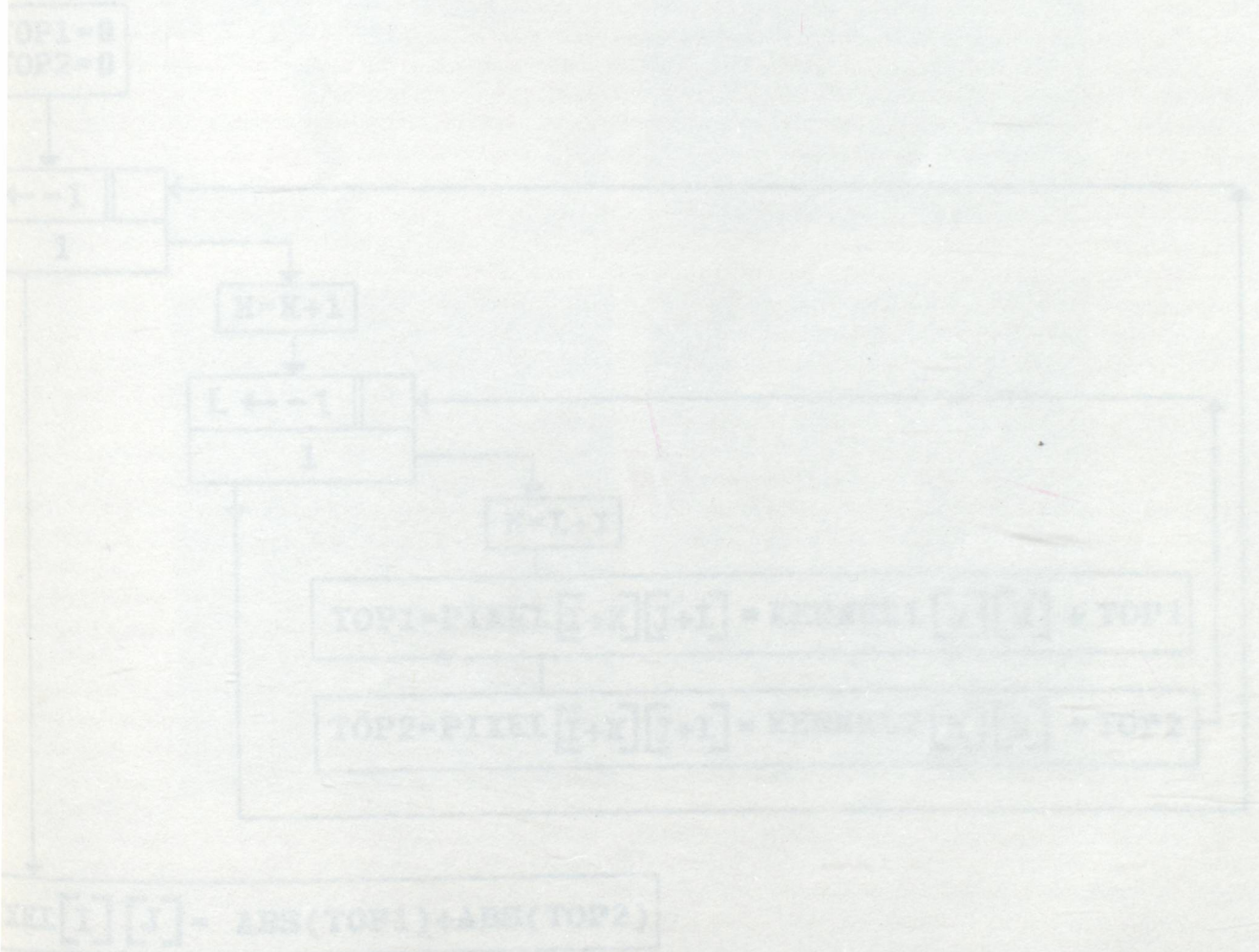
$$\text{kuvvet} = \text{sqrt}(X*X + Y*Y)$$

$$\text{düzenleme} = \text{arctan}(Y/X)$$

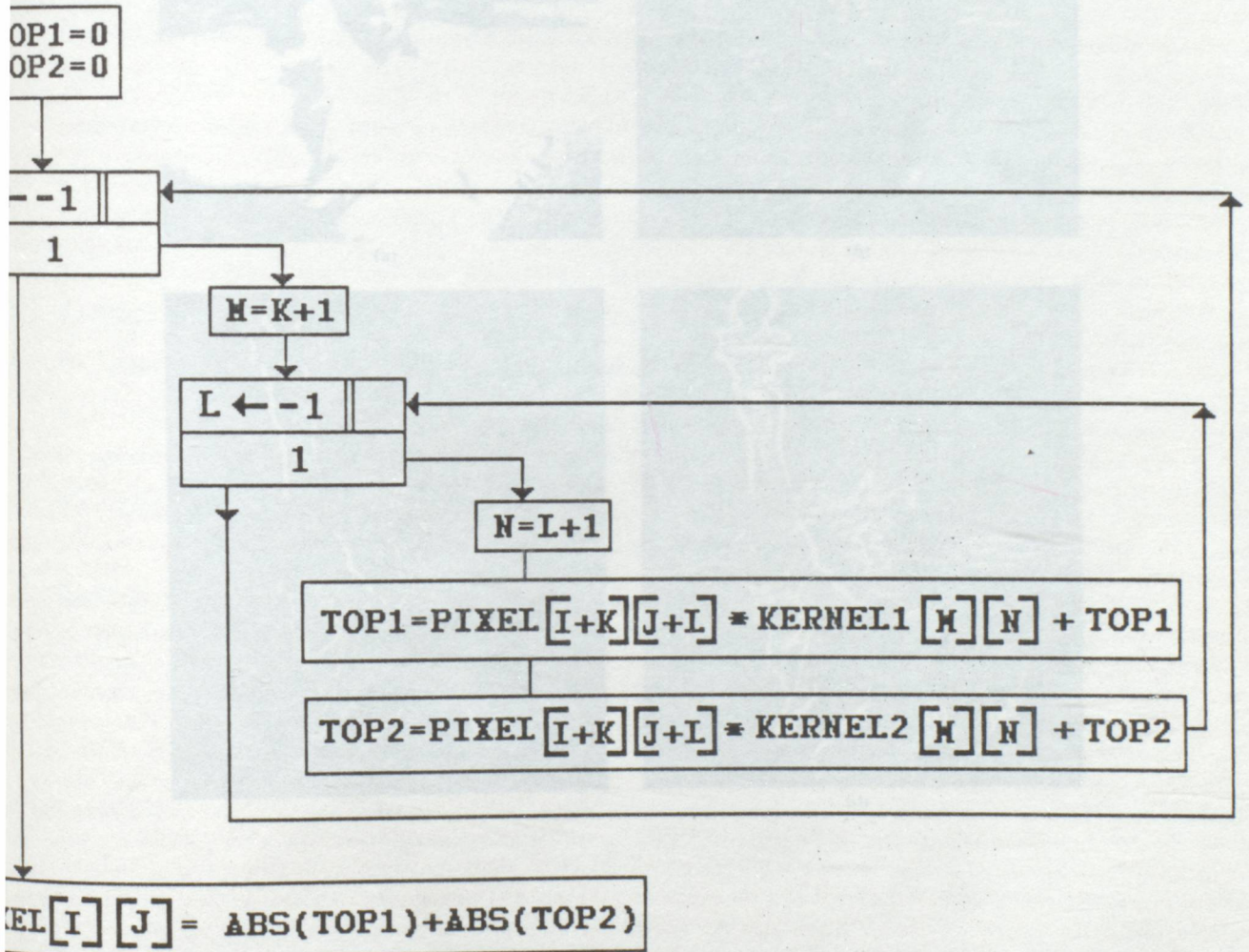
Köşe belirleme işlemi sonunda, şeklin köşeleri beyaz,

diğer bütün bilgiler siyah renkte olacaktır. Şekil 3.4'de bu yöntemin akış diagramı gösterilmiştir.

Şekil 3.5 'deki (a) şekline x eksenindeki köşeleri belirginleştiren birinci kernel uygulandığında (b) şekli elde edilir. (a) şekline y eksenindeki köşeleri belirginleştiren ikinci kernel uygulandığında (c) şekli elde edilir. İki kernel birlikte uygulandığında ise (d) şekli elde edilmiş olur.



ŞEKİL 3.4 : KÖŞE BELİRLEME ALGORİTMASI



ŞEKİL 3.5 : KÖŞE BELİRLEME UYGULAMASI



3.2.6 KONTUR BELİRLEME

KONTUR BULMA

Şekillerinin köşeleri belirginleştirilmiş resim, şekillerin koordinatlarının belirlenmesi işlemine hazırdır. Bunun için kullanabilecek yöntemlerden birisi şöyledir :

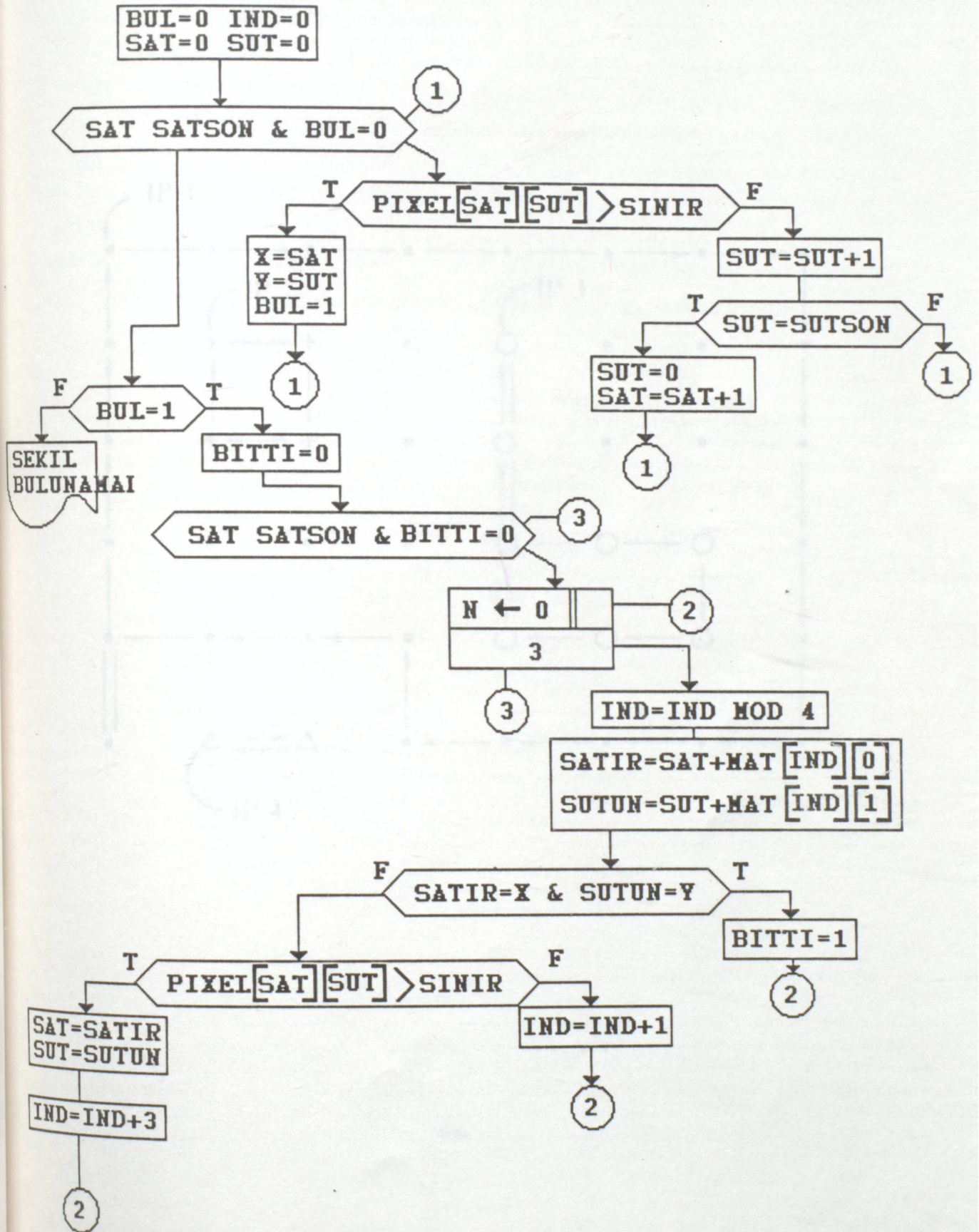
1. Önce şekil üzerindeki noktalardan birinin yerinin bulunması
2. Bulunan bu noktadan hareket ederek komşu noktaların yerlerinin bulunması

Açıklarsak : Önce üzerinde köşe belirleme algoritması uygulanmış resim üzerinde köşeye ait ilk nokta bulunur.

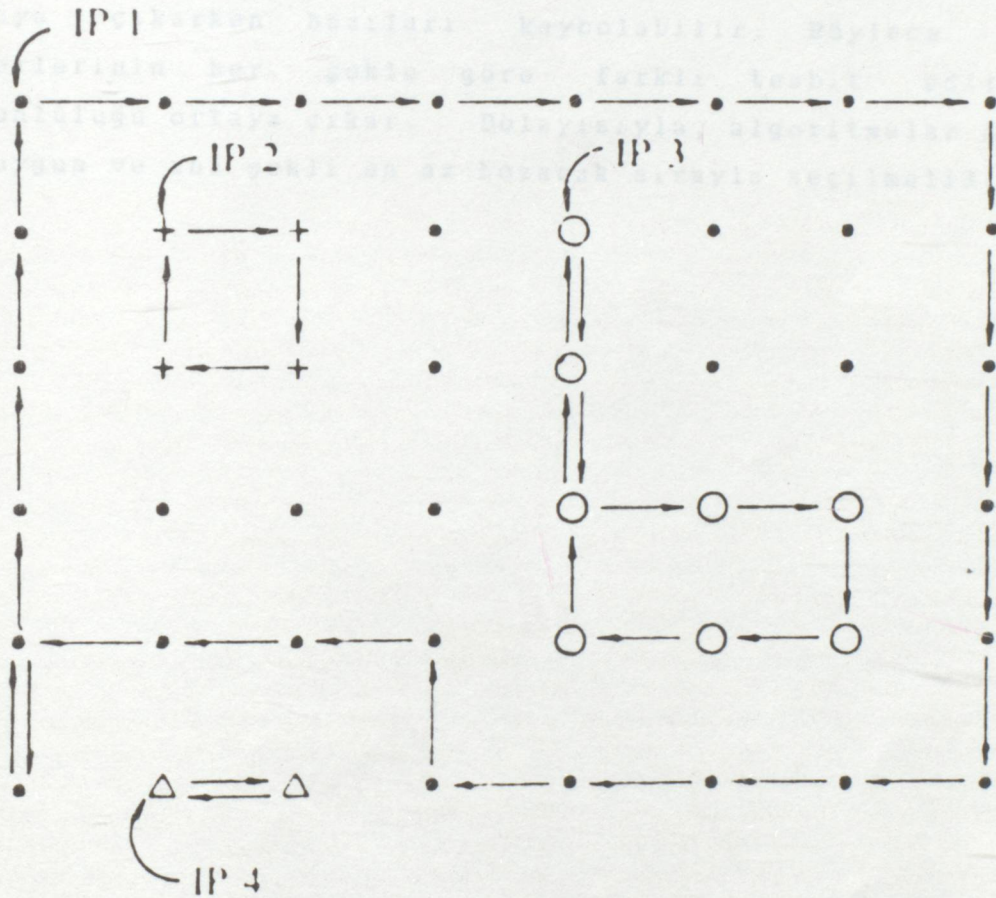
Daha sonra bu noktaya komşu sekiz noktanın gri seviyesine bakarak gri seviyesi bu noktayla aynı değerde nokta olup olmadığına bakılır. Eğer komşu varsa komşunun şekle ait, bulunan ilk nokta olup olmadığı kontrol edilir. Eğer ilk noktaysa işleme son verilir. Değilse yeni noktanın komşuları aranır. Eğer komşu nokta bulunamazsa şeklin parazit olduğuna karar verilir.

Şekil 3.6 'da bu algoritma gösterilmektedir.

KONTUR BULMA



ŞEKİL 3.7 : 4 FARKLI ŞEKLİN KONTURLARININ BELİRLENMESİ



SONUÇ

Yapay Zeka araştırmacılarının makinalara görme yeteneğini kazandırma yolundaki çalışmaları ile Görüntü İşleme'nin önemi daha da artmıştır. Bu çalışmalarda amaç görüntüyü tanınmaya uygun hale getirmek olduğuna göre, Görüntü İşleme Algoritmalarından yararlanan kişinin algoritmaların hangisini hangi sırayla seçeceğine çok iyi karar vermesi gerekir. Yapılan işlemler görüntüde bazı değerlerin istek dışı olarak kaybolmasına sebep olabilir. Bazı detaylar ortaya çıkarken bazıları kaybolabilir. Böylece eşik değerlerinin her şekle göre farklı tesbit edilmesi zorunluluğu ortaya çıkar. Dolayısıyla, algoritmalar amaca en uygun ve ana şekli en az bozacak sırayla seçilmelidir.

4.1 GİRİŞ

Bu çalışmada gerçekleştirilen Kromozom Tanıma Programı iki bölümden oluşmaktadır:

1. Görüntü İşleme
2. İşlemlerden geçmiş görüntü bilgilerinden yararlanarak şekil tanıma

4.1.1 GÖRÜNTÜ İŞLEME

4.1.1.1 PARAZİTLERİN TEMİZLENMESİ

Scanner aracılığıyla bilgisayara aktarılan kromozom resimleri incelendiğinde hem resmin kendisinden gelen, hem de bilgisayara aktarma sırasında oluşan bazı parazitler görülür. Tanıma işlemi sırasında bu parazitlerden dolayı meydana gelecek hataları önlemek için önce 3. bölümde incelenen ALAN İŞLEME ykernel matrislerinden biri kullanılarak görüntü belirginleştirilip parazitler mümkün olduğunca silik duruma getirilmiştir.

4.1.1.2 KÖŞE BELİRLEME :

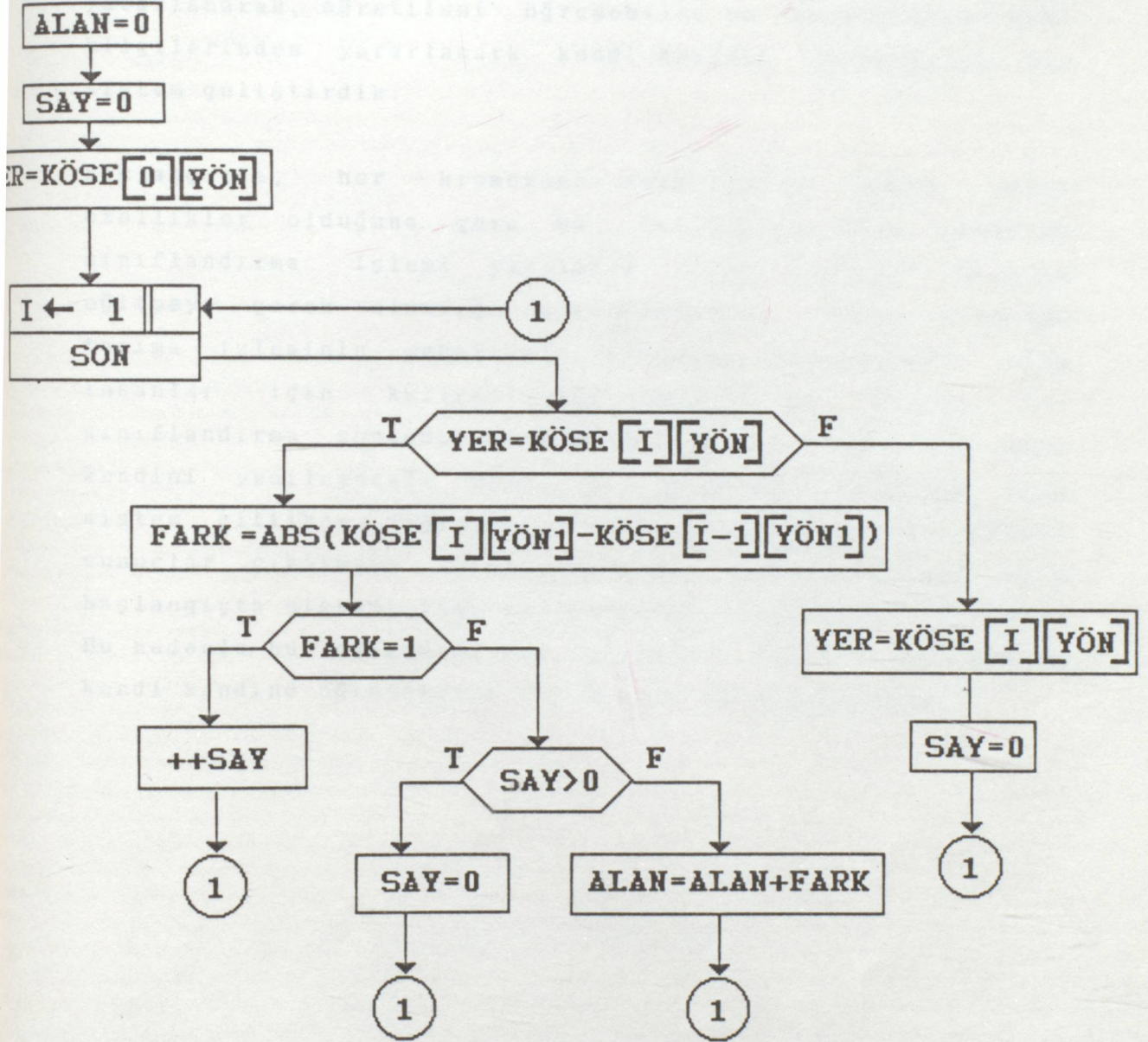
Bilgisayara aktarılan resim üzerindeki şekillerin tanınması işlemine geçilebilmesi için öncelikle şekillerin yerlerinin belirlenmesi gerekir. Bunun için Şekil 3.4'de görülen Köşe Belirleme Algoritması kullanılmıştır.

4.1.1.3 KONTUR BELİRLEME :

Köşeleri belirginleştirilmiş şekillerin çerçevelerini belirleyen koordinatları bulmak artık kolaydır. Bu çalışmada biz şeklin çevresini belirleyen koordinatları bulmak için önce şeklin bir pixel'inin yerini belirledikten sonra 8 komşusunun durumuna bakarak koordinat belirleme algoritmasını denedik. Fakat köşelerin kalınlığı bir pixel'den fazla olduğu için bu yöntemden başarılı sonuç elde edemedik. Bunun üzerine, şeklin sadece en dışta kalan noktalarına bakabilmek için Şekil 3.8 de akış diagramı görülen yöntemi kullandık.

4.1.1.4 ŞEKİLLERİN ALANLARININ BULUNMASI :

1. bölümde kromozomların sınıflandırma yöntemlerini incelemiştik. Bu çalışmada biz, ilk etapta boya göre sınıflandırma ve sentromer pozisyonuna göre sınıflandırma yöntemlerini kullanmayı uygun bulduk. Tanıma işlemi için kullandığımız resimler Şekil 1.4'deki gibi, kromozomların dağınık bulunduğu resimlerdir. Bu durumda kromozomların boylarına göre sınıflandırma yapabilmek için öncelikle kromozomları eksen sistemine dik olacak şekilde döndürmek gerekecektir. Kolayca tahmin edilebileceği gibi bu oldukça zahmetli bir iştir. Biz bunu yapmak yerine, her kromozomun eninin aynı olduğunu göz önünde bulundurarak, kromozomların alanlarını ayırıcı bilgi olarak kullandık. Bunun için geliştirdiğimiz algoritma Şekil 4.1'deki gibidir.



4.2 ŞEKİL TANIMA

Kromozomları Tanıma İşlemi için, bu çalışmada bir Yapay Nöron Ağı yöntemi olan Adaptif Resonans Teorisinden yararlanarak, öğretileni öğrenebilen ve gerektiğinde eski bilgilerinden yararlanarak kendi kendine öğrenebilen bir sistem geliştirdik.

Başlangıçta, her kromozomu birbirinden ayırt edici özellikler olduğuna göre bu özellikler esas alınarak sınıflandırma işlemi yapılması için sistemi önceden eğitmeye gerek olmadığı düşünülebilir. Fakat kromozom tanıma işleminin genellikle kromozom bozuklukları olan insanlar için kullanılacağı düşünülecek olursa her sınıflandırma sonunda sistem yeni gelen bilgilere göre kendini yenileyeceği için, her kusurlu kromozom için sistem gittikçe doğru verilerden uzaklaşacak ve hatalı sonuçlar çıkarmaya başlayacaktır. Bunu önlemek için başlangıçta sistemi kusursuz kromozomlarla eğitmek gerekir. Bu nedenle bu çalışmada hem öğretileni öğrenebilen hem de kendi kendine öğrenebilen bir sistem tasarlanmıştır.


```
/* BU PROGRAM, GEXILLERE KERNEL UYGULANARAK GORUNTULEMENIN
DUZENLENMESI ICIN KULLANILIR.
```

```
*/
```

```
#include<stdio.h>
#include<math.h>
#include<float.h>
FILE *fo;
FILE *fy;
long pixel[3][256];
double pikyen[3][256];
int kernell[3][3];
```

```
main(argc,argv)
int argc;
char *argv[];
```

```
{
    int i,j,m,n,say,k,l,x,ay,ay2;
    int topl,top2,ker;
    int *point1,*point2;
    int basla;
    char oku;
    int carpan;
    if((fo=fopen(argv[1],"r"))==NULL)
    {
        puts("FILE AÇILAMADI.");
        exit(0);
    }
    if((fy=fopen(argv[2],"w"))==NULL)
    {
        puts("FILE AÇILAMADI.");
        fclose(fo);
        exit(0);
    }
    puts("UYGULANACAK YERLER ?");
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    {
        scanf("%d",&kernell[i][j]);
    }
    puts("ACTIRILAN GORUNTULEME YERLER ?");
    scanf("%d",&say);
    puts("SATAHIN SAĞ YERLER ?");
    scanf("%d",&say2);
    puts("SATIR SAĞ YERLER ?");
    scanf("%d",&say3);
    puts("SUTUN SAĞ YERLER ?");
    scanf("%d",&say4);
    for(i=0;i<say4;i++)
    {
        puts("Görüntüleme Yeri");
        for(j=0;j<3;j++)
        {
            for(l=0;l<say2;l++)
            {
                scanf("%d",&ay);
                pixel[j][ay]=ay;
            }
        }
    }
    for(x=0;x<say3;x++)
    {
        for(l=0;l<say4;l++)
        {
            puts("Görüntüleme Yeri");
            for(i=0;i<say;i++)
            {
                scanf("%d",&ay);
                pikyen[i][ay]=ay;
            }
        }
    }
}
```

Ek-1


```
/* BU PROGRAM, SEKILLERE KERNEL UYGULANARAK GORUNTULERIN  
DUZENLENMESI ICIN KULLANILIR.
```

```
*/
```

```
#include<stdio.h>
```

```
#include<math.h>
```

```
#include<float.h>
```

```
FILE *fo;
```

```
FILE *fy;
```

```
long pixel[3][256];
```

```
double pikyen[3][256];
```

```
int kernell[3][3];
```

```
main(argc,argv)
```

```
int argc;
```

```
char *argv[];
```

```
{
```

```
    int i,j,m,n,say,k,l,z,satsay,sutsay;
```

```
    int topl,top2,ker;
```

```
    int *point1, *point2;
```

```
    int basla;
```

```
    char oku;
```

```
    int carpan;
```

```
    if((fo=fopen(++argv,"rb"))==NULL)
```

```
    {
```

```
        puts("FILE ACILAMADI ");
```

```
        exit(0);
```

```
    }
```

```
    if((fy=fopen(++argv,"wb"))==NULL)
```

```
    {
```

```
        puts("FILE ACILAMADI ");
```

```
        fclose(fo);
```

```
        exit(0);
```

```
    }
```

```
    puts("UYGULANACAK KERNEL'I VERINIZ ");
```

```
    for(i=0;i<3;i++)
```

```
    for(j=0;j<3;j++)
```

```
    {
```

```
        scanf("%d",&ker);
```

```
        kernell[i][j]=ker;
```

```
    }
```

```
    puts("AGIRLIK CARPANINI VERINIZ ");
```

```
    scanf("%d",&carpan);
```

```
    puts("DATANIN BASLANGIC NOKTASINI VERINIZ");
```

```
    scanf("%d",&basla);
```

```
    puts("SATIR SAYISINI VERINIZ");
```

```
    scanf("%d",&satsay);
```

```
    puts("SUTUN SAYISINI VERINIZ ");
```

```
    scanf("%d",&sutsay);
```

```
    for(i=0;i<basla;i++)
```

```
        putc(getc(fo),fy);
```

```
    for(j=0;j<2;j++)
```

```
        for(i=0;i<sutsay;i++)
```

```
        {
```

```
            oku=getc(fo);
```

```
            pixel[j][i]=oku;
```

```
        }
```

```
    for(z=0;z<satsay;z++)
```

```
    {
```

```
        for(i=0;i<sutsay;i++)
```

```
            pixel[2][i]=getc(fo);
```



```

/* BU PROGRAM R for(j=0;j<sutsay;j++) ERIN KOSELERINI
BELTIRGINLEST {
*/
point1=&kernel1[0][0];
top1=0;
#include<stdio.h> for(n=-1;n<=1;n++)
#include<math.h> for(m=-1;m<=1;m++)
#include<float.h> top1+=pixel[1+n][j+m]*(*point1++);
FILE *fo; top1 = top1 / carpan ;
FILE *fy; if (top1>255)
top1=255;
int kernel1[3][3] else if(top1<0)
top1=0;
putc(top1,fy);
}
for(i=1;i<3;i++)
for(j=0;j<sutsay;j++)
int kernel2[3][3]={ pixel[i-1][j]=pixel[i][j];
}
(-1,0,1),
(-2,0,2),
(-1,0,1)
}
long pixel[3][900];

main(argc,argv)
int argc;
char *argv[];
{
int i,j,n,n,say,k,l,satsay,sutsay,y,z;
double top1,top2;
int top;
int *point1, *point2;
int basla;
char oku;

if((fo=fopen(++argv,"rb"))==NULL)
{
puts("FILE ACILAMADI ");
exit(0);
}
if((fy=fopen(++argv,"wb"))==NULL)
{
puts("FILE ACILAMADI ");
fclose(fo);
exit(0);
}

puts("datahin baslangic noktasini veriniz");
scanf("%d",&basla);
printf("%d\n",basla);
puts("satir sayisini veriniz ");
scanf("%d",&satsay);
printf("%d\n",satsay);
puts("sutun sayisini veriniz ");
scanf("%d",&sutsay);
printf("%d\n",sutsay);
for(l=0;l<basla;l++)
putc(getc(fo),fy);
for(j=0;j<2;j++)
for(i=0;i<satsay;i++)
pixel[j][i]=getc(fo);
for(j=0;j<sutsay;j++)

```



```
/* BU PROGRAM RESİM ÜZERİNDEKİ SEKİLLERİN KOSELERİNİ  
BELİRGİNLESTİRMEK İÇİN KULLANILIR .
```

```
*/
```

```
#include<stdio.h>  
#include<math.h>  
#include<float.h>  
FILE *fo;  
FILE *fy;
```

```
int kernel1[3][3]={
```

```
    {-1,-2,-1},  
    {0,0,0},  
    {1,2,1}  
};
```

```
int kernel2[3][3]={
```

```
    {-1,0,1},  
    {-2,0,2},  
    {-1,0,1}  
};
```

```
long pixel[3][900];
```

```
main(argc,argv)
```

```
int argc;
```

```
char *argv[];
```

```
{
```

```
    int i,j,m,n,say,k,l,satsay,sutsay,y,z;  
    double topl,top2;  
    int top;  
    int *point1, *point2;  
    int basla;  
    char oku;
```

```
    if((fo=fopen(++argv,"rb"))==NULL)
```

```
    {  
        puts("FILE ACILAMADI ");  
        exit(0);  
    }
```

```
    if((fy=fopen(++argv,"wb"))==NULL)
```

```
    {  
        puts("FILE ACILAMADI ");  
        fclose(fo);  
        exit(0);  
    }
```

```
    puts("datanın baslangic noktasini veriniz");
```

```
    scanf("%d",&basla);
```

```
    printf("%d\n",basla);
```

```
    puts("satir sayisini veriniz ");
```

```
    scanf("%d",&satsay);
```

```
    printf("%d\n",satsay);
```

```
    puts("sutun sayisini veriniz ");
```

```
    scanf("%d",&sutsay);
```

```
    printf("%d\n",sutsay);
```

```
    for(i=0;i<basla;i++)
```

```
        putc(getc(fo),fy);
```

```
    for(j=0;j<2;j++)
```

```
        for(i=0;i<sutsay;i++)
```

```
            pixel[j][i]=getc(fo);
```

```
    for(j=0;j<sutsay;j++)
```



```

        putc(pixel[0][j],fy);
for(i=1;i<satsay;i++)
{
    for(j=0;j<sutsay;j++)
        pixel[2][j]=getc(fo);
    for(j=0;j<sutsay;j++)
    {
        point1=&kernel1[0][0];
        point2=&kernel2[0][0];
        top1=0;
        top2=0;
        for(n=-1;n<=1;n++)
            for(m=-1;m<=1;m++)
            {
                top1+=pixel[1+n][j+m]*(*point1++);
                top2+=pixel[1+n][j+m]*(*point2++);
            }
        top=sqrt(top1*top1+top2*top2);
        if(top<0)
            top=0;
        else if(top>255)
            top=255;
        putc(top,fy);
    }
}

main(argc,argv)
int argc;
char *argv[];
{
    int n,i,j,sakla;

    if((fo=fopen(++argv,"rb"))==NULL)
    {
        puts("FILE ACILANADI");
        exit(0);
    }

    if((fy=fopen(++argv,"wb"))==NULL)
    {
        puts("FILE ACILANADI");
        fclose(fo);
        exit(0);
    }

    fo=fopen("chain.dat","w");
    oku();
    sutak=sutson-i;
    for(i=0;i<basla;i++)
        putc(getc(fo),fy);

    for(i=0;i<sutson;i++)
        for(j=0;j<sutson;j++)
            pixel[i][j]=getc(fo);

    sat=0;
    sut=0;
    while(sat<sutson || sut<sutson)
        kerar_bul();
    pixel[x][y]=backgr;
    karsu();
    if(bitti==1)
    {
        fprintf(fo,"id id ",x,y);
        for(i=0;i<chain_say;i++)
            fprintf(fc,"id ",chain[i]);
    }
}

```



```

/* BU PROGRAM KOSE BELIRLEME YONTEMI ILE KOSELERI
BELIRGINLESTIRILMIS SEKLIN KONTURUNU BULUR.
*/

```

```

#include<stdio.h>
#include<math.h>
#include<float.h>
FILE *fo;
FILE *fy;
FILE *fc;

```

```

int chain_say=0; /* CHAININ UZUNLUGU */
int chain[100]; /* CHAIN YONLERININ SAKLANDIGI DIZI */
int sinir=245; /* SEKLIN KOSELERINI BELIRLEYEN RENK BEYAZ */
int backgr=25; /* FON RENGİ SIYAH */
int sat,sut,satson,sutson,sutek,basla,x,y;
int pixel[150][170];
int bitti=0;
int bul=0;

```

```

main(argc,argv)
int argc;
char *argv[];
{

```

```

    int m,i,j,sakla;

```

```

    if((fo=fopen(*++argv,"rb"))==NULL)
    {
        puts("FILE ACILAMADI ");
        exit(0);
    }

```

```

    if((fy=fopen(*++argv,"wb"))==NULL)
    {
        puts("FILE ACILAMADI ");
        fclose(fo);
        exit(0);
    }

```

```

    fc=fopen("chain.dat","w");
    oku();
    sutek=sutson-1;
    for(i=0;i<basla;i++)
        putc(getc(fo),fy);

```

```

    for(i=0;i<satson;i++)
        for(j=0;j<sutson;j++)
            pixel[i][j]=getc(fo);

```

```

    sat=0;
    sut=0;
    while(sat<satson && bul==0)

```

```

        kenar_bul();
        pixel[x][y]=backgr;
        komsu();
        if(bitti==1)

```

```

        {
            fprintf (fc,"%d %d ",x,y);
            for(i=0;i<chain_say;i++)
                fprintf(fc,"%d ",chain[i]);

```



```

    }
    for(i=0;i<satson;i++)
        for(j=0;j<sutson;j++)
            putc(pixel[i][j],fy);
}

/* SEKLİN BİR NOKTASINI BULMAK İÇİN KULLANILAN ALT PROGRAM */
kenar_bul()
{
    if(pixel[sat][sut]>=sinir)
    {
        x=sat;
        y=sut;
        bul=1;
    }
    else
    {
        ++sut;
        if(sut==sutek)
        {
            sut=1;
            sat++;
        }
    }
}

/* SEKLİN BİR NOKTASININ YERİ BULUNDUUKTAN SONRA BU
NOKTAYLA AYNI GRI SEVİYESİ DEĞERİNDEKİ KOMSU
NOKTALAR ARANIR. BUNUN İÇİN NOKTANIN 8 KOMSUSUNUN
GRI SEVİYESİ DEĞERLEİNE BAKILIR. EĞER KOMSU
BULUNAMAZSA BU SEKLİN PARAZİT OLDUGU ANLASILIR.
*/

komsu()
{
    while(sat<satson && bitti==0)
    {
        komsu_bul();
    }
}

/* NOKTALARININ KOMSULARININ ARANMASI İŞLEMİNE NOKTANIN
HİCBİR KOMSUSU BULUNAMAYANA KADAR VEYA SEKLE AIT
BULUNAN İLK NOKTANIN KOORDİNATLARINA ULASILANA
KADAR DEVAM EDİLİR.
*/

komsu_bul()
{
    int n;
    int satir,sutun;
    for(n=0;n<8;n++)
    {
        satir=sat+mat[n][0]-1;
        sutun=sut+mat[n][1]-1;
    }
}

```



```

        if(satir==x && sutun==y )
/* BU PROGRAM KESSE BELIRLENEN YONTEMI ILE KOSELERI
BELIRGINLESTIRILIR. bitti=1; 0 SEKILLERIN CEVRE
KOORDINATLARINI BEL return 1; TN KULLANILIR.
*/
    )
    else if(pixel[satir][sutun]>=sinir)

#include<stdio.h>
#include<math.h>
#include<float.h>
FILE *fo;
FILE *fy;
FILE *fo;
    sat=satir;
    sut=sutun;
    pixel[satir][sutun]=backgr;
    chain[chain_say]=n;
    printf("%d \n",chain[chain_say]);
    ++chain_say;
    return 1;
int chain_say=0; /* C HLOSU */
int sinir=245; } SEKILIN KOSELERINI BELIRLEYEN RENK BEYAZ */
int backgr=25; /* FON RENGI SIYAH */
int sat, satson, sutson, sutek, basla, x, y;
int pix; return 0; }
} int bitti=0;
int bul=0;
int ind=0;
/* SEKILLE ILGILI BILGILERIN OKUNDUGU BOLUM */
int mat[4][2]={
oku()
{
    puts ("datanın baslangic noktasini veriniz");
    scanf("%d",&basla);
main(ar puts("satir sayisini veriniz ");
int argv scanf("%d",&satson);
char *a puts("sutun sayisini veriniz ");
scanf("%d",&sutson);
int a,i,j,bakla;
}

if((fo=fopen(++argv,"r"))==NULL)
{
    puts("FILE ACILAMADI ");
    exit(0);
}
if((fy=fopen(++argv,"wb"))==NULL)
{
    puts("FILE ACILAMADI ");
    fclose(fo);
    exit(0);
}
fo=fopen("chain.dat","w");

oku();
sutek=sutson-1;
for(i=0;i<basla;i++)
    puts(getc(fo),fy);
for(i=0;i<satson;i++)
    for(j=0;j<sutson;j++)
        pixel[i][j]=getc(fo);

sat=0;
sut=0;
while(sat<satson && bul==0)
    repeat bul();
if (bul==0)

```



```

/* BU PROGRAM KOSE BELIRLEME YONTEMI ILE KOSELERI
BELIRGINLESTIRILIMIS ICI DOLU SEKILLERIN CEVRE
KOORDINATLARINI BELIRLEMEK ICIN KULLANILIR.
*/

```

```

#include<stdio.h>
#include<math.h>
#include<float.h>
FILE *fo;
FILE *fy;
FILE *fc;

int chain_say=0; /* CHAININ UZUNLUGU */
int sinir=245; /* SEKLIN KOSELERINI BELIRLEYEN RENK BEYAZ */
int backgr=25; /* FON RENGİ SIYAH */
int sat,sut,satson,sutson,sutek,basla,x,y;
int pixel[100][200];
int bitti=0;
int bul=0;
int ind=0;

int mat[4][2]={
    {0,1},{1,0},{0,-1},{-1,0}
};

main(argc,argv)
int argc;
char *argv[];
{
    int m,i,j,sakla;

    /* SEKILIN KOSERINI BELIRLEMEK ICIN KULLANILAN ALT PROGRAM */
    if((fo=fopen(++argv,"rb"))==NULL)
    {
        puts("FILE ACILAMADI ");
        exit(0);
    }
    if((fy=fopen(++argv,"wb"))==NULL)
    {
        puts("FILE ACILAMADI ");
        fclose(fo);
        exit(0);
    }
    fc=fopen("chain.dat","w");
    oku();
    sutek=sutson-1;
    for(i=0;i<basla;i++)
        putc(getc(fo),fy);
    for(i=0;i<satson;i++)
        for(j=0;j<sutson;j++)
        {
            pixel[i][j]=getc(fo);
        }
    sat=0;
    sut=0;
    while(sat<satson && bul==0)
        kenar_bul();
    if (bul==1)
    {

```



```

        komsu();
    }

    else
        puts(" SEKIL BIR PARAZIT ");
    for(i=0;i<satson;i++)
    for(j=0;j<sutson;j++)
        putc(pixel[i][j],fy);
}

/* SEKLIN BIR NOKTASINI BULMAK ICIN KULLANILAN ALT PROGRAM */
kenar_bul()
{
    if(pixel[sat][sut]>=sinir)
    {
        x=sat;
        y=sut;
        bul=1;
    }
    else
    {
        ++sut;
        if(sut==sutek)
        {
            sut=0;
            sat++;
        }
    }
}

/* SEKLIN BIR NOKTASI BULUNDUKTAN SONRA SEKLI TAMAMLAYACAK
SEKILDE BU NOKTANIN KOMSULARIN ARANIR */

komsu()
{
    while(sat<satson && bitti == 0)
        komsu_bul();
}

komsu_bul()
{
    int n,satir,sutun;
    for(n=0;n<4;n++)
    {
        ind = ind % 4;
        satir =sat+mat[ind][0];
        sutun=sut+mat[ind][1];
        if(satir==x && sutun==y )
        {
            bitti=1;
            fprintf(fc,"%d %d ",satir,sutun);
            return 1;
        }
        else if(pixel[satir][sutun]>=sinir)
        {
            fprintf(fc,"%d %d ",satir,sutun);

```



```

/* BU PROGRAM KEVRE KODU BELIRLENNIS SEKILLERIN
SEKIL TANIMA ISLEMINI MAK UZERE ALANLARININ
BULUNMASI ISLEMINDE ONCE KOORDINATLAR SO
*/
sat=satir;
sut=sutun;
pixel[sat][sut]=backgr;
ind = ind + 3;
return 1;
}
#include<stdio.h>
int count[500];
int mat[500][2];
int yen;
return 1;
}
FILE *fp;
/* ILK DEGERLERIN OKUNDUGU BOLUM */
oku()
{
    int yon=1, yonl=0;
    puts("datanın baslangic noktasini veriniz");
    scanf("%d",&basla);
    puts("satir sayisini veriniz ");
    scanf("%d",&satson);
    puts("sutun sayisini veriniz ");
    scanf("%d",&sutson);
    fp=fopen("sorty.dat","w");
    while(!feof(fp))
    {
        fscanf(fp,"%d",&mat[i][0]);
        fscanf(fp,"%d",&mat[i][1]);
        i++;
    }
    n=i;
    sort(n);
    for(i=0;i<n;i++)
    {
        yeni[count[i]][yon]=mat[i][yon];
        yeni[count[i]][yonl]=mat[i][yonl];
    }
    for(i=0;i<n;i++)
    {
        mat[i][yon]=yeni[i][yon];
        mat[i][yonl]=yeni[i][yonl];
    }
    yon=0; yonl=1;
    sort(n);
    for(i=0;i<n;i++)
    {
        yeni[count[i]][yon]=mat[i][yon];
        yeni[count[i]][yonl]=mat[i][yonl];
    }
    for(i=0;i<n;i++)
    {
        mat[i][yon]=yeni[i][yon];
        mat[i][yonl]=yeni[i][yonl];
    }
    for(i=0;i<n;i++)
        fprintf(fp,"%d %d ",mat[i][0],mat[i][1]);
    alan(n);
}

/* LINEER SORT WITH COUNTING ILE SORT */
sort(n)
int son;

```



```

/* BU PROGRAM CEVRE KOORDINATLARI BELIRLENMIS SEKILLERIN
SEKIL TANIMA ISLEMINDE KULLANILMAK UZERE ALANLARININ
BULUNMASI ISLEMINDE KULLANILIR. ALAN ISLEMINE GECMEDEN
ONCE KOORDINATLAR SORT EDILIRLER .

```

```

*/
#include<stdio.h>
int count[600];
int mat[600][2];
int yeni[600][2];

FILE *fp;
FILE *fx;
FILE *fy;
int yon=1, yonl=0;

main()
{
    int i=0,n;
    fp=fopen("chain.dat","r");
    fx=fopen("sortx.dat","w");
    fy=fopen("sorty.dat","w");
    while(!feof(fp))
    {
        fscanf(fp,"%d",&mat[i][0]);
        fscanf(fp,"%d",&mat[i][1]);
        i++;
    }
    n=i;
    sort(n);
    for(i=0;i<n;i++)
    {
        yeni[count[i]][yon]=mat[i][yon];
        yeni[count[i]][yonl]=mat[i][yonl];
    }
    for(i=0;i<n;i++)
    {
        mat[i][yon]=yeni[i][yon];
        mat[i][yonl]=yeni[i][yonl];
    }

    yon=0; yonl=1;
    sort(n);
    for(i=0;i<n;i++)
    {
        yeni[count[i]][yon]=mat[i][yon];
        yeni[count[i]][yonl]=mat[i][yonl];
    }
    for(i=0;i<n;i++)
    {
        mat[i][yon]=yeni[i][yon];
        mat[i][yonl]=yeni[i][yonl];
    }
    for(i=0;i<n;i++)
        fprintf(fx,"%d %d ",mat[i][0],mat[i][1]);
    alan(n);
}

/* LINEER SORT WITH COUNTING ILE SORT */
sort(son)
int son;

```



```

{
    int i,j,n;
    n=son;
    for(i=0;i<n;i++)
        count[i]=0;
    for(i=0;i<n;i++)
        for(j=i+1;j<n;j++)
        {
            if(mat[i][yon]<mat[j][yon])
                count[i]=count[i]+1;
            else
                count[j]=count[j]+1;
        }
}

int d[4],x1[4][2],x2[4][2];
float y[4],x2[4][2],w1[2][2],w2[2];
float toplan,theta,thetay,bata,dw;

/* SEKLIN ALANININ BULUNDUGU BOLUM */
alan(son)
int son;
{
    threesy=3;
    threes[0] int say=0,alan=0;
    threes[1] int i,fark,yer;
    yer=mat[0][yon];
    for(i=1;i<son;i++)
    {
        if (yer==mat[i][yon])
        {
            fark=abs(mat[i][yon1]-mat[i-1][yon1]);
            if (fark == 1)
                ++say;
            else if(say>0)
                say=0;
            else
                alan+=fark;
        }
        else
        {
            yer=mat[i][yon];
            say=0;
        }
    }
    printf("alan=%d\n",alan);
}

for(j=0;j<2;j++)
    x2[i][j]=getnum();
puts("bu degerler icin cixis degeri ne olacaktir");
d[i]=getnum();

/* ORNEGIN CIXIS DEGERLERI VE O AGRI AGILLIK DEGERLERINI
KULLANILARAK GERCEK CIXIS DEGERINI HESAPLAYIR.
*/
do
{
    errors=0;
    for(i=0;i<4;i++)
    {
        for(j=0;j<2;j++)

```



```

/*
BU PROGRAM BACK PROPAGATION YONTEMI ILE BIR TEK
GIZLI SEVIYEDEN IBARET NORON AGLARINI EGITMEK
ICIN KULLANILIR.
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

int exp2(float *, float );
main()
{
    int d[4],x1[4][2],xi2[4][2];
    float y[4],x2[4][2],w1[2][2],w2[2];
    float toplam,theta,thetay,hata,dw;
    int error,i,j,k,l,thresy,thres[2];
    float step=0.4;

    thresy=3;
    thres[0]=-5;
    thres[1]=3;

    /* E°itime baflanmadan önce a°irliklerin de°eri 0 olarak
    alinir. E°itim sirasinda bu a°irlikler do°ru sonuca
    ulafana kadar de°ifeceklerdir.
    */

    for(i=0;i<2;++i)
    {
        w2[i]=0.0;
        for(j=0;j<2;++j)
            w1[i][j]=0.0;
    }

    /* NORON AGINA OGRETILMESI ISTENEN ORNEGIN GIRIS VE CIKIS
    DEGERLERI BELIRLENIR.
    */

    puts("EGITILMESINI ISTEDIGINIZ icin giris degerlerini veriniz .");

    for(i=0;i<4;++i)
    {
        for(j=0;j<2;++j)
            x1[i][j]=getnum();
        puts("bu degerler icin cikis degeri ne olmalı ");
        d[i]=getnum();
    }

    /* ORNEGIN GIRIS DEGERLERI VE O ANKI AGIRLIK DEGERLERI
    KULLANILARAK GERÇEK ÇIKIŞ DEGERLERI HESAPLANIR.
    */

    do
    {
        error=0;
        for(l=0;l<4;++l)
        {
            for(i=0;i<2;++i)
            {
                toplam=0.0;

```



```

        for(j=0;j<2;++j)
            toplam+=w1[j][i]*x1[1][j] ;
        toplam=toplam-thres[i];
        exp2(&x2[1][i],toplam);
        xi2[1][i]=sgn(toplam);
    }

    toplam=0.0;
    for(i=0;i<2;++i)
        toplam+=w2[i]*xi2[1][i];
    toplam=toplam-thresy;
    exp2(&y[1],toplam);
    hata=d[1]-y[1];
    error+=abso(hata);
}

/* GERCEK CIKIS DEGERLERI HESAPLANDIKTAN SONRA
   BU DEGERLER ISTENEN CIKIS DEGERLERI KARSILASTIRILIR
   EGER ISTENEN CIKIS DEGERLERINE ULASILAMAMISSA
   AGIRLIKLAR YENIDEN DUZENLENIR.
*/

if(error!=0)
{
    for(l=0;l<4;++l)
    {
        thetay=y[1]*(1-y[1])*(d[1]-y[1]);
        for(i=0;i<2;++i)
        {
            dw=step*thetay*xi2[1][i];
            w2[i]+=dw;
        }
        for(j=0;j<2;++j)
        {
            theta=x2[1][j]*(1-x2[1][j])*w2[j]*thetay;
            for(i=0;i<2;++i)
            {
                dw=step*thetay*x1[1][i];
                w1[i][j]+=dw;
            }
        }
    }
}

while(error!=0);

/* GERCEK CIKIS DEGERLERI ISTENEN CIKIS DEGERLERINE
   YAKLASANA KADAR NORON AGININ EGITIMINE DEVAM EDILIR.
*/

puts(" Agirlik Fonksiyonlari ");
printf("w1[0][0] = %f   w1[1][0] = %f \n",w1[0][0],w1[1][0]);
printf("w1[0][1] = %f   w1[1][1] = %f \n",w1[0][1],w1[1][1]);
puts("Cikis Degerleri ");
for(i=0;i<4;++i)
    printf("y[%d]=%f\n",i,y[i]);
}

getnum()
{
    char s[80];

```



```
gets(s);  
return(atoi(s));  
}
```

```
void exp2(y,toplam)  
float *y;  
float toplam;  
{  
    toplam=-toplam;  
    *y=1./(1.+(float)exp((double)toplam));  
}
```

```
abso(x)  
float x;  
{  
    if(x<0.0)  
        x=-x;  
    if(x<0.2)  
        return(0);  
    return(1);  
}
```

```
sgn(x)  
float x;  
{  
    if(x>0.0)  
        return(1);  
    return(0);  
}
```

```
/* TANITILACAK ORNEGIN GIBI BULANIKLAŞTIRILAN ORNEKLERİN GİRİŞİ */
```

```
ornek()  
{  
    int i,j;  
    float ornek;  
    scanf("%d",&orsay);  
    scanf("%d",&girsay);  
    printf("ornek=%d girsay=%d",orsay,girsay);  
    fopen("ornek.txt","w");  
    for(i=0;i<orsay;i++)
```



```

/* BU PROGRAM ADAPTIF REZONANS TEORI 2 YONTEMI ILE SEKIL
TANIYAN BIR PROGRAMDIR. TANINMASI ISTENEN SEKIL O ANA
KADAR MEVCUT SINIFLARDAN BIRINE DAHILSE O SINIFIN
AGIRLIKLARI YENIDEN DUZENLENIR. EGER HICBIR SINIFA
DAHIL DEGILSE, BU SEKIL OGRENILIR, YANI YENI BIR
SINIFA ALINIR.
*/

#include<stdio.h>
#include<float.h>
#include <math.h>
FILE *fo;
int akt_nod,orsay,girsay,class;
float threshold;
int cluster[20][20];
float bot[20][20],ornek[20][20],euclidean[20],x[20];
main()
{
    int i,j,say;
    scanf("%f",&threshold);
    oku();
    initial();
    for(i=0;i<orsay;i++)
    {
        for(j=0;j<girsay;j++)
            x[j]=ornek[i][j];
        hesap_ed();
        if(kontrol_ed())
            update_we(i);
        else
            sınıf_ac(i);
    }
    for(i=0;i<akt_nod;i++)
    {
        say=cluster[i][0];
        for(j=0;j<say;j++)
            printf("cluster[%d][%d]=%d\n",i,j,cluster[i][j]);
    }
    for(i=0;i<akt_nod;i++)
        for(j=0;j<girsay;j++)
            printf("bot[%d][%d]=%f\n",i,j,bot[i][j]);
}

/* TANITILACAK ORNEGIN GIRIS BILGILERININ OKUNDUGU BOLUM */
oku()
{
    int i,j;
    float orn;
    scanf("%d",&orsay);
    scanf("%d",&girsay);
    printf("ornek=%d giris=%d \n",orsay,girsay);
    fo=fopen("ornek.dat","r");
    for(i=0;i<orsay;i++)
    {
        for(j=0;j<girsay;j++)

```



```

        fscanf(fo,"%f",&ornek[i][j]);
        fscanf(fo,"\n");
    }
} /* EGER ORNEK BELIRLENEN SINIFA AITSE BU SINIFTN
   BOTTOM_UP AGIRLIKLARIN UPDATE EDILIR
*/

/* BU BOLUMDE BOTTOM_UP AGIRLIKLARIN ILK DEGERLERI
   BELIRLENIR .
*/
initial()
{
    int i,j;
    for(i=0;i<girsay;i++)
        bot[0][i]=ornek[0][i];
    akt_nod=1;
    cluster[0][0]=1; /* Sadece bir ornek var */
    cluster[0][1]=0; /* Sifirinci ornek ilk sinifa yerlestirildi
}

/* YENI GELEN ORNEGI TANIMAK ICIN ORNEGIN MEVCUT SINIFLARLA
   EUCLIDE DISTANCE HESABI YAPILIR. ORNEGIN MESAFESININ
   EN AZ OLDUGU SINIF BELIRLENIR.
*/

hesap_ed()
{
    int i,j;
    float cikar;
    float min;
    for(i=0;i<akt_nod;i++)
    {
        euclide[i]=0.0 ;
        for(j=0;j<girsay;j++)
        {
            cikar=bot[i][j]-x[j];
            euclide[i]=euclide[i]+cikar*cikar;
        }
    }
    min=euclide[0];
    class=0;
    for(i=0;i<akt_nod;i++)
        if(euclide[i]<min)
        {
            min=euclide[i];
            class=i;
        }
}

/* ORNEGIN, GERCEKTEN EUCLIDE DISTANCE'IN MINIMUM OLDUGU
   SINIFA AIT OLUP OLMADIGI KONTROL EDILIR .
*/

kontrol_ed()
{
    float hes;
    hes=sqrt(euclide[class]*1.0);
    if(hes<=threshold)
        return 1;
    return 0;
}

```



```
}
```

```
/* EGER ORNEK BELIRLENEN SINIFA AITSE BU SINIFIN  
BOTTOM_UP AGIRLIKLARIN UPDATE EDILIR  
*/
```

```
update_we(inp)  
int inp;  
{  
    int n,i,eleman;  
    float m,k;  
    n=cluster[class][0];  
    m=n+1.0;  
    k=1.0/(m*1.0);  
    for(i=0;i<akt_nod;i++)  
    {  
        bot[class][i]=n*k*bot[class][i]+k*x[i];  
        eleman=cluster[class][0]+1;  
        cluster[class][0]++;  
        cluster[class][eleman]=inp;  
    }  
}
```

```
/* ORNEK HICBIR SINIFA DAHIL DEGILSE BU ORNEK ICIN YENI  
BIR SINIF ACILIR.  
*/
```

```
sinif_ac(inp)  
int inp;  
{  
    int i;  
    cluster[akt_nod][0]=1;  
    cluster[akt_nod][1]=inp;  
    for(i=0;i<girsay;i++)  
        bot[akt_nod][i]=ornek[inp][i];  
    akt_nod++;  
}
```


BU PROGRAMDAN SCANNER'DAN ALINAN RESIMLERIN SAKLANDIGI
TIF FILE'LARIN FORMATINI COZMEK ICIN KULLANILIR.

```

/*
#include <fcntl.h>

/* TIFF DATA TIPLERI */

#define INTELSTR      "II"
#define INTEL         0x4949
#define MOTOROLA      0x4d4d
#define BYTE          char
#define ASCII         char
#define SHORT          int
#define LONG           long

typedef struct RAT {
    long upper;
    long lower;
} RATIONAL;

struct tdirant {
    short tagno;
    short ttype;
    long lcount;
    long loffset;
};

#define BYTE_SIZE      sizeof(BYTE)
#define ASCII_SIZE     sizeof(ASCII)
#define SHORT_SIZE     sizeof(SHORT)
#define LONG_SIZE      sizeof(LONG)
#define RATIONAL_SIZE  sizeof(RATIONAL)

#define BYTE_TYPE      1
#define ASCII_TYPE     2
#define SHORT_TYPE     3
#define LONG_TYPE      4
#define RATIONAL_TYPE  5

#define ERROR_VALUE    0
#define DIRECT_VALUE   1
#define INDIRECT_VALUE 2

#define SUBFILE_TYPE_TAG 0x0001
#define IMAGE_WIDTH_TAG  0x0100
#define IMAGE_LENGTH_TAG 0x0101
#define BITS_PER_SAMPLE_TAG 0x0102
#define COMPRESSION_TAG  0x0103

#define PHOTOMETRIC_INTERP_TAG 0x0106
#define THRESHOLDING_TAG 0x0107
#define CELL_WIDTH_TAG 0x0108
#define CELL_LENGTH_TAG 0x0109
#define FILL_ORDER_TAG 0x010a

#define DOCUMENT_NAME_TAG 0x010d
#define IMAGE_DESCRIPTION_TAG 0x010e
#define MAKE_TAG 0x010f

```

Ek-2


```

/*
BU PROGRAMDAN SCANNER'DAN ALINAN RESIMLERIN SAKLANDIGI
TIF FILE'LARIN FORMATINI COZMEK ICIN KULLANILIR.
*/

#include <fcntl.h>

/* TIFF DATA TIPLERI */

#define INTELSTR      "II"
#define INTEL         0x4949
#define MOTOROLA      0x4d4d
#define BYTE          char
#define ASCII          char
#define SHORT          int
#define LONG           long

typedef struct RAT {
    long upper;
    long lower;
    } RATIONAL;

struct tdirent {
    short tagno;
    short ttype;
    long lcount;
    long foffset;
};

#define BYTE_SIZE      sizeof(BYTE)
#define ASCII_SIZE     sizeof(ASCII)
#define SHORT_SIZE     sizeof(SHORT)
#define LONG_SIZE      sizeof(LONG)
#define RATIONAL_SIZE  sizeof(RATIONAL)

#define BYTE_TYPE      1
#define ASCII_TYPE     2
#define SHORT_TYPE     3
#define LONG_TYPE      4
#define RATIONAL_TYPE  5

#define ERROR_VALUE    0
#define DIRECT_VALUE   1
#define INDIRECT_VALUE 2

#define SUBFILE_TYPE_TAG 0x00ff
#define IMAGE_WIDTH_TAG 0x0100
#define IMAGE_LENGTH_TAG 0x0101
#define BITS_PER_SAMPLE_TAG 0x0102
#define COMPRESSION_TAG 0x0103
#define PHOTOMETRIC_INTERP_TAG 0x0106
#define THRESHOLDING_TAG 0x0107
#define CELL_WIDTH_TAG 0x0108
#define CELL_LENGTH_TAG 0x0109
#define FILL_ORDER_TAG 0x010a
#define DOCUMENT_NAME_TAG 0x010d
#define IMAGE_DESCRIPTION_TAG 0x010e
#define MAKE_TAG 0x010f

```



```

#define MODEL_TAG 0x0110
#define STRIP_OFFSETS_TAG 0x0111
#define ORIENTATION_TAG 0x0112
#define SAMPLES_PER_PIXEL_TAG 0x0115
#define ROWS_PER_STRIP_TAG 0x0116
#define STRIP_BYTE_COUNTS_TAG 0x0117
#define MIN_SAMPLE_VALUE_TAG 0x0118
#define MAX_SAMPLE_VALUE_TAG 0x0119
#define X_RESOLUTION_TAG 0x011a
#define Y_RESOLUTION_TAG 0x011b
ne PLANAR_CONFIG_TAG 0x011c
#define PAGE_NAME_TAG 0x011d
#define X_POSITION_TAG 0x011e
#define Y_POSITION_TAG 0x011f
#define FREE_OFFSETS_TAG 0x0120
#define FREE_BYTE_COUNTS_TAG 0x0121
#define GRAY_RESP_UNIT_TAG 0x0122
#define GRAY_RESP_CRVE_TAG 0x0123
#define GROUP_3_OPT_TAG 0x0124
#define GROUP_4_OPT_TAG 0x0125
#define RESOLUTION_TAG 0x0128
#define PAGE_NO_TAG 0x0129
#define COLOR_RSP_TAG 0x012C
#define COLOR_RSP_CRV_TAG 0x012D

#define PACK_TIGHTLY 1
#define ONE_D_MOD_HUFFMAN 2
#define TWO_D_CCITT 3

#define LEGAL_VERSION 42
#define SIZEOF_DIR_ENT 12

char *tag_expl[] = {
"SUBFILE_TYPE_TAG", "IMAGE_WIDTH_TAG",
"IMAGE_LENGTH_TAG", "BITS_PER_SAMPLE_TAG",
"COMPRESSION_TAG", "\"\", \"\", \"PHOTOMETRIC_INTERP_TAG\",
"THRESHOLDING_TAG", "CELL_WIDTH_TAG",
"CELL_LENGTH_TAG", "FILL_ORDER_TAG", "\"\", \"\", \"\",
"DOCUMENT_NAME_TAG", "IMAGE_DESCRIPTION_TAG",
"MAKE_TAG", "MODEL_TAG",
"STRIP_OFFSETS_TAG", "ORIENTATION_TAG", "\"\", \"\", \"\",
"SAMPLES_PER_PIXEL_TAG", "ROWS_PER_STRIP_TAG",
"STRIP_BYTE_COUNTS_TAG", "MIN_SAMPLE_VALUE_TAG",
"MAX_SAMPLE_VALUE_TAG", "X_RESOLUTION_TAG",
"Y_RESOLUTION_TAG", "PLANAR_CONFIG_TAG",
"PAGE_NAME_TAG", "X_POSITION_TAG",
"Y_POSITION_TAG", "FREE_OFFSETS_TAG",
"FREE_BYTE_COUNTS_TAG", "GREY_RESPONSE_UNIT",
"GREY_RESPONSE_CURVE", "GROUP_3_OPTIONS",
"GROUP_4_OPTIONS_TAG", "\"\", \"\", \"RESOLUTION_UNIT\",
"PAGE_NUMBER_TAG", "\"\", \"\", \"COLOR_RESP_UNIT\",
"COLOR_RESP_CRVE_TAG"};

#define OPTION '-'
#define OPT_VERBOSE 'v'
#define LOWER 0x20

char creator;
long ifdptr;
int vflag;

```



```

char ifile[80];
int main(argc,argv)
int argc;
char *argv[];
{
    int fd;          /* file descriptor */

    if(valid_parm(argc,argv)) {
        printf("usage: %s [-v] infile.tif\n",argv[0]);
        exit(1);
    }

    if((fd = open_tif_file(ifile)) == -1) {
        printf("FILE ACILAMADI \n");
        exit(2);
    }

    if(!read_tif_header(fd)) {
        exit(3);
    }

    get_field_data(fd);

    valid_parm(argc,argv)
    int argc;
    char *argv[];
    {
        int i;
        #ifdef DEBUG
            for(i=0;i<argc;++i) {
                printf("argv[%d] = '%s'\n",i,argv[i]);
            }
        #endif

        if(argc < 2 || argc > 3) {
            return(1);
        }

        for(i=1;i<argc;++i){
            if(argv[i][0] == OPTION) {
                switch(argv[i][1]|LOWER) {

                    case OPT_VERBOSE: /* found a 'v' */
                        vflag = 1;
                        break;

                    default:
                        /* found an illegal parameter */
                        return(3);
                        break;

                }
            } else {
                strcpy(ifile,argv[i]);
            }
        }

        #ifdef DEBUG

```



```

        printf("leaving valid_parm, vflag = %d, ifile = '%s'\n",vflag
#endif
        return(0);
    }

int open_tif_file(infilename)
char *infilename;
{
    return(open(infilename,O_RDONLY|O_BINARY));
}

int read_tif_header(filedes)
int filedes;
{
    int retval;
    struct tif_head {
        char creat[2];
        int version;
        long ifd;
    } head_data;

    if((retval = read(filedes,(char *)&head_data,sizeof(struct ti
count != sizeof(struct tif_head)) {
    return (-1);
}

    if(!strcmp(head_data.creat,INTELSTR,2)){
        printf("File was created on an Intel Processor\n");
        creator = 'I';
    } else {
        printf("File was created on an Motorola Processor\n")
        creator = 'M';
    }

    if(creator == 'M')
        swapit(&head_data.version,SHORT_TYPE);

    printf("Tiff spec. used to create format of file = %d\n",
        head_data.version);

    if(creator=='M'){
        swapit(&head_data.ifd,LONG_TYPE);
    }
    printf("Offset of 1st IFD = %ld\n",head_data.ifd);
    ifdptr = head_data.ifd;

    d_data(filedes)
    ds;

    short count;
    long retval;
    int bytes_to_read;
    char *ifd;
    char *sifd;

    struct tdirent dirent;

```



```

    retval = lseek(fileds, ifdptr, 0);
    if(read(fileds, &count, sizeof(short)) != sizeof(short)) {
        printf("Cannot read tiff file, ifd directory entry co
        exit(1);
    }

    if(creator == 'M')
        swapit(&count, SHORT_TYPE);

    bytes_to_read = count * SIZEOF_DIR_ENT;

    ifd = (char *)malloc(bytes_to_read);

    if(read(fileds, ifd, bytes_to_read) != bytes_to_read) {
        printf("Cannot read tiff file, ifd directory elements
        exit(2);
    }

    printf("number of entries in the first ifd = %d\n", count);

    sifd = ifd;

    while(count != 0) {
        memcpy(&dirent, ifd, (unsigned)SIZEOF_DIR_ENT);
        ifd += SIZEOF_DIR_ENT;
        breakup(fileds, &dirent);
        count--;
    }
    free(sifd);
}

breakup(fd, ptr)
int fd;
struct tdirent *ptr;
{
    long i;
    union values {
        char byte[512];
        char ascii[512];
        short word[256];
        long longw[128];
        RATIONAL ratio[64];
    } *buffer;
    long bytes_to_read;

    if(creator == 'M') {
        swapit(&ptr->tagno, SHORT_TYPE);
        swapit(&ptr->ttype, SHORT_TYPE);
        swapit(&ptr->lcount, LONG_TYPE);
        if(ptr->ttype == SHORT_TYPE && ptr->lcount == 1) {
            swapit(&ptr->foffset, SHORT_TYPE);
        } else {
            swapit(&ptr->foffset, LONG_TYPE);
        }
    }

    printf("\ntag no. = %d ", ptr->tagno);

    if(ptr->tagno - 255 >= 0) {
        printf("%s\n", tag_expl[ptr->tagno - 255]);
    }
}

```



```

else {
    printf("Private Tag\n");
}

printf("\tttype = %d", ptr->ttype);
switch(ptr->ttype) {
    case BYTE_TYPE:
        printf(" type of TAG is BYTE\n");
        break;
    case ASCII_TYPE:
        printf(" type of TAG is ASCII\n");
        break;
    case SHORT_TYPE:
        printf(" type of TAG is SHORT\n");
        break;
    case LONG_TYPE:
        printf(" type of TAG is LONG\n");
        break;
    case RATIONAL_TYPE:
        printf(" type of TAG is RATIONAL\n");
        break;
}
printf("\tlcount = %ld\n", ptr->lcount);
printf("\tfoffset = %ld\n", ptr->foffset);

if( ptr->lcount > 1 || ptr->ttype > 4 ) {
    printf("values of tag at offset:\n\t");
    switch(ptr->ttype) {
        case BYTE_TYPE:
        case ASCII_TYPE:
            bytes_to_read = BYTE_SIZE;
            break;
        case SHORT_TYPE:
            bytes_to_read = SHORT_SIZE;
            break;
        case LONG_TYPE:
            bytes_to_read = LONG_SIZE;
            break;
        case RATIONAL_TYPE:
            bytes_to_read = RATIONAL_SIZE;
            break;
    }
    bytes_to_read *= ptr->lcount;

    if(bytes_to_read > 4 ) {
        lseek(fd, ptr->foffset, 0);

        buffer = (union values *)malloc(bytes_to_read);
        if(read(fd, buffer, bytes_to_read) != bytes_to_read) {
            printf("invalid tiff file, cannot read at fof\n");
            close(fd);
            free(buffer);
            exit(5);
        }
    }
    else
    {
        memcpy(buffer, &ptr->foffset, bytes_to_read);
    }
    /* now based on type, break up the read in data area */
    for(i=0; i<ptr->lcount; i++) {
        switch(ptr->ttype) {

```



```

case BYTE_TYPE:
    printf("%x ",buffer->byte[i]);
    break;
case ASCII_TYPE:
    printf("%c",buffer->byte[i]);
    break;
case SHORT_TYPE:
    if(creator == 'M'){
        swapit(&buffer->word[i],
            SHORT_TYPE);
    }
    printf("%d ",buffer->word[i]);
    break;
case LONG_TYPE:
    if(creator == 'M') {
        swapit(&buffer->longw[i],
            LONG_TYPE);
    }
    printf("%ld ",buffer->longw[i]);
    break;
case RATIONAL_TYPE:
    if(creator == 'M') {
        swapit(&buffer->ratio[i].uppe
            LONG_TYPE);
        swapit(&buffer->ratio[i].lowe
            LONG_TYPE);
    }
    printf("%ld / %ld ",
        buffer->ratio[i].upper,
        buffer->ratio[i].lower);
    break;

```

```

    }
    printf("\n");
    free(buffer);
}

```

```

:(ptr,type)
ptr;
pe;

```

```

char temp;

```

```

if(type == SHORT_TYPE) {
    temp = *ptr;
    *ptr = *(ptr+1);
    *(ptr+1) = temp;
}
else if(type == LONG_TYPE) {
    temp = *(ptr+3);
    *(ptr+3) = *ptr;
    *ptr = temp;
    temp = *(ptr+2);
    *(ptr+2) = *(ptr+1);
    *(ptr+1) = temp;
}

```


FileOrder

Tag = 255 (FF)

Type = SHORT

N = 1

SubfileType

Tag = 255 (FF)

Type = SHORT

N = 1

ImageWidth

Tag = 256 (100)

Type = SHORT

N = 1

ImageLength

Tag = 257 (101)

Type = SHORT

N = 1

BitsPerSample

Tag = 258 (102)

Type = SHORT

N = SamplesPerPixel

Compression

Tag = 259 (103)

Type = SHORT

N = SamplesPerPixel for PlanarConfiguration equal to 1 or 2.

PhotometricInterpretation

Tag = 262 (106)

Type = SHORT

N = 1

Thresholding

Tag = 263 (107)

Type = SHORT

N = 1

CellWidth

Tag = 264 (108)

Type = SHORT

N = 1

CellLength

Tag = 265 (109)

Type = SHORT

N = 1

FillOrder

Tag = 266 (10A)

Type = SHORT

N = 1

DocumentName

Tag = 269 (10D)

Type = ASCII

ImageDescription

Tag = 270 (10E)

Type = ASCII

Make

Tag = 271 (10F)

Type = ASCII

Model

Tag = 272 (110)

Type = ASCII

StripOffsets

Tag = 273 (111)

Type = SHORT or LONG

N = StripsPerImage for PlanarConfiguration equal to 1.

= SamplesPerPixel * StripsPerImage for PlanarConfiguration equal to 2

Orientation

Tag = 274 (112)

Type = SHORT

N = 1

SamplesPerPixel

Tag = 277 (115)

Type = SHORT

N = 1

RowsPerStrip

Tag = 278 (116)

Type = SHORT or LONG

N = 1

StripByteCounts

Tag = 279 (117)

Type = LONG

N = StripsPerImage for PlanarConfiguration equal to 1.

= SamplesPerPixel * StripsPerImage for PlanarConfiguration equal to 2

MinSampleValue

Tag = 280 (118)

Type = SHORT

N = SamplesPerPixel

MaxSampleValue

Tag = 281 (119)

Type = SHORT

N = SamplesPerPixel

XResolution

Tag = 282 (11A)

Type = RATIONAL

N = 1

YResolution

Tag = 283 (11B)

Type = RATIONAL

N = 1

PlanarConfiguration

Tag = 284 (11C)

Type = SHORT

N = 1

PageName

Tag = 285 (11D)

Type = ASCII

XPosition

Tag = 286 (11E)

Type = RATIONAL

YPosition

Tag = 287 (11F)

Type = RATIONAL

FreeOffsets

Tag = 288 (120)

Type = LONG

FreeByteCounts

Tag = 289 (121)

Type = LONG

GrayResponseUnit

Tag = 290 (122)

Type = SHORT

N = 1

GrayResponseCurve

Tag = 291 (123)

Type = SHORT

N = $2^{**} \text{BitsPerSample}$

Group3Options

Tag = 292 (124)

Type = LONG

N = 1

Group4Options

Tag = 293 (125)

Type = LONG

N = 1

ResolutionUnit

Tag = 296 (128)

Type = SHORT

N = 1

PageNumber

Tag = 297 (129)

Type = SHORT

N = 2

ColorResponseUnit

Tag = 300 (12C)

Type = SHORT

N = 1

ColorResponseCurves

Tag = 301 (12D)

Type = SHORT

N = $2^{**} \text{BitsPerSample (for Red sample)} +$
 $2^{**} \text{BitsPerSample (for Green sample)} +$
 $2^{**} \text{BitsPerSample (for Blue sample)}$

Parallel Distributed
Processing Vol. 1
The MIT Press, 1988

Explorations in
Parallel Distributed
Processing
The MIT Press, 1988

Adaptive Pattern
Recognition and
Neural Networks
Addison Wesley, 1989

An Introduction to
Computing with
Neural Networks
IEEE ASSP, April 1987

Neural Networks for
Visual Pattern Recognition
IEEE Comp., March 1988

Neural Networks
IEEE Expert, April 1989

Neural Networks Systems
Computer Design, 1988

Back-Propagation
BYTE, October 1987

YARALANILAN KAYNAKLAR

1. Rumelhart D.- Mc Clelland J. Parallel Distributed Processing Vol.1
The MIT Press, 1988
2. Rumelhart D.- Mc Clelland J. Explorations in Parallel Distributed Processing
The MIT Press, 1988
3. Pao Yoh-Han Adaptive Pattern Recognition and Neural Networks
Addison Wesley P., 1989
4. Lippmann Richard An Introduction to Computing with Neural Networks
IEEE ASSP, April 1987
5. Fukushima K. Neural Network for Visual Pattern Recog.
IEEE Comp., March 1988
6. Wasserman P.-Schwartz T. Neural Networks
IEEE Expert, April 1988
7. Buffa M.G. Neural Networks Systems Computer Design, 1988
8. Jones W.P.-Hoskins J. Back-Propogation
BYTE, October 1987

ÖZGEÇMİŞ

Elif Karslıgil 1968 yılında İstanbul'da doğdu. Lisesi Kadıköy Kız Lisesinde bitirdi. 1988 yılında Yıldız Üniversitesi Bilgisayar Bilimleri Mühendisliği Bölümünden birincilikle mezun oldu ve aynı yıl Yıldız Üniversitesi Fen Bilimleri Enstitüsünde Bilgisayar Bilimleri Anabilim Dalında yüksek lisans eğitimine başladı. 1988 yılından beri aynı üniversitenin Bilgisayar Bilimleri Mühendisliği Bölümünde araştırma görevlisidir.



