Geriyansımalı Yapay Sinir Ağının Görüntü Sıkıştırmada Kullanılması

Mustafa Tosun

YÜKSEK LİSANS TEZİ
Dumlupınar Üniversitesi, Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı
Ağustos -**1997**

GERİYANSIMALI YAPAY SİNİR AĞININ GÖRÜNTÜ SIKIŞTIRMADA KULLANILMASI

MUSTAFA TOSUN

Dumlupınar Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik-Elektronik Mühendisliği Anabilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır.

Danışman: Yrd. Doç. Dr. Mehmet Ali Ebeoğlu

Ağustos-1997

Mustafa TOSUN'un YÜKSEK LİSANS tezi olarak hazırladığı "GERİYANSIMALI YAPAY SİNİR AĞININ GÖRÜNTÜ SIKIŞTIRMADA KULLANILMASI " başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

1./10./1997

Üye : Yrd. Doç. Dr. M. Ali EGEOĞLU

Üye : Dr. Yılmaz ASLAN

Üye: Yrd. Doç. Dr. Kaan ERARSLAN

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 23./10./1997 gün ve ...14.....sayılı. kararıyla onaylanmıştır.

Doç. Dr. İ. Gökhay EDİZ

Fen Bilimleri Enstitüsü Müdürü

GERİYANSIMALI YAPAY SİNİR AĞININ
GÖRÜNTÜ SIKIŞTIRMADA KULLANILMASI

Mustafa Tosun
Elektrik-Elektronik Mühendisliği Yüksek Lisans Tezi, 1997
Tez danışmanı : Yrd. Doç. Dr. M. Ali Ebeoğlu

ÖZET

Bilgi çağında iletişim teknolojileri hızla gelişmiştir. Bu durum birey ve toplumlar arasındaki haberleşmeyi artırmıştır. Böylece bilgi iletim kanallarının verimli kullanılmasını zorunlu hale gelmiştir. Bu zorunluluk sıkıştırma tekniklerine yönelimi artırmıştır.

Bu çalışmada kayıplı ve kayıpsız bilgi sıkıştırma yöntemleri ile ilgili temel bilgiler verilmiştir. Minimum fazlalık kodlama, aritmetik kodlama ve run length kodlama yöntemleri incelenmiştir. Ayrıca görüntü sıkıştırma kodlayıcılarından vektör gruplama yöntemi üzerinde durulmuştur. Son yıllarda yaygın olarak çalışılan, yapay sinir ağları ile görüntü sıkıştırma yöntemleri üzerinde de durulmuştur. Geriyansımalı yapay sinir ağı ile görüntü sıkıştırma konusunda detaylı bilgiler verilmiş ve algoritması gerçekleştirilmiştir. Bu algoritma ile 4:1 oranında sıkıştırma, %3 hata ile başarılmıştır.

Anahtar Kelimeler: Görüntü sıkıştırma, sinyal işleme, vektör gruplama, yapay sinir ağları, geri yansıma algoritması.

IMAGE COMPRESSION BY BACKPROPAGATION ALGORITHM

Mustafa Tosun

Electric-Electronic Engineering Ms. thesis,1997

Supervisor: Yrd. Doc. Dr. M. Ali Ebeoğlu

SUMMARY

Communication technologies are developed in information age. This is raised up the communicating between human and societies. Therefore, it is necessary using the communication channel succesfuly. Compression techniques are real solutions of this problem

In This study, basic information about lossy and lossless data compression techniques has been given. Minimum redundancy coding, arithmetic coding and run length coding techniques have been analysed. Vector quantisation technique has been considered. Detailed information has been given about image compression technique by backpropagation neural network and realised the algorithm of this study. This algorithm has been succeeded by 4:1 compression ratio with 3% error.

Keywords: Image compression, Signal processing, vector quantisation, neural network, backpropagation algorithm.

TEŞEKKÜR

" Geriyansımalı yapay sinir ağının görüntü sıkıştırmada kullanılması" konulu bu çalışma, Yrd. Doc. Dr. M. Ali EBEOĞLU'nun gözetiminde gerçekleştirilmiştir. Çalışmamın sonuçlanmasında her türlü destek ve yardımı esirgemeyen sayın Yrd. Doç. Dr. M. Ali. EBEOĞLU' na teşekkürlerimi sunarım.

Yüksek Lisans tez çalışmam boyunca her türlü fedakarlıktan kaçınmayarak, görüntü sıkıştırma konusu ile ilgili çalışmalarından yararlanmamı sağlayan, TÜBİTAK-MAM Bilişim teknolojileri bölümünden sayın Selim ÇETİN'e teşekkür ederim.

Ayrıca Dumlupınar üniversitesi Mühendislik Fakültesi araştırma görevlilerinden Hasan Temurtaş, Mehmet Bulut ve H. Melih Saraoğlu'na katkılarından dolayı teşekkür ederim.

İÇİNDEKİLER

İÇİNDEKİLER (devam)

ŞEKİLLER DİZİNİ

ŞEKİLLER DİZİNİ (devamı)

ÇİZELGELER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
B	Parlaklık
ΔB	Parlaklık farkı
B_o	Çevre aydınlığı
X	Giriş değerleri
X_c	Sıkıştırılmış değerler
Y	Geri elde edilen değerler
P	Olasılık
L	Olasılıktaki ortalama uzunluk
$n(a_i)$	a_i karakteri için kod kelimedeki bit sayısı
X( )	Rastgele değişken
F_x( )	Kümülatif yoğunluk fonksiyonu
T_x ( )	Etiket değer fonksiyonu
S_c	Özel karakter işaretleyici
C_c	Karakter sayıcı
$İ_n$	Kod kelime endeksi
S_n	Durum kod kitabı
W	Ağırlık
θ	Eğilimleme terimi
$İ_p$	Birim aktivasyonu
μ	Pozitif sabit
ε_k	k'ıncı çıkışın hatası
δ	Hata tanım ifadesi
E_p	Hataların karesel toplamı
η	Öğrenme parametresi
O_p	Çıkış katman çıkışı
Y_p	Çıkış katman hedef çıkışı
α	Momentum parametresi
M_t	Sıkıştırma oranı
n_h	Ara katman sayısı
u_h	Bir ara katman birimi çıkışının uzunluğu (byte)

n_v	Vektör sayısı
di	Görüntü boyutu
w_n	Ara katman ağırlık ağırlık sayısı
u_i	Görüntü piksel uzunluğu (byte)
u_w	Bir ağırlık değerinin uzunluğu (byte)

Kısaltmalar

H.D.T.V	High Definition Television
J.N.D	Just Noticable Difference
D.C.T	Discrete Cosine Transform
D.P.C.M	Delta Pulse Cod Modulation
P.C.M	Pulse Cod Modulation
D.M	Delta Modulation
V.Q	Vector Quantization
L.B.G	Linda Buzo Gray
D.F.T	Discrete Fourier Transform
D.S.T	Discrete Sine Transform
S.O.F.M	Self Organizing Feature Map
S.N.R	Signal Noise Ratio
A.P.V.Q	Adress Predictive Vector Quantization
N.N	Neural Network
B.P.N	Back Propagation Network

1. GİRİŞ

Bilgi çağında, bilginin depolanması ve iletilmesi önemli problemler arasında yer almaktadır. Bunun nedeni, haberleşmede kullanılan bakır ve fiberoptik kabloların iletim bant genişliklerinin sınırlı olmasıdır. Bu nedenle, bir kanal üzerindeki bilginin güvenilir olarak taşınabildiği maximum oran olarak tanımlanan kanal kapasitesinin verimli kullanılması gerekir. Kanalın sınırlı bant genişliğini verimli kullanabilmek için iletilen bilgideki fazlalığın atılması gerekir. bilgi içerisinde bulunan ve insan gözünün farkedemediği bazı bilgiler, bilgi içerisindeki fazlalıklar olarak kabul edilebilir. Görüntü bilgisindeki fazlalıkların çıkartılarak, bilgilerin daha az uzunluklarda işaretlenmesi "**sıkıştırma**" olarak tanımlanır. Bilgilerin sıkıştırılmasında kullanılan çeşitli yöntemler vardır. Bu yöntemler "**kayıplı sıkıştırma**" ve "**kayıpsız sıkıştırma**" olmak üzere iki grupta toplanabilir. Kayıpsız sıkıştırma, bilginin, sıkıştırılan işaretleri kullanılarak, asıl bilginin tekrar elde edilebildiği sıkıştırma yöntemidir ve genellikle bilgi doğruluğunun çok önemli olduğu bilgiler için kullanılır. Kayıplı sıkıştırma, bilgi sıkıştırıldığında, asıl bilgide bazı kayıpların oluştuğu sıkıştırma yöntemidir. Bu yöntem bilgi doğruluklarının çok önemli olmadığı bilgiler için kullanılmaktadır. Örneğin, görüntü ve ses bilgilerinde bulunan bazı işaretlenmiş bilgiler insan duyu organlarıyla algılanamıyorsa, bu bilgiler asıl bilgiden çıkarılabilir. Kayıplı sıkıştırma, sayısal televizyon sinyallerinin iletilmesinde de önemli bir rol oynamaktadır. Eğer high definition television (HDTV) sinyalleri sıkıştırılmadan iletilirse, saniyede yaklaşık 884 Mbit/sn bilgi iletilmesi gerekir. Böyle bir bilgiyi iletmede yaklaşık 220 Mhz bant genişligi olan bir kanala ihtiyaç duyulur. Fakat bu bilgi sıkıştırılarak iletilecek olursa 20 Mbit/sn den daha az bilgi iletim hızı ve 6 Mhz civarında bant genişliğine indirilebilir.

Bilgi Sıkıştırmanın diğer bir avantajı da bilgilerin daha az alanda saklanmasıdır. Örneğin dünyadaki pekçok ajanslar terabaytlar seviyesindeki bilgileri saklamak zorundadırlar. Bu tür geniş bilgi saklama çalışmalarında sıkıştırma yöntemleri oldukça yaygın olarak kulanılmaktadır. Sıkıştırma yöntemlerinin kullanılması depolama maliyetini de düşürmektedir. Görüntü sıkıştırmada kullanılan metodlar, aritmetik kodlama, sözlük temelli kodlama, huffman kodlama, vektör gruplama gibi yöntemler olduğu gibi, discrete fourier transform, discrete cosine transform, discrete sine transform gibi dönüşümler de kullanılmaktadır.

Görüntü sıkıştırmada yeni bir yaklaşım ise yapay sinir ağlarının kullanılmasıdır (Haykin and Dony, 1995). Yapay sinir ağlarının paralel yapıları gereği fazla sayıda birbiriyle ilişkisiz bilgiler arasında kolayca ilişki kurabilmeleri nedeniyle görüntü sıkıştırmada kullanılabilmektedirler. Yapay sinir ağları, öngörmeli kodlama yaklaşımlarında yardımcı eleman olarak kullanılabildiği gibi başlı başına, sıkıştırma işlemlerinde de kullanılabilirler. Bu işlem için yapay sinir ağı algoritmalarından biri olan geri yansımalı yapay sinir ağı algoritması görüntü sıkıştırmada kullanılabilir. Bu algoritma öğretmenli öğrenme algoritmasına sahiptir. Bu tür öğrenmede ağa giriş ve hedef çıkış bilgileri verilerek, hedeflenen çıkışlara uygun sonuçlar üretmesi istenir. Hedeflenen sonuçları üretmek için eğitim süresi boyunca, çıkışlar ile hedef çıkışlar karşılaştırılarak oluşan hata geri yansıtılır. Geriyansımalı ağın bu özelliğinden yararlanmak için, giriş değerleri ve hedef çıkış değerleri olarak görüntü bilgi matrisi verilerek belirlenen ara katman eleman sayısında kabul edilebilir hata düzeylerindeki ağırlık değerleri bulunmaya çalışılır. Ara katmandan alınacak olan çıkışlar eğitilen görüntünün sıkıştırılmış işaretleri olacaktır. Bu çalışmada üç katmanlı geri yansımalı ağ kullanıldı. Giriş katmanında 64, ara katmanda 4, çıkış katmanında ise 64 işlem elemanları kullanılarak 4/1 oranında sıkıştırma başarıldı.

2. SIKIŞTIRMA TEKNİKLERİ VE YAPAY SİNİR AĞLARI

Evrende en iyi görüntü anlama sistemi insanın görüntü anlama sistemidir. Çözümleme üzerinde yapılan yoğun çalışmalara rağmen insan algılama süreci henüz tam olarak anlaşılamamıştır.

İnsan algılama sisteminin anlaşılmasında kullanılan ilginç yaklaşımlardan biri de Gestalt kuramıdır. Kuramın amacı, algılama sürecini açıklayan ve idare eden kurallara benzer yasaları ortaya koymaktır. Bu yasalar, bilgisayarla görü araştırıcıları için geçerli çalışma varsayımları olmaktadır. Örneğin Gestalt kuramı resimsel sahneleri algılamak için gerekli olan bütün bilgiyi içerdiğini savunmakta ve algılama için gerekli olan bilgiyi tanıma problemini şöyle ifade etmektedir (Di Ruocco et.al., 1992):

1) Verilen bir resimsel sahnedeki herbir bölütün bilgisel katkısı belirlenmelidir.

2) Bu bölütlerin bilgisel katkıları arasında oluşan süreçler belirlenmelidir. Birbirleri ile etkileşen bu süreçler sahneyi oluşturmaktadır (Bütünlük teorisi).

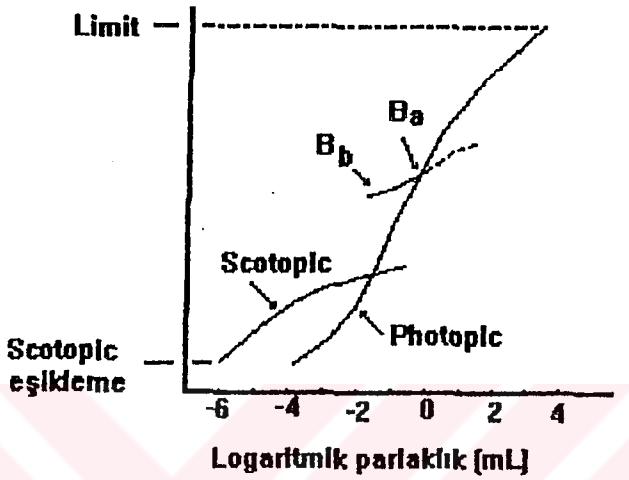
3) Bölütlerin topolojik dağılımının çözümlenmesi yapılmalıdır. Böylelikle sahnedeki mümkün olabilecek belirsizlikler çözümlenebilmektedir.

4) Sahnedeki görüntüyü tanımlayabilmek, görüntüdeki bölütlerin bilgisel katkısını ayırmaktır. Bu farklı yollarla yapılabilir: Eşikleme, piksellerin istatistiksel dağılımı (beraber oluşum matrisi), piksellerin uzamsal dağılımı gibi yöntemlerdir.

Sayısal imgeler noktalar halinde görüntülenir. Görüntü işlemede gözün parlaklıklar arasındaki seviye farkının algılaması, görüntü işlemenin en önemli noktasını oluşturmaktadır. Gözün algılayabildiği parlaklık aralığı, gece görme eşik değerinden 10^{10} mertebesine kadar çıkmaktadır. Şekil 2,1'de nesnenin parlaklığının ışığın yoğunluğuyla değişimi verilmiştir. Gözün algılayabildiği aydınlık aralığı şekilde uzun eğriyle gösterilmiştir. Gece görmesinde aralık 10^{6} mertebesindedir. Gece görmesinden gündüz görmesine geçiş 10^{-3} ile 10^{-1} miliLambert aralığında olmaktadır.

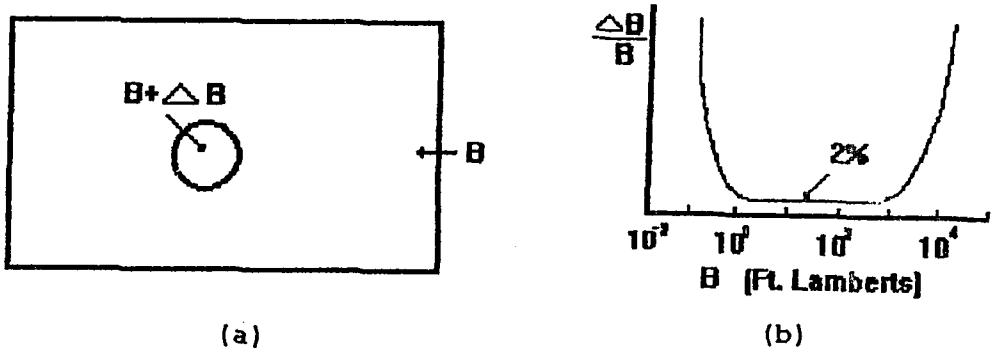
Şekil 2,1'de verilen açıklamada insan görme sisteminin geniş aralıkta gece ve gündüz görmesinin üst üste çakışması tanımlanamamaktadır. Bundan dolayı bu büyük aralıkta göz hassasiyetini değiştirmekte ve bu işleme de gözün parlaklık uyumu denilmektedir. Parlaklık yoğunluğu seviyesi ağırlığı, uyum ağırlığının yanında oldukça küçük kalmaktadır. Verilen

herhangibir yoğunluk için görme sisteminin hassasiyet seviyesine de parlaklık uyum seviyesi denilmektedir.



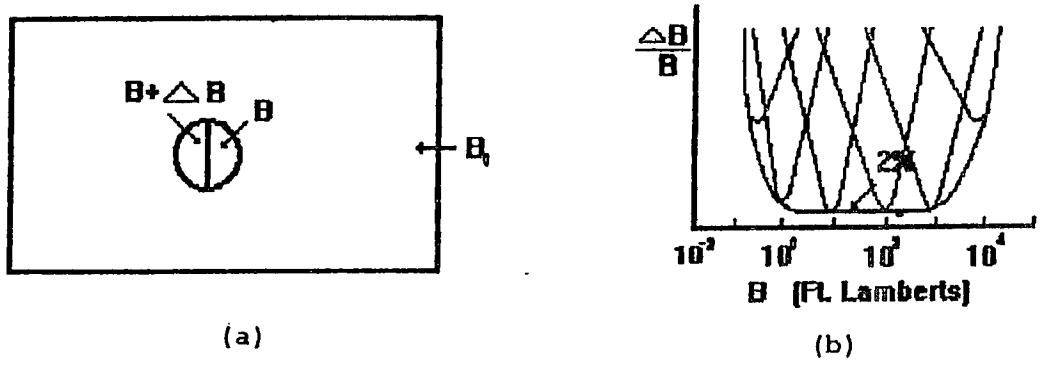
Şekil 2.1. Nesnenin parlaklığının ışığın yoğunluğuyla olan değişimi.

Bir gözün aydınlık hassasiyeti şu şekilde ölçülür; şekil 2,2 (a)' da gösterildiği gibi parlaklığı B olan bir zeminin ortasına B+ΔB parlaklığında bir daire yerleştirilir. ΔB sıfırdan başlayarak artırılır. Ortadaki daire arka plandan farklı bir parlaklık olarak algılandığı andaki ΔB değerine, en küçük algılanabilen fark (just Noticable Difference, JDN) denir. ΔB (JDN), B' ye bağlı olarak ölçülür.



Şekil 2.2. Arka plan parlaklığına göre parlaklığın algılanması.

ΔB/B oranına Weber oranı denilmekte ve bu oran, şekil 2.2 (b)' de gösterildiği gibi parlaklık aralığının büyük bir bölümünde -yaklaşık olarak 0.02 değerinde- sabittir.



Şekil 2.3. Değişken arka plan parlaklığı ile parlaklık algılanması.

Şekil 2.3' ten anlaşlacağı gibi insan gözü çok dinamik bir yapıya sahiptir. Ne yazık ki bu, gerçeğe yakın görme modeli değildir. Daha uygun sonuçlar elde etmek için şekil 2,3 (a)'daki gibi bir model kullanılmalıdır. Tekrar Weber oranı ΔB/B olarak ölçülür. B_0 çevre aydınlığı olup Weber oranının parametresidir ve ölçüm sonuçları şekil 2,3 (b)' de gösterilmiştir (Haralik and Shapirou, 1992).

Bir görüntü içerisinde yer alan bilgi birkaç faktöre bağlıdır. Bunlar; görüntüdeki piksel sayısını belirleyen ayrışım (resolution) ve herbir pikselin sahip olduğu ayrı renk sayısını belirleyen renk derinliği (color depth) olarak sayılabilir. Bunların yanında gürültü ve dinamik saha (dynamic range) da bu farklılıklara katkıda bulunmaktadır. Bu nedenle görüntünün tam esas bilgisini ölçmek oldukça zordur. Görüntü bilgisi, insanın görebilmesi için gerekenden fazla bilgi bulundurur. Bu fazlalıklar (Jain, 1981);

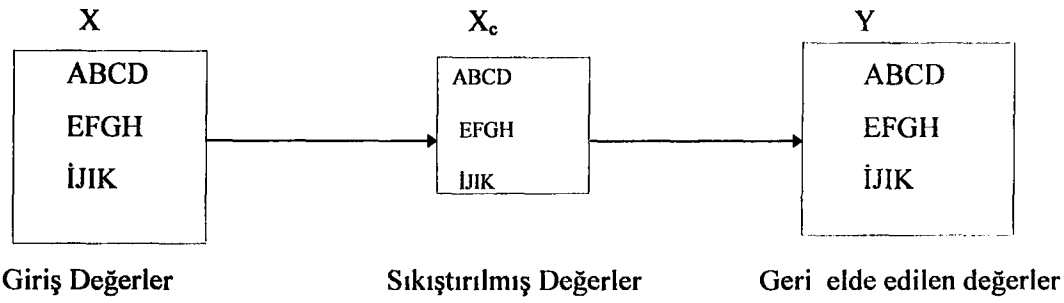
1) Komşu piksellerin karşılıklı ilişkisini tanımlayan konumsal fazlalık.

2) Farklı bölgelerin ilişkisini tanımlayan uzamsal fazlalık.

3) Yakın zamanlardaki aynı piksellerin ilişkisini tanımlayan zamana ait fazlalık.

Sıkştırma teknikleri olarak genellikle iki algoritma veya teknikten bahsedilebilir (Sayood,1996). Bu tekniklerdeki sıkıştırma algoritmalarında, giriş değerler X, giriş değerlerinden daha az olan sıkıştırılmış işaretler X_c ve geri elde edilmiş değerlerinden oluşan Y

ise, X_c sıkıştırılmış işaretleri işleyen geri elde edim algoritması şekil 2,4'de gösterilmiştir.



Şekil 2.4. Sıkıştırma işleminin sembolik gösterimi

Sıkıştırma işleminde bilgi işlenirken, bilgi çıktıya iki şekilde aktarılabilmektedir. Bunlar; bilgide kayıpların oluştuğu, bire bir sıkıştırmanın yapılmadığı kayıplı sıkıştırma ve kayıpsız olarak bilgi iletişiminin yapıldığı kayıpsız sıkıştırmadır.

2.1. Kayıplı Sıkıştırma

Bilgi sıkıştırılırken, orjinal bilgiden belli oranlarda bilginin kaybolduğu sıkıştırma yöntemine kayıplı sıkıştırma adı verilir. Kayıplı sıkıştırmalarda genellikle bazı bilgiler kaybolur ve tam olarak orjinal bilgi geri elde edilemez. Bilginin geri elde edilmesi durumunda, orjinal bilgide bazı bozulmalar olur (Jain, 1981). Pek çok bilgi iletişim uygulamalarında tamamen bilginin orjinalinin elde edilmesi önemli değildir. Örneğin ses örneklemesinin depolanmasında veya iletilmesinde tam değeri gerekli değildir. Bu, sesin geri eldesindeki kalite ile ilgilidir. Herbir örneklem değeri ile, ilgili bilgideki değişiklik kabul edilebilir oranlarda tutulabilir. Telefon konuşmalarında ve video bilgilerinde de genellikle bilginin bit bit özdeş olarak geri eldesi önemli değildir. Bundan dolayı bu tür bilgiler de kayıplı olarak sıkıştırılabilirler (Sayood, 1996).

Görüntü sıkıştırmada kayıplı yöntemin kullanılması ile, görüntü, sıkıştırılarak daha az bir alanda yerleştirilir ve görüntünün kodu çözüldüğünde, orjinaline çok benzeyen bir işaret

elde edilebilir. Şu andaki çalışmalar kalitede az bir kayba sebep olan fakat başarılı sıkıştıma oranlarını gerçekleştiren yaklaşımlar üzerine odaklanmış bulunmaktadır (Alonso, 1992; Sayood, 1996; Lım, 1994). Pekçok kayıplı yaklaşımlar daha az sıkıştırma pahasına daha iyi kalite kazanımlı durumlar üzerinde yoğunlaşmaktadır (Sayood, 1996).

2.2. Kayıpsız Sıkıştırma

Kayıpsız sıkıştırma, orjinal bilgi sıkıştırıldıktan sonra elde edilien çıkış bilgisinin, giriş bilgisine bit-bit özdeş olduğu sıkıştırma yöntemidir. Bu yöntem çok önemli bilgiler için uygulanmalıdır. Bu metod genellikle yüksek sıkıştırma oranı vermez. Sıkıştırılmış bilgiden tam olarak orjinal bilginin üretilmesini sağlayan veya hiçbir bilgi kaybının söz konusu olmadığı kayıpsız sıkıştırma tekniği, genellikle kesikli bilgiler için; örneğin text, bilgisayar da üretilen bilgi, bazı resim ve video bilgilerinde kullanılmaktadır (Jain, 1981).

Text sıkıştırma, kayıpsız sıkıştırmanın önemli bir alanıdır. Bu durum genellikle küçük farklılıklarla büyük anlam değişiklikleri oluşabilecek durumlarda önemlidir. Örneğin banka kayıtlarında " do not send money" ile "do now send money" cümleleri karakter olarak biribirine çok yakın kodlardan oluşmasına karşın anlam olarak çok farklıdır (Sayood, 1996).

Tıbda radyolojik görüntülerde de sıkıştırılmış bilgiden aynen orjinal bilginin elde edilmesi gerekebilir (Sayood, 1996).

Kayıpsız bilgi sıkıştırma teorisinde işlemler ikiye ayrılabilir. Bunlar ; modelleme ve kodlamadır: Modelleme giriş bilgi akıntısı üzerinde önceden haber üretme işlemidir. Karakterler için ikili (binary) ardışılları işaretleyen işleme ise kodlama adı verilir (Jain, 1981). Bir alfabe karakterlerden oluşur. Örneğin pek çok kitaplarda kullanılan alfabede 26 harf bulunmaktadır. a harfi için ASCII kod 100011 dir, A harfi ise 1000001 olarak işaretlenir. Görüldüğü gibi, ASCII kodu her harf için aynı sayıda bit kullanır. Eğer farklı mesajları işaretlemek için gerekli olan bit sayısının azaltılması istenirse, farklı bit sayısı kullanılmalıdır. Eğer sürekli kullanılan semboller daha az sayıda bit ile işaretlenirse, bu durumda ortalama olarak her sembol için daha az bit kullanılmış olur. Örneğin Mors alfabesi E için (:), Z için (- - ..) sembollerini kullanmaktadır.

Çizelge 2.1. Dört karakter için farklı dört kod kullanımı

Karakter	Kod1	Kod2	Kod3	Kod4
a_1	0	0	0	0
a_2	0	1	10	01
a_3	1	00	110	011
a_4	10	11	111	0111
Ortalama bit kullanım oranı	1.125	1.25	1.75	1.875

Çizelge 2,1'de verilen dört karakterin olasılıkları: $P(a_1)=1/2$, $P(a_2)=1/4$, $P(a_3)=P(a_4)=1/8$ olarak belirleyebiliriz. Olasılıktaki ortalama uzunluk L ise;

$$\mathbf{L} = \sum_{i=1}^{4} \mathbf{P(a_i)n(a_i)} \qquad (2.1)$$

ifadesi ile hesaplanır. Çizelge 2,1'de verilen a_1, a_2, a_3 ve a_4 için kullanılan dört farklı kodlama için kullanılan ortalama uzunluk eşitliğinde, $n(a_i)$, a_i harfi için kod kelimedeki bit sayısıdır. Kod1'de a_1 ve a_2 harflerinin kod kelimesi (0) dır. Bu durum iletimde karışıklıklara sebep olur. Bu nedenle her sembol için tek bir kod kelimesi kullanılır. Kod2'de aynı anlama gelme problemi yoktur. Her bir sembol ayrı kod kelimesiyle işaretlenmiştir. Kod2 ile a_2 a_1 a_1 harfleri 100 ikili yazım ile kodlanır. Bu kod, kod çözücüye ulaştığında a_2 a_1 a_1 veya a_2 a_3 olarak kodu çözülebilir. Bu durmda orjinal bilgi elde edilemez. Bu nedenle kodlarda tekli çözülebilme özelliği aranmalıdır (Sayood, 1996).

Kayıpsız sıkıştırma algoritmalarının iki ana devresi vardır. Bunlar minimum fazlalık kodlama ve sözlük temelli sıkıştırmadır. Bu devreler günümüzde kullanılan pek çok kayıpsız sıkıştırma için temeldir (Alonso, 1992). Minimum fazlalık kodlama, aritmetik kodlama ve run length kodlama ile gerçekleştirilebilir.

2.2.1. Minimum fazlalık kodlama

Bilgi sıkıştırma ile ilgili çalışmalar Claude Shannon'un enformasyon teorisi üzerine çalışmaları ile başladı (Sayood, 1996). Bilgi teorisindeki entropi terimi bir sembol bulunma olasılığının negatif logaritması olarak tanımlanır. Bilgi akıntısının entropisi bireysel bütün sembollerin entropisinin toplamıdır. Bir mesaj entropisi mesajın bilgi içeriğidir. Böylece bit sayısı:

$$\text{bit sayısı} = -\log(\text{probability}) \tag{2.2}$$

ile hesaplanır. Mesaj üzerindeki bit fazlalığı mesajdaki fazla bilginin var olduğunu gösterir ve fazlalığın varlığı bilgi sıkıştırmaya imkan sağlar.

Text türü için çok yaygın bir işaretleme olan ASCII kodu -bütün karekterler aynı bit sayısı ile işaretlendiğinden- fazlalıklı bir işaretlemedir. Eğer farklı sembollerin bilgi akıntısında görünmesinin olasılığı farklı ise bu durumda değişken uzunluk kodlar ile bilgi özlerinin herbirisinin olabildiğince fazla bitlerinin alınması gerekir. Bu model, statik ve dinamik olmak üzere ikiye ayrılır. Statik model; işaretli bilgi bloklarının analizi yapılarak seçilen bir modelin bu bilgi bloklarına uygulanmasıdır. Bu model benzer diğer bilgi bloklarını sıkıştırmak için kullanılabilir. Fakat bu sıkıştırma sadece modeli oluşturmak için kullanılan bilgi bloklarına olabildiğince benzer olduğu durumlarda kullanılabilir. Eğer sıkıştırılacak bilgi nesnesi bu modeli oluşturan elemanlardan birinden farklı ise sıkıştırma düzgün olmayacak ve sıkıştırmadan sonra geri elde edilme durumunda bit sayısı azalacaktır (Sayood, 1996). Dinamik modelleme her bir bilgi akıntısı için mümkün olan en iyi modelin oluşturulduğu bir modeldir. Fakat her bir bilgi akıntısı farklı model olduğundan bu modelin kendi sıkıştırılmış bilgisi ile depolanmış olması gerekir. Bu modelde bilgi depolamak için gerekli kısımlar, sonlayıcı faktör ile ayrılmalıdır (Sayood, 1996).

Kodlama işlemi üç farklı yöntemle yapılabilmektedir. Bunlar Shanon-Fano, Huffman ve Adaptif Huffman kodlamadır.

a) Shanon-Fano Kodlama

Bu metod C.E.Shanon ve Robert M.Fano tarafından geliştirilmiştir. Bu yöntemle mesajdaki her bir sembolün olasılığı biliniyorsa bu mesaj daha az yer kaplayacak tarzda kodlanabilir (Alonso, 1992). Kodlamada, kodu çözülebilen degişik uzunlukta kodlar kullanılabilir. Karekter kümesinde bulunan her bir karekterin Shannon-Fano kodunu geliştirmeden önce herbir karekterin bulunma olasılığının belirlenmesi gerekir. Daha sonra herbir karekterin bulunma olasılığı üzerine kurulu soy [descending] düzeni içinde karekter kümesi düzenlenmelidir (Held, 1991). Bir kez karekter kümesi düzenlendiğinde, bu küme, herbir altkümedeki karekterlerin bulunma olasılığı üzerine kurulu iki eşit veya yaklaşık eşit iki alt kümeye bölünür. Bir kümenin birinin ilk basamağı 0 değeri iken ikinci alt kümenin ilk basamağı 1 değerini alır. Bu alt kümeleri şekillendirme işlemi tüm karekter kümesi bölünene kadar devam eder. Daha sonra alt küme içindeki bir karekterin ikili karşılığını diğer karekterden ayırmak için iki karekter arasına sonek biti (suffix bit) eklenir (Held, 1991). Çizelge 2,2'de 7 karekter içeren bir kümedeki karekterlerin bulunma olasılıklarını göstermektedir (Held, 1991).

Çizelge 2.2. Karekter küme bulunma olasılığı.

Karakter	Olasılık (P)
a_1	0.10
a_2	0.05
a_3	0.20
a_4	0.15
a_5	0.15
a_6	0.25
a_7	0.10
	P=1.00

Daha sonra bulunma olasılıklarına göre mümkün olduğunca eşit alt kümelere gruplandırılır. Bu gruplara daha sonra 1 ve 0 bitleri yerleştirilerek kodlar elde edilir. Bu durum şekil 2,5'de gösterilmiştir (Held, 1991).

Karakter	Olasılık	Kod		
a_6	0.25	1		
a_3	0.20	1		
a_4	0.15	0	1	
a_5	0.15	0	1	
a_1	0.10	0	0	1
a_7	0.10	0	0	0
a_2	0.05	0	0	0

Şekil 2.5. Çizelge 2.2'de verilen a_i'den a_7'ye kadar olan karakterlerin Shanon-Fano kodlaması.

Şekil 2,5'te alt kümelere ayrılan karekterlerden bazıları tekli kod kelimesine sahip değildir. Bu nedenle her bir alt kümeye 1 ve 0 ilave edilerek kodlama işlemi tamamlanır. Bu durum şekil 2,6'da görülmektedir (Held, 1991).

Karakter	olasılık	kod		
a_6	0.25	1 1		
a_3	0.20	1 0		
a_4	0.15	0	1 1	
a_5	0.15	0	1 0	
a_1	0.10	0	0	1
a_7	0.10	0	0	0 1
a_2	0.05	0	0	0 0

Şekil 2.6. Tamamlanmış Shanon-Fano kodlama işlemi

Şekil 2.6 da görüldüğü gibi en yüksek olasılıklı a_6 ve a_3 iki bit ile, 0.15 olasılıklı a_4, a_1 ve a_5 üç bit ile, a_7 ve a_2 dört bitle kodlanmıştır. Shanon-Fano kodlamanın en önemli karekteristikleri;

1) Ençok olası semboller küçük bir bit sayıları ile uyumlu kodlara sahiptir. En az olası semboller en uzun kodlara sahiptir.

2) Kodlar tek olarak çözülebilir.

3) Kodların tekli, çözülebilir olması ve ikili sistemle yapılmasıdır. Kod ağacı kökünden başlar ve bu yön boyunca bitleri toplayıcı dalların uyumu bir yaprağa ulaşıncaya kadar devam eder (Held, 1991).

b) Huffman Kodlama

Hufman Kodlama algoritması David Huffman tarafından geliştirilmiştir. Bu tekniği kullanarak üretilmiş kodlara Huffman kodları adı verilir. Bu kodlar verilen bir model için optimumdur ve aynı zamanda prefix kodlardır. Hufman prosedürü optimum prefix kodları ön gören iki temel algoritma üzerine kurulmuştur.

1) Bir optimum kodda çok sık olarak bulunan semboller daha az bulunan sembollere göre daha kısa kod kelimelerine sahip olmalıdır.

2) Bir optimum kod içerisinde en az sıklıkla bulunan semboller aynı uzunlukta olamalıdır (Sayood, 1996).

Huffman prosedürü bu iki tanıma basit bir gereksinim ilave edilerek elde edilir. Bu gereksinim, iki endüşük olasılıklı sembollere karşılık gelen kod kelimelerinin son bitlerinde farklılık olmasıdır. Örneğin alfabede γ ve δ en az olasılıklı karakterler ise bu durumda γ, m*0 kod ile δ, m*1 kod ile kodlanır. Huffman kodun dizaynında a_1, a_2, a_3, a_4, a_5 karakterlerinden oluşan A alfabesinde olasılıklar; $P(a_1)=P(a_3)=0.2$, ve $P(a_2)=0.4$, $P(a_5)=P(a_4)=0.1$ olduğunu kabul edelim.

Çizelge 2.3. Beş karakterli A alfabesinin olasılık ve kod kelimeleri.

Karakter	olasılık	kodkelime
a_2	0.4	$c(a_2)$
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$
a_4	0.1	$c(a_4)$
a_5	0.1	$c(a_5)$

Çizelge 2,3'te görüldüğü gibi a_4 ve a_5 en düşük olasılıklı iki semboldür. Bu nedenle onları ; $c(a_4)=\alpha_1*0$, $c(a_5)=\alpha_1*1$ şeklinde gösterebiliriz, burada α_1 ikili kareterdir. *0 ve *1 ise α_1' e ilave bittir. Şimdi yeni A' alfabesi, a_1, a_2, a_3, a_4' karakterlerinden oluşan dört karakterli bir alfabe olarak tanımlanabilir. Burada a_4', a_4 ve a_5 karakterlerinden oluşmaktadır. Olasılık olarak , $p(a_4')= p(a_4)+p(a_5)=0.2$ yazılabilir. Bu alfabede a_3 ve a_4' listenin altında iki karakterdir ve bunların kodkelimeleri; $c(a_3) = \alpha_2*0$, $c(a_4') = \alpha_2*1$ şekilde işaretlenebilir. Fakat $c(a_4') = \alpha_1$ ve $\alpha_1= \alpha_2*1$ olduğundan, böylece a_4 ve a_5 karakterlerinin kod kelimeleri $c(a_4) = \alpha_2*10$, $c(a_5) = \alpha_2*11$ olur. Bu adımda a_1, a_2, a_3' karakterlerinden oluşan yeni bir A'' alfabesi tanımlanabilir. A'' alfabesindeki a_3' karakteri, a_3 ve a_4' karakterlerinden oluşur ve a_3' karakterinin olasılığı: $P(a_3')=P(a_3)+ P(a_4')$ olur. Bu durumda iki en küçük olasılıklı karakterler olan a_1 ve a_3' karakterlerinin kod kelimeleri $c(a_3')= a_3*0$, $c(a_1) = a_3*1$ tür. $c(a_3')= \alpha_2$ olduğundan $\alpha_2=\alpha_3*0$ olur. Böylece kod kelimeleri; $c(a_3)= \alpha_3*00$, $c(a_4)=\alpha_3*010$, $c(a_5)= \alpha_3*011$ yazılabilir. Yeni bir alfabe bu kez a_3'' ve a_2 karakterlerinden oluşur. Bu alfabede a_3' ve a_1 karakterlerinden oluşan a_3'' karakterinin olasılığı; $P(a_3'')=P(a_3')+P(a_1)=0$. İki karakterli bir alfabe meydana gelir. Alfabede bulunan karakterlerin kod kelime işareti: $c(a_3'')=0$, $c(a_2)=1$

şeklinde verilebilir. Bu durumda kod kelimeleri: $c(a_2)=1$, $c(a_1)=01$, $c(a_3)=000$, $c(a_4)=0010$, $c(a_5)=0011$ olur (Sayood, 1996). Bu durumda oluşan orjinal beş karakterli A alfabesi aşğıdaki çizelge 2,4'de gösterilmiştir (Sayood, 1996).

Çizelge 2.4. Beş karakterli A alfabesi için Huffman kodları.

Karakter	Olasılık	kod kelime
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011

Bu kodun ortalama uzunluğu :

$$L=(0.4 \times 1)+(0.2 \times 2)+(0.2 \times 3)+(0.1 \times 4)+(0.1 \times 4)=2.2 \text{ bit / karakter} \qquad (2.2)$$

olarak bulunur (Sayood, 1996).

c) Adaptive Huffman Kodlama

Huffman kodlama sembolleri kodlamak için sadece sembol olasılığını kullanır. Bu kodlamanın etkinliği her bir sembolün olasılığını hesaplamak için kullanılan modellere bağlıdır. Her bir sembolün olasılığı, sembollerin üstünlüğü tanımlanmaksızın hesaplanan sıfır modelde sıkıştırmayı kodlamak için sadece kod çözücü için kod ağacı ve bilgi gerekir. Eğer kod ağacı durgun ise bu kod çözücünün parçası olabilir ve bunun kod çözücüye aktarılması gereksizdir (Alonso, 1992). İletici ve alıcı aynı ağaç ile başlar ve kullanılan güncelleştirme işlemi her ikisi için de aynıdır. Bu nedenle kodlama ve kodçözme işlemleri eşzamanlı olur. (Sayood, 1996).

2.2.2. Aritmetik kodlama

Aritmetik kodlama özellikle içerisinde az sayıda karakter türü bulunduran kaynaklarda kullanışlıdır. Bu kodlamanın faydalı yönü kayıpsız sıkıştırmanın modelleme ve kodlama yönleri ayrılmayı ve bölünmeyi saklamasıdır. Aritmetik kodlamada ardışılları (sequences) kodlamak için etiket (tag) veya tanımlayıcılar (identifiers) üretilir. Bu etiketler ikili ardışıllara benzer. Uygulamada etiket üretimi ve ikili kod üretimi aynı işlemdir. Bununla birlikte bu işlem iki kısımda incelenebilir. İlk adımda verilen sembollerin ardışılı için bir tekli etiket veya tanımlayıcı üretilir. Bu etikete daha sonra ikili kod verilir. Tekli bir aritmetik kod, m uzunluklu bütün ardışıllar için kod kelimeler üretilmesine ihtiyaç duyulmaksızın, m uzunluğunda bir ardışıl için üretilebilir. Bu durum Huffman kodlamadan farklıdır. Bir başka tanımla bir ardışılın Huffman kodunu üretmek için bütün m uzunluklu ardışılların kod kelimelerini üretmek gerekir (Sayood, 1996).

Bir ardışılın sembollerini başka bir ardışılın sembollerinden ayırmak için tekli tanımlayıcı ile onun etiketlenmesi gerekir. Bir semboller ardışılını işaretlemenin bir yolu birim aralıktaki (0,1) sayılar ile onu etiketlemektir. Çünkü birim aralıktaki sayı sonsuz olduğundan ortaya çıkan ardışıllar için tekli etiket işaretlemek mümkün olabilir. Her bir sembolü kodlamak için bit sayısını tam sayı kullanmayan bir kodlama tasarımı mümkündür. Bu kodlama aritmetik kodlama tarzındadır (Sayood, 1996).

$$X(a_i)=i \quad a_i \in A \tag{2.3}$$

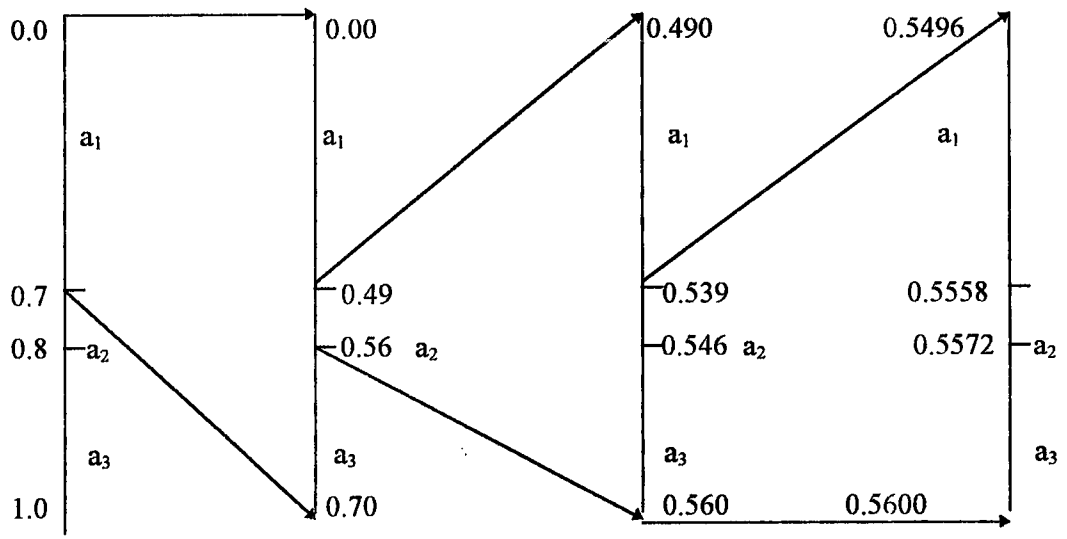
Burada $A=\{a_1,a_2,...a_m\}$ kaynak alfabe, X rastgele(random) değişkendir. Bu random değişken için olasılık yoğunluk fonksiyonu:

$$P(X=i)=P(a_i) \tag{2.4}$$

ve kümülatif yoğunluk fonksiyonu (cumulative density function) :

$$F_x(i) = \sum_{k=1}^{i} P(X = k) \tag{2.5}$$

Kümülatif yoğunluk fonksiyonunun en düşük değeri (0) en büyük değeri (1) dir. Birim aralığı tam bölmek bu şekilde mümkün olur. Ardışılın ilk karekterinin görülmesi etiketin içerdiği aralığı sınırlar. Farzedelim ki ilk karakter a_k olsun. Bu durumda $X(a_k) = x_k$ olur. Böylece bu etiket değerini kapsayan aralık $[F_x(x_k-1), F_x(x_k))$ alt aralığı olacaktı. Bu alt aralık orjinal aralık olarak aynı oranda bölünebilir. Takip eden sembol bir alt aralık için bu etiketi sınırlamaya neden olur (Sayood, 1996). Şekil 2.7'de $A=\{a_1,a_2,a_3\}$ ve $P(a_1)=0.7$, $P(a_2)=0.1$ $P(a_3)=0.2$ ise Bu durumda eşitlik.. ten $f_x(1)=0.7$, $f_x(2)=0.8$, $f_x(3)=1$ olduğu gösterilmektedir.



Şekil 2.7. Karakter olasılıklarına göre birim aralığın bölünmesi.

Şekil 2,7'de görüleceği gibi ilk sembol a_1 ise 0.0 ile 0.7 aralığına, ilk sembol a_2 ise 0.7 ile 0.8 aralığına, ilk sembol a3 ise 0.8 ile 1 aralığına yerleşir. Eğer ilk sembol a_1, ikinci sembol a_2, üçüncü sembol a_3 olduğu durumda sınırlama işleminin nasıl gerçekleştiği Şekil 2,7' de görülmektedir. Herbir mesaj içindeki toplanan karakterlerin olasılıkları ile orantılı bir değer olasılık ara değerini azaltır.

Son kod, orjinal alfabeyi geri elde etmek için, kodu çözülen ve tek olabilen ara sonucundan tekli gerçel bir sayıdır. Başlangıçta mesaj 0-1 aralığı ile işaretlenir. İlk sembol ile , ara, bu sembole ayrılan ara ile azaltılır. Bu işlem bütün semboller kodlanana kadar devam ettirilir (Sayood, 1996). Matematiksel olarak ; $A=\{a_1,a_2,a_3...a_m\}$ alfabesinde a_i sembolü için etiket değeri;

$$\mathbf{T_x(a_i) = \sum_{k=1}^{i} P(X=k) + 1/2(P(X=i))} \quad (2.6)$$

$$\mathbf{=F_x(i-1)+1/2(P(X=i))} \quad (2.7)$$

her a_i için $T_x(a_i)$ tekli bir değer üretecektir ve bu değer a_i için tekli etiket olarak kullanılabilir. Bu etiket bu ardışıl için tek (unique) ikili oluşturarak etiketin taşınabileceği birim aralıktaki hangi değerler üzerine sınırlamalar yapılacağı belirtilmeksizin yerleştirilir. Bu değerlerin bazı ikili işaretlemeleri sonsuz uzunlukta olabilir. Bu durumda kod her nekadar tekli olsada verimli olmayabilir. Bu durum aritmetik kodun hem tekli hem de verimli olabilmesi için kök (truncate) olmak zorundadır (Sayood, 1996).

$T_x(x)$ [0,1) aralığında bir sayıdır. $T_x(x)$ için bir tek ikili kod bu sayının ikili işaretlemesi eşitlik (2.8) kullanarak;

$$\mathbf{I(x)=[log(1/P(x))]+1} \quad (2.8)$$

gerçekleştirilebilir. $A=\{a_1,a_2,a_3,a_4\}$ alfabesinde olasılıklar; $P(a_1)=1/2$, $P(a_2)=1/4$, $P(a_3)=1/8$ $P(a_4)=1/8$ ise böyle bir alfabenin ikili kodları çizelge 2,5'de gösterilmiştir (Held, 1991).

Çizelge 2.5. Dört karakterli $A=\{a_1,a_2,a_3,a_4\}$ alfabesinin ikili kodu.

Karakter	F_x	T_x	İkilide	Kök	Kod
a_1	0.5	0.25	0.010	2	01
a_2	0.75	0.625	0.101	3	101
a_3	0.875	0.8125	0.1101	4	1101
a_4	1.0	0.9375	0.1111	4	1111

2.2.3. Sözlük temelli kodlama

Pek çok uygulamada kaynak çıkışı, tekraralanan vektörleri içerir. Örneğin bir text kaynakta belirli vektörler veya kelimeler sürekli tekrarlanır (Sayood, 1996). Böyle kaynakları kodlamak için uygun bir yaklaşım, sıklıkla görülen vektörlerin listesini veya sözlüğünü tutmaktır. Kaynak çıkışında bu vektörler görüldüğünde sözlük bir referansla kodlanır. Eğer vektör sözlükte görünmezse o zaman daha az verimli diğer bir metot kullanılarak kodlanabilir. Bu işlemler ile giriş, sık görülen vektörler ve sık görülmeyen vektörler olmak üzere iki sınıfa ayrılmaktadır. Bundan dolayı sözlüğün hacmi tüm olası vektörlerin sayısından daha az olmaktadır (Sayood, 1996). Kısaca bir örnekle açıklanacak olursa; bütün kelimeleri ile bir kitap düşünelim. Her bir kelime sayfa numaraları ve sayfa içindeki konumları ile tekli olarak belirlenebilir. Şimdi kelimeler yerine bu kitaptan sayfa numara ve konumları kullanarak cümleler kurmak mümkündür. 20 bit içinde herhangi bir kelimenin sayfa ve konumunu açıklamak mümkündür. İngilizcede bir kelimenin ASCII kodu kullanımı ortalama 40 bit uzunluğundadır. Bu örnekte mesaj hacminde yarı yarıya bir azalış başarılmış olur (Alonso, 1992). Bir sözlük oluşturmak için durgun ve uyumlu olmak üzere iki metod kullanılabilir.

a) Statik Sözlük Sıkıştırma

Bu kodlama özellikle belirli uygulamalar için uygundur. Örneğin bir üniversitedeki öğrenci kayıtlarını sıkıştırmak gerektiğinde statik sözlük yaklaşımı en uygun seçim olabilir çünkü bu kaydın tamamında görülecek belirli kelimeler-isim, numara gibi- önceden bilinir. Bu durumda, tekraralanan vektörleri içeren statik bir sözlük üzerine kurulu sıkıştırma şeması

tanımlamak çok verimlidir. Bu şema uygulamaya-özel veya bilgiye-özel statik sözlük temelli kodlamada çok etkilidir. Bu şemalar başka uygulamalarda sıkıştırma yerine genişletmeye neden olabilirler (Sayood, 1996).

b) Adaptif Sözlük Sıkıştırma

En yaygın sözlük tasarımları uyumlu olanlardır. Bu sistemler sonlu sözlük ile veya hiçbir sözlük olmaksızın başlayabilir. Karakter dizileri işlenmiş mesaj olarak sözlükte toplanır. Adaptif sözlük sıkıştırma tasarımları 1970 sonlarında Jakopz ve Abraham'ın çalışmaları ile iki temel sıkıştırma metodlarına yoğunlaşmıştır. Bunlar LZ77 ve LZ78 olarak bilinir. LZ77 de sözlük, bildirimin önceki parçası üzerinden kayan bir penceredir. LZ78 de sözlük, bildirimdeki bir uyuşum bulunduğunda sözlükteki girişe bir fazla sembol ilave edilerek yapılabilir (Sayood, 1996).

2.2.4. Run Length kodlama

Run lenght kodlama, sadece bir tekrar sayısı ve bir karakterli tekrarlanan karakterlerin uzun dizilerini kodlayan bir kodlamadır. Bu kodlama, bilgi sıkıştırmak için, tekrarlanan karakter ardışılını azaltma yöntemine dayanır. Run lenght kodlamada özel karakterler kullanılması gerekir. Sıkıştırma gösrerge karekteri kullanıldığında, tekrarlanan karekterler izlenir. Sonuçta bir sayım karakteri ile bu ardışılda tekrar eden karakterlerin sayısı gösterilir (Held, 1991).

C_c =Karakter sayıcı

S_c =Özel karakter işaretleyen

X = herhangi bir tekrarlanan karakter

Şekil.2.8. Run length kodlama sıkıştırma formatı.

Orjinal Bilgi	kodlu bilgi
$******55.72	S_c$*655.72
---------	S$_c$-9
Gunsbbbbbbbbbb	GunsS$_c$b10

Şekil 2.9. Run Length kodlamada tekrar eden karakterlerin kodlanmış durumu .

Şekil 2,9'da görüldüğü gibi 12 karakterlik bilgi 9 karakterde ifade edilebilmiştir. Yine 9 karakterlik (-) karakterlerinden oluşan bilgi 3 karakterde saklanabilmiştir. Bu kodlama tekrar edilen karakterlerin fazla olduğu bilgilerde çok etkilidir. Örneğin iki seviyeli (beyaz-siyah) görüntüler için çok kullanışlıdır (Held, 1991).

2.3. Görüntü Sıkıştırma Kodlayıcıları

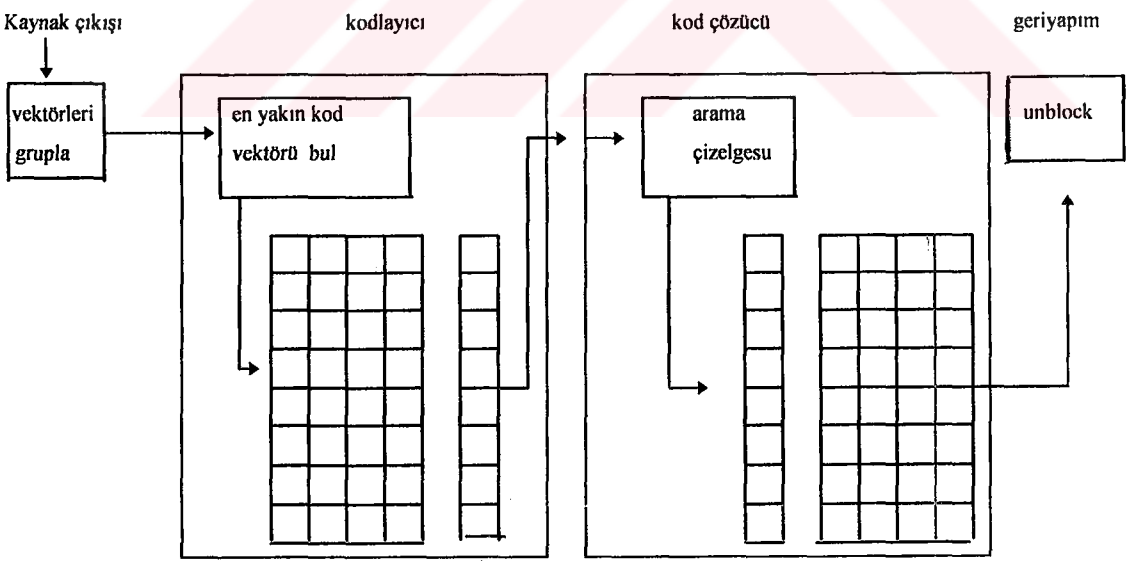
Görüntü sıkıştırma metodları içinde çok miktarda metodlar bulunmakta ve bunlar yazılım, donanım veya her ikisini kullanarak yürütülmektedir. Kodlayıcılar daha iyi performans elde etmek için birleştirilebilirler. Örneğin Discrete Cosine Transform (DCT) ve Delta Pulse Cod Modulation (DPCM), Sub-band decomposition ve DCT, veya DPCM ve Vektör Quantizasyon gibi karışımlı yaklaşımlar karışık kodlama olarak adlandırılırlar. Tipik bir birleştirme; özel sıkıştırma için transform kodlama, istatistiksel sıkıştırma için Huhuffman veya aritmetik kodlama kullanılmaktadır (Lim, 1994).

2.3.1. Dalga şekli kodlayıcıları

Dalgaform kodlamada hedef görüntüyü tanımlayan dalga şeklini basit bir şekilde kodlamaktır. Bu kodlama; herbir pikseldeki görüntü yoğunluğunun doğrudan kodlanmasıdır veya birbirini takip eden piksel yoğunluklarının arsındaki farkın kodlanmasıdır. Dalgaform kodlayıcılarının kolaylığı kavram ve hesaplama açısından basitliğidir (Lim, 1994). Bu yöntem uygulamada PCM, DM, DPCM, gibi yöntemlerdir.

2.3.2. Vektör gruplama

Vektör gruplamada kaynak çıkış bloklara veya vektörlere gruplanır. Örneğin bir görüntüden L piksel bloğu ele alındığında herbir piksel değeri L boyutlu bir vektörün bileşenleri olur. Kaynak çıkışında bu vektör, vektör gruplayıcı için tanımlanır. Vektör gruplayıcının kodlayıcı ve kod çözücüsünde L boyutlu küme vardır. Bu kümeye vektör gruplayıcının kod kitabı adı verilir. Bu kod kitabındaki vektörler, kod vektörler olarak bilinirler ve kaynak çıkışından üretilen vektörlerin işaretlenmesinde kullanılırlar. Seçim işleminde, kodlayıcıda bulunana giriş vektör, kod vektörler ile karşılaştırılarak en yakın kod vektör bulunur. Böylece kaynak çıkışın gruplanmış değerleri elde edilir. Kod çözücüyü bilgilendirmek için kod vektörün ikili kodu iletilir. Çünkü kod çözücü de aynı kod kitabına sahiptir (Nasrabadi and King, 1988). Bu işlemin sembolik gösterimi şekil 2,10'da gösterilmiştir (Sayood, 1996).



Şekil 2.10. Vektör gruplamanın sembolik gösterimi

V.Q'nun tercih edilen yönlerinden biri hızlı kod çözme yeteneğine sahip olmasıdır (Sayood, 1996). Fakat V.Q'nun bazı istenilmeyen yönleride vardır. Bunlar; çok yoğun kodlama

işleminin olması ve depolama gereksinimlerinin fazla olmasıdır. Ayrıca blok kenarlarında blok etkisinin görülmesidir (Sayood, 1996). Ortalama bozunumu en aza indiren kodlayıcı için temel algoritma;

1) Bir öğrenme ardışılı(sequence) kodlayıcıya verilir.

2) Gruplama bölgesi bulunduktan sonra bozunum hesaplanır ve verilen eşik değeri ile karşılaştırılır. Eğer ortalama bozunum yeterince küçükse durur.

3) Herbir gruplama bölgelerinin elemanlarının ortalama değeri olan yeni geriyapım değerini bulunur ve ikinci adıma gidilir (Linde, Buzo and Gray, 1980).

L.B.G algoritması olarak bilinen bu algoritma Linde, Buzo ve Gray tarafından geliştirilmiştir. Görüntü kodlamak için V.Q tekniklerinin pekçok uygulamaları geliştirilmiştir. Bunlardan Transform kodlama, istatistiksel olarak bağımlı skalerleri bağımsız katsayılara dönüştürür. Görüntü sıkıştırmak için, bit işaretleme kullanılarak tekdüze olmayan skalerlerin elde edilmesidir. Adaptif kodlama için bit işaretleme matrisleri, blokların sınıflandırılmasına bağlı olarak, farklı bloklar için kullanılabilir. Bir diğer basit teknik, görüntünün dikey tarama çizgileri boyunca bir boyutlu transform kullanmaktır. Bu durumda çizgiler arası katsayılar ayrılarak kolon katsayıları kullanılabilir. Bu durum iki boyutlu transforma genişletilerek görüntü bloklara bölünebilir ve bir transform herbirine bağımsız olarak uygulanabilir.

2.3.3. Değişim kodlayıcıları

Görüntü bilgilerinin kodlanmasında bazı değişim kodlayıcılarından yararlanılabilir. Bu kodlayıcılar görüntü piksel değerlerinin belirli bölgelerde toplanmasını sağlayarak görüntünün daha az alana yığılmasını sağlarlar (Lim, 1994).

D.C.T (discrete cosine transform) piksel bloklarını kullanarak, onları bir başka boyuta dönüştürür. Bu boyut piksel yoğunluk boyutundan farklıdır. Edim ve hesaplama maliyeti için D.C.T en güzel seçim olarak düşünülebilir. D.C.T ile bit sayısını azaltmak için iki mekanizma kullanılır;

a) Bazı değişim katsayıları, sezilmiş görüntü içeriğine fazla bir katkıda bulunmadıklarından çıkarılabilir.

b) Bazı değişim katsayıları kabaca gruplanabilir ve bu resim kalitesini fazla miktarda etkilemez (Lim, 1994).

D.F.T (discrete fourier transform) ilk görüntü kodlama sistemlerinde kullanıldı. Fakat D.C.T kadar iyi bir enerji yoğunluğu sağlamaz. D.F.T blok sınırlamasında keskin süreksizlik gösterir (Lim, 1994).

D.S.T (discrete sine transform) tekrarlı blok kodu durumunda iki parçaya ayrılabilen görüntü bilgisinde kullanılması uygundur (Lim, 1994).

2.4. Yapay Sinir Ağları

Yapay sinir ağları teknolojisi bilgisayar dünyasında insan beyninin ve sinir sisteminin davranışlarını taklid etme esası üzerine kurulmuş yeni bir bilgi işleme yaklaşımıdır. Bu ağlar birbirine parelel olarak bağlanmış işlem elemanlarından (yapay sinir hücresi) ve onların hiyerarşik bir organizasyonundan oluşurlar. Yapay sinir ağları, daha çok biyolojik sistemlerin hücreler üzerinde dağıtılmış bilgiyi parelel olarak işleme özelliklerinden yararlanan bir mekanizmadır. Hücreler birbirine bağlı ve parelel çalıştıkları için bazılarının işlevini yitirmesi ile sinir sistemi fonksiyonunu yitirmez. Bu ağların temel amacı gerçek dünyadaki nesnelere ve olaylara karşı biyolojik sinir sisteminin davrandığı gibi davranmaktır. Fakat günümüzde bu noktaya ulaşıldığını söylemek mümkün değildir (Lippmann, 1987).

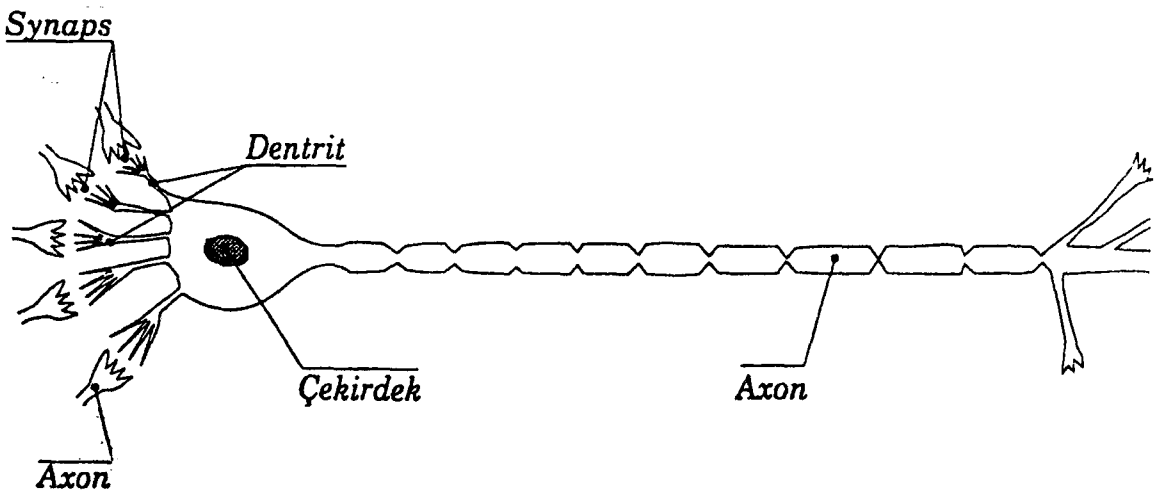
Son on yıl içinde yapay sinir ağları üzerindeki çalışmalar hızla artmaktadır. Yapay sinir ağlarının mühendislik başta olmak üzere birçok bilimsel alanda, karmaşık ve belirsiz veriler altında problemlere çözümler ürettikleri ispatlanmıştır. Algoritmasız tamamıyle parelel, adaptif, öğrenebilen ve parelel dağıtılmış bir hafızaya sahip olma, bu sistemlerin ana özelliklerinin başında gelir. Bu teknoloji insana benzer yaklaşımlarla robotların oluşturulmasında ilk adım olarak görülebilir (Lippmann, 1987).

Yapay sinir ağları olayları öğrenerek karar verme prensibi üzerine kurulmuşlardır. Öğrenme, zeki sistemlerin bilgi yetenek ve tecrübelerini artırma olayı olarak düşünülebilir. Farklı tanımlar yapılmakla beraber en genel şekliyle öğrenme, sistemlerin aynı veya benzeri işleri yaptıklarında, o işi veya işleri bir önceki yapıldığı şeklinden daha verimli ve etkin olarak gerçekleştirecek değişiklikleri oluşturma süreci olarak tanımlanır. Araştırmacıların zeki robotlar veya benzeri nesneler oluşturma yönündeki çalışmalarının önemli bir noktasını da bu

öğrenme sürecinin bilgisayarlaştırılması oluşturur. Dolayısıyla ortaya atılan öğrenme metod ve yöntemlerinin sayısı her geçen gün artmaktadır. Bunun temel nedeni insanoğlunun programlanabilir makinalar yerine, eğitilebilir makinalara sahip olma arzusudur. Yapay sinir ağları insanoğlunun bu merakını giderebilmek için başlatılan çalışmaların ortaya çıkarttığı bir tür bilimsel öğrenme mekanizmalarıdır (Lippmann, 1987). Yapay sinir ağları her geçen gün ilgi odağı olmaktaysa da insan beyninin fonsiyonları ile ilgili çalışmalar pek de yeni sayılmaz. Özellikle bu çalışmalara 1940'larda başlanmış, hem teknolojik yetersizlik ve hemde beynin karmaşık yapısından dolayı çalışmalar yavaş gelişmiştir. Son yıllardaki teknolojik ve nörofizyolojik gelişmeler nedeniyle elde edilen başarılı sonuçlar dikkatleri yeniden yapay sinir ağlarına çevirmiştir. Bu ağların parelel yapıları ve bilgisayarları geleneksel yöntemlerden çok daha farklı kullanarak özellikle seri bilgisayarlarda bilinen yöntemlerle yapılması mümkün olmayan veya çok zor olan birtakım işlevleri rahatlıla yapmaları önemlerini daha da artırmaktadır. Yapay sinir ağlarını daha iyi kavramak için önce biyolojik sinir ağlarının genel olarak bilinmesinde yarar vardır (Öztemel, 1996).

2.4.1. Biyolojik sinir ağları

Bir sinir hücresi sinir ağlarının en temel elemanlarından biri olup sinir sisteminde fonksiyon ve görevlerine göre değişik şekil ve büyüklükte olabilir. Bir hücrenin bir ucunda "**dendrit**" adı verilen ve hücreye diğer hücrelerden veya dış dünyadan bilgiler (sinyaller) getiren bağlantı elemanları, diğer ucunda ise bir life benzer "**axon**" adı verilen ve hücrelerden diğerlerine veya dış dünyaya bilgiler taşıyan bağlantı elemanları vardır. Bu axon daha sonra diğer hücrelerle birleşme esnasında dağınık dallara ayrılmaktadır. Bu iki uçtaki bağlantı noktalarının elektrofizyolojik olarak hücrelerdeki bilgileri işlemede önemli yeri vardır. Biyolojik bir sinir hücresi şekil 2,11'de verilmiştir.



Şekil 2.11. Biyolojik sinir hücresi

Sinyaller bir hücrenin axonundan diğerinin dentritine gönderilir. Bir axon birden fazla dentrit ile ilişkiye girebilir. Bu bağlantının yapıldığı yere "**synaps**" adı verilir. Hücreler, elektrik sinyallerini hücre duvarlarındaki gerilimi değiştirerek üretirler. Bu ise hücrenin içinde ve dışında bulunan dağılmış iyonlar vasıtasıyla olur. Bu iyonlar sodyum, potasyum, kalsiyum ve klorin gibi iyonlardır. Bir hücre diğer hücreye elektrik sinyalini bu kimyasal iyonlar sayesinde transfer eder. Bazı iyonlar elektrik ve mağnetik kutuplaşmaya neden olurken bazıları kutuplaşmadan kurtulup hücre zarını geçerek iyonların hücreye geçmesini sağlarlar. Sinyallerin bir hücreden diğerine iletilmesini sağlayan bu kutuplaşmanın zayıflamasıdır. Sinyaller hücrenin etkinliğini belirlerler. Bir hücrenin etkinliği hücreye gelen synaps sayısı, synapslardaki iyonların konsantrasyonu ve synaps'ın sahip olduğu güç olmak üzere üç faktöre bağlıdır. Bir hücre sahip olduğu uyarı miktarınca diğer hücreleri etkiler. Bazı hücreler diğerlerinin uyarmalarını pozitif yönde bazı hücrelerde negatif yönde etkiler. İnsan beyni bu şekilde çalışan sayısız hücrenin bir araya gelmesinden oluşur (Öztemel, 1996).

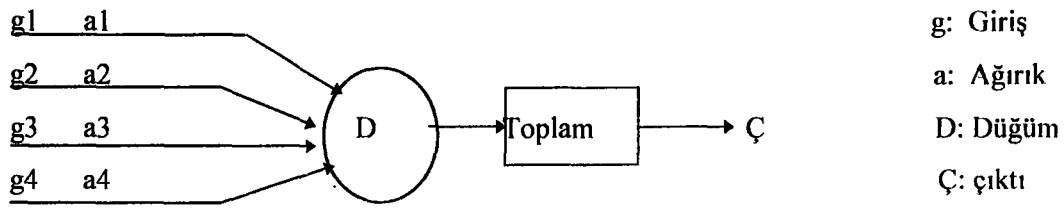
Biyolojik beynin en önemli özelliklerinden birisi de öğrenme olayıdır. İnsanlar ve hayvanlar sürekli olarak içlerinde bulunduğu çevre ile ilişkiler neticesinde bir öğrenme işlemi içerisindedirler. Öğrenilen her yeni bilgi beynin fonksiyonlarını etkileyerek davranışlarda kendini gösterir. Bu özellik yapay sinir ağlarının da temelidir (Öztemel,1996).

Yapay sinir ağları biyolojik sinir ağlarından esinlenerek modellendirilmeye çalışılan daha basit yapıya sahiptirler. Geliştirilen birçok yapay sinir ağı biyolojik sinir ağlarının bilinen

temel birkaç özelliğini (öğrenme kabiliyeti gibi) simüle etmek üzere gerçekleştirilmişlerdir. Bir yapay sinir ağının yapısını belirleyen bazı faktörler vardır. Yapay sinir ağı hücreleri veya işlem elemanları, sinir ağının yapısal topolojisi, ağın sahip olduğu öğrenme kural ve stratejisi bunların başında gelir. Yapay sinir ağları biyolojik sinir ağları gibi hücrelerin veya işlem elemanlarının bir araya gelmesinden oluşur. Her bir işlem elemanı beş elemandan oluşur. Bunlar; girdi, ağırlık, toplama fonksiyonu, çıktı fonksiyonu ve çıktıdır. İşlemci elemana birden fazla girdi gelmekte ve tek bir çıktı oluşturulmaktadır. Girdiler (dendritler) benzer şekilde, diğer hücrelerden bağlantılar vasıtasıyla işlem elemanına bilgi gelmesini sağlamaktadırlar. Bazı durumlarda bir işlem elemanı kendisine de bilgiyi geri gönderebilir. Bahsedilen bu bilgi, elemanlar arasında bulunan bağlantı hatları üzerinde depolanır. Her bağlantının bir ağırlığı vardır. Bu ağırlık bir işlem elemanının diğeri üzerindeki etkisini gösterir. Ağırlık büyüdükçe etki de büyür. Ağırlığın sıfır olması hiçbir etkinin olmaması, negatif olması ise etkinin ters yönde olması demektir. Bu ağırlıklar sabit olabildikleri gibi değişken de olabilirler. Toplama fonksiyonu bir işlem elemanına gelen net girdiyi hesaplayan bir fonksiyondur. Net girdi genellikle girdilerin ilgili bağlantıların ağırlıkları ile çarpılıp toplanması ile belirlenir. Çıktı fonksiyonu da toplama fonksiyonu tarafından belirlenen net girdiyi alarak işlem elemanının çktısını belirleyen fonksiyondur. Genel olarak türevi alınabilen bir fonksiyon olması tercih edilir. Toplama ve çıktı fonksiyonları ilgili probleme bağlı olarak farklı şekiller alabilirler. İşlem elemanının çıktı birimi, çıktı fonksiyonunun ürettiği uyarıyı işlem elemanlarına veya dış dünyaya aktarma işlevini yapar. İşlem elemanları ağın topolojik yapısına göre tamamen birbirinden bağımsız ve parelel olarak çalışabilirler (Simpson, 1990)

Yapay sinir ağlarının fonksiyonlarını gerçekleştirmede sahip oldukları fiziksel yapının önemi büyüktür. Bugün birçok model mevcut olmakla birlikte bu sayı her geçen gün artmaktadır. Farklı yapılaşma, işlem elemanlarının birbirleri ile olan bağlantılarından ve uygulanan öğrenme kuralından kaynaklanmaktadır. İşlem elemanları ya tamamen birbirleri ile bağlantılı veya yerel olarak gruplar halinde bağlantılı olabildikleri gibi, dağınık şekilde de birbirleri ile bağlanabilmektedirler. Bilgi akışı bu bağlantılar üzerinde tek yönlü olduğu gibi çift yönlü de olabilir (Simpson, 1990). Bir grup işlem elemanı bir araya gelerek bir katman (layer) oluştururlar. Genel itibariyle yapay sinir ağlarında üç tür katman bulunur. Sinir ağının dış dünya ile bağlantısını kuran girdi katmanı, gelen sinyalleri işleme kabiliyetine sahip ara katmanlar ve sinir ağının kararlarını dış dünyaya aktaran çıkış katmanı vardır. Girdi katmanında çoğu zaman bilgi işleme sözkonusu olmaz. Bu katmandaki işlem elemanları aldıkları bilgiyi herhangi bir değişikliğe uğratmadan ara katmandaki işlem elemanlarına

aktarırlar. Burada sözü geçen bilgi sinir ağının işlem elemanları arasındaki bağlantı hatları üzerindeki ağırlıklarla gösterilir. Dolayısıyle bilgi bütün ağa dağıtılmıştır.



Şekil 2.12. Yapay sinir hücresinin şematik gösterimi

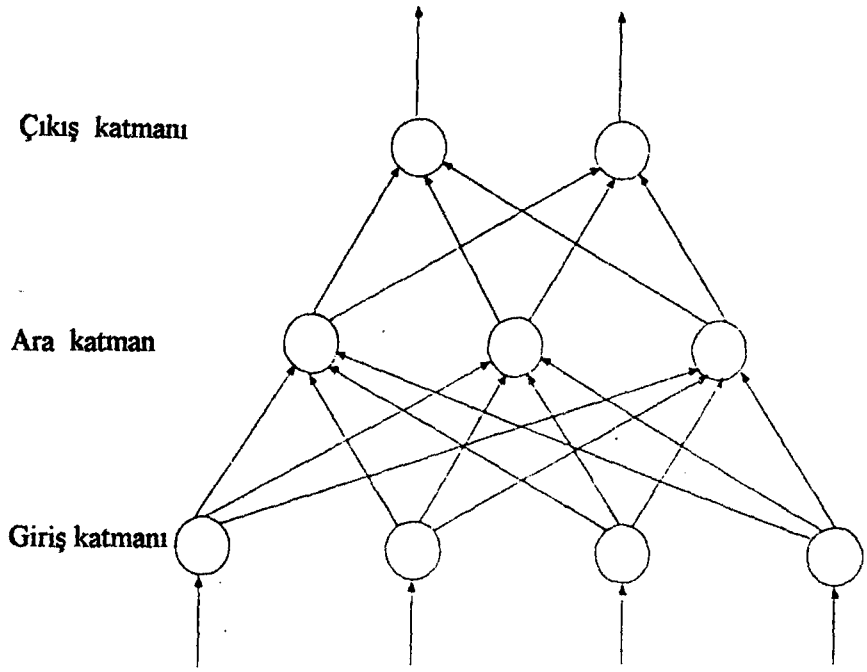
İşlem elemanlarının birbirleri ile ilişkileri ve katmanlar arası ilişkiler değişik yapısal modellerin oluşmasına neden olmaktadır. Şekil 2,13'de tek yönlü bağlantıların olduğu, bir katmandaki bilgilerin sadece bir üst katmana geçtiği ve yine bir katmandaki işlem elemanlarının tamamen bir üst katmana bağlandıkları bir ağ topolojisi görülmektedir. İçbağlantı sinir ağı mimarisi tanımının bir parçasıdır. Bunlar bir sinyali bir birimden diğerine veya birimin kendisine yansıtırlar. Bu iç bağlantı tek yönlüdür ve herbir bağlantıda işaretli bir ağırlık değeri vardır. Bu ağırlık değerleri ağların hafızasını oluşturur. Sinir ağlarında genellikle pozitif sinyaller birimin giriş değerini artırır. Negatif sinyaller birimin giriş değerini azaltır. Birimleri bir ağ içerisinde birbirine bağlayan üç değişik bağlantı şemaları vardır (Öztemel, 1996).

a) Aynı katmandaki birimleri biribiriyle bağlayan bağlar.

b)Farklı katmandaki birimleri bağlayan bağlar.

c)Birimin kendisine geri bağlanan ve dönen bağlantılar.

Eğer bu bilgi bir yönde akarsa bu bağlantılara ileri besleme denir. Geribesleme, bilginin hem ileri hemde geri yönde işlem elemanları boyunca akmasına izin verir.



Şekil 2.13. Yapay sinir ağı mimarisi

Bir yapay sinir ağının sahip olduğu bilgi, işlem elemanları arasındaki bağlantı hatları üzerinde saklanır ve ağırlıklar vasıtasıyle gösterilir. Ağ, olaylar hakkında girdi ve çıktılar arasındaki ilişkiyi daha önce tanımlanan örneklerden genellemeler yaparak öğrenir ve bu genelleme ile yeni oluşan veya ortaya çıkan daha önce hiç görülmemiş olaylar hakkında karar verebilir. Ağa bir örnek olay gösterildiğinde girdi katmanından alınıp ara katmanlarda işletilerek ağın, o olay hakkında ürettiği sonuç, çıktı katmanından sunulur. Bu bilgi işleme, ağın sahip olduğu tecrübeye göre bilginin ara katmanlarda çağrıştırılması ile gerçekleştirilir. Bu çağrıştırma olayı modelden modele değişmektedir. Örneğin ara katmanlardaki işlem elemanları sahip oldukları bağlantı ağırlıkları ile kendi kararlarını üretir ve çıktı katmanındaki işlem elemanlarına gönderirler. Çıktı katmanındaki işlem elemanları da yine ilgili ağırlıkları kullanarak ağın en son kararını oluşturular. Bu ağırlıklar tıpkı ilgili olayın belirli özelliklerini hafızada saklayan elemanlar gibi düşünülebilirler. Bilgi işleme ise bir olay gösterildiğinde hafızadan ilgili özellikleri çağırmak ve bunlar ile ilgili girdileri birlikte analiz ederek karar vermek şeklinde yorumlanabilir (Hopfield and Tank, 1985).

Ağın zeki davranış gösterebilmesi (öğrenme) için sahip olduğu bütün ağırlıkların ilgili problemde öğrenilmesi istenen özellikleri genelleştirecek şekilde doğru değerlere sahip olması gerekir. Bu doğruluk ne kadar artarsa genelleme kabiliyeti dolayısıyle zeki davranışı o denli artar. Optimum ağırlık değerleri bir öğrenme kuralına göre tanımlanır. Çoğunlukla bağlantılara

başlangıç değerleri olarak rastgele ağırlıklar atanır ve bu ağırlıklar eldeki örnekler incelendikce bir kurala göre değiştirilerek optimum ağırlık değerleri bulunmaya çalışılır Kısaca belirtmek gerekirse, öğrenme kuralları bir işlem elemanının örnekleri gördükçe kazandığı tecrübeye göre ilgili bağlantı ağırlıklarını nasıl değiştireceğini belirleyen algoritmalardır (Hopfield and Tank, 1985).

Yapay sinir ağlarının hem yaygın kullanılmasını sağlayan hem de geleneksel bilgi işleme metodlarından ayrılan özellikleri vardır. Yapay sinir ağlarının bir takım özellikleri kullanılan sinir ağı modeline bağlı ise de bunun yanında birtakım genel özellikleri de vardır. Bu özellikler:

a)Yapay sinir ağları, olaylar arasındaki ilişkileri belirli bir algoritmaya dayanarak çözmek yerine o ilişkiyi gösteren örnekleri incelemek suretiyle çözümler üretmeyi sağlarlar. Olay ile ilgili sinir ağına örneklerden başka hiçbir önbilginin verilmemiş olması önemlidir. Ağ kendisine tanımlanan örnekleri tekrar tekrar inceleyerek ağdaki ilişkiyi kavramaya çalışır. Her yeni örnek ağın sahip olduğu bilgiye bir yenisini ekler ve bu işlem tekrar ettikçe ilgili problem hakkında genellemeler yapılır (Öztemel, 1996).

b)Yapay sinir ağları kendisine tanımlanan bir şekli, daha önce öğrendikleri ile mukayese ederek aradaki banzerlikleri ortaya koyma ve eksik şekilleri tamamlama, benzer şekilleri oluşturma veya şekilleri belirli sınıflara ayırma özelliklerine sahiptir.

c)Bir ağ öğrenme esnasında sahip olduğu bilgileri temsil etme şeklini kendisi belirleyebilir. Bu daha çok kodlanması zor veya mümkün olmayan olayların üzerindeki çalışmalarda önemlidir. Bu özellikleri neticesinde sinir ağları, kendilerine sunulan örneklerden genelleme yapabilirler. Benzeri olayları değerlendirmede de bu genellemeden yararlanırlar. Eksik, gürültülü, doğruluğu belli olmayan olaylarda bu genelleme özelliği oldukça faydalıdır. Genelleme sonunda eksik bilgiler tamamlanabilir, gürültülü bilgiler süzülerek ayrıştırılabilir, özellikle görüntü tanıma, sınıflandırma ve sinyal analizinde kullanılabilir.

d)Verilerde bir eksik sözkonusu olursa geleneksel yöntemler çalışmazlar. İyi eğitilmiş genelleme kapasitesi yüksek bir sinir ağı kendisine tanımlanan veriler eksik olsa da karar verme işlemine devam edebilir. Aynı şekilde sinir ağı üzerinde bir takım problemler ve bozukluklar da olabilir. Geleneksel sistemlerin tersine sinir ağları bu durumda da çalışmalarına devam ederler. Verilerdeki eksiklik veya sinir ağlarındaki yapısal bozukluk arttıkça sinir ağının performansı yavaş yavaş azalmaya başlar. Fakat sistem fonksiyonunu tamamen durdurmaz. Her durumda bir sonuç üretilebilir. Bu özellik sinir ağının yapısından kaynaklanmaktadır. Çünkü ağın sahip olduğu bilgi, ağ üzerindeki hücrelerin birbiri ile olan bağlantıları üzerine

dağıtılmıştır. Böyle bir durumda tek bir bağlantı ve onun üzerindeki bilgi başlıbaşına hiçbir anlam ifade etmez. Ancak bir grup halinde veya tam olarak bağlantıların birlikte düşünülmesi sonucu anlamlı bilgiler üretilir.

Bilgi işleme hızı bilgisayar teknolojisinde halen önemli bir etkendir. Sistemlerin her geçen gün biraz daha karmaşık olması nedeniyle daha çok bilgiyi daha verimli bir şekilde işleme gerekliliği yeni yazılım/donanım sistemlerini zorunlu hale getirmiştir. Halbuki insan beyni, oldukça fazla bilgiyi gerçek zamanlı olarak oldukça hızlı bir şekilde işleyebilmektedir. Bu durum, yapısındaki hücrelerin parelel olarak çalışması ile açıklanmaktadır. Yapay sinir ağları da yine birbirlerine bağlı ve parelel işlem elemanlarından oluştuğundan hızlı işleyebilmeleri bu ağlara özellikle endüstride gerçek zamanlı çalışma kabiliyeti de kazandırır. Yapay sinir ağları öğrenme durumlarına göre genel olarak üç türlü öğrenme algoritmalarına sahiptirler (Öztemel, 1996):

a) Öğretmenli öğrenme: Bu öğrenme türünde, dışarıdan bir öğretmenin sinir ağının öğrenmesine müdahalesi sözkonusudur. Öğretmen, sinir ağının ilgili girdi için üretmesi gereken sonucu sinir ağı sistemine tanımlar. Diğer bir ifade ile ağa girdi-çıktı ikilisinden oluşan örnekler sunulur ve girdi-çıktı bilgisinin ağa tanımlanması gereklidir. Ağ, girdi kısmını alır ve o anki bağlantı ağırlıklarının tanımladığı bilgi ile bir çıktı oluşturur. Bu çıktı, hedef çıktı ile karşılaştırılır ve ağdaki hata tekrar ağa aktarılarak ağırlıklar bu hatayı azaltacak şekilde değiştirilirler.

b) Takviyeli öğrenme: Bu tür öğrenmede yine bir öğretmene ihtiyaç vardır. Öğretmenli öğrenmeden farkı ise bu durumda öğretmenin ağın üretmesi gereken sonuç yerine, onun ürettiği sonucun sadece doğru veya yanlış olduğunu söylemesidir. Bu ise ağa bir takviye sinyalinin gönderilmesi ile sağlanır. Bu tür öğrenme örnek için beklenen çıktının oluşturulamadığı durumlarda çok faydalıdır.

c) Öğretmensiz Öğrenme: Bu durumda hiçbir öğretmene ihtiyaç yoktur. Onun için buna çoğu zaman kendi kendine organize öğrenme (self-organized learning) de denilmektedir. Ağ kendisine gösterilen örnekleri alır ve belli bir kritere göre sınıflandırır. Bu kriter önceden bilinmeyebilir. Ağ kendi öğrenme kriterlerini kendisi tanımlamaktadır.

2.5. Sinir Ağı Yaklaşımları İle Görüntü Sıkıştırılması

Yapay bilgi işleme sistemleri için seri ve parelel bilgi işleme olmak üzere iki yaklaşım sözkonusudur. En yaygın kullanılan seri bilgi işlemeli olanıdır. Seri hesaplamalar yapıları gereği matematik formüller ve algoritmalar için uygundur. Son çalışmalarda parelel bilgi işleme sistemlerine yoğun ilgi vardır (Dony and Haykin, 1995). Bunlar yapay sinir ağlarından oluşan, parelel ve birbirine bağlılık içinde işlem yapan işlem elemanlarının ağlarıdır.

Yapay ve gerçek parelel işlemci sistemleri arasındaki ilişki sinir ağlarının özelliklerine benzetilmiştir. Gerçek hayattaki bazı sistemlerin performansı yapay sistemlerle doğru olarak modellenebilmiştir. Yapay sinir ağlarındaki mimari karakterisrik normal sistem ile uyum içerisindedir. Görüntü işleme uygulamaları sinir ağlarının parelel işleme modelinin çok etkili olduğunu göstermiştir.

Genellikle yapay ağ, karmaşık hesaplamaları çözmek için birbirleriyle bağlanmış çok sayıda hesaplama birimi ile çalışan bir hesaplama paradigması olarak tanımlanabilir. Böyle bir hesaplama modeli ile karmaşık nörobiyolojik sistemlerin fonksiyonu arasında bir benzerlik vardır. Yüksek düzey nörobiyolojik sistemler, örneğin insan beyni, karmaşık işlemleri nöron ağları kullanarak ve her nöron basit bir işlem gerçekleştirerek yerine getirir. Seri modelde program adımları veya durumlar, işlemci için tanımlanır. Bu adımlar olası tüm giriş adımlarının nedenlerini açıklamalıdır, fakat sinir ağları bilgi örnekleri kullanarak eğitilir. Eğitim esnasında ağ yeterince işlenmiş bilgiyi iç işaretleri düzenleyerek sağlar. Pek çok uygulamalarda bu ağlar seri modelden daha çok kullanışlı olabilirler. Çünkü paralel yapılarından dolayı, seri bilgi işleme sistemlerinin performansını sınırlayan hesaplamalar olabilir. Fakat sinir ağları örnek bilgi kullanarak eğitildiklerinden dolayı yeni bilgi işleme esnasında sistem, yeni değişikliklere uyum sağlayabilir. Eğitimin bir diğer avantajı, bilgi bireysel olarak işaretlendiği için başlangıçta eğitim kümesini depolamaya gerek yoktur. Bu durum özellikle çok geniş bilgi kümesi olduğu durumlarda önemlidir (Dony and Haykin).

2.5.1. Yapay sinir ağı kullanarak öngörüşlü kodlama

Çok katmanlı bir ağ doğrusal olmayan öngörücü olarak kullanılabilir. Giriş, olası bilgileri içerir. Giriş ve çıkışlar arasında bir veya daha fazla ara katman kullanılabilir. Ağın doğrusal olmayan yapısı gereği ağın öngörme hatasının değişimi doğrusal öngörücünün değişiminden daha düşük olabilir. Bu durumda DPCM sistem için tahmin, kazancın yükselmesini sağlar. Üç katmanlı bir ağ kullanılarak sinyal gürültü oranı (signal noise ratio) (S.N.R.) düşürülebilir (Li and Manikopoulos, 1990).

2.5.2. Yapay sinir ağı kullanarak vektör gruplama

Yapay sinir ağı kullanılarak vektör gruplamasında S.O.F.M algoritması, Adres öngörmeli V.Q,Finitestate V.Q olmak üzere üç önemli algoritmadan bahsedilebilir.

S.O.F.M algoritması, Khonen'in S.O.F.M algoritması V.Q' da kod kitabı tasarımı için matematik modellerin kullanılması fikrine dayanır. Kohonen'in S.O.F.M algoritması ile, pekçok gruplama algoritmalarında olduğu gibi her giriş vektörü sınıflandırılır ve "winning" sınıf her iterasyonda değiştirilir. S.O.F.M algoritmasında giriş vektörü sadece kazanç sınıfı için olmaksızın güncelleştirilir (Khonen, 1990).

Adres Öngörmeli V.Q, kod kitabı dizaynında S.O.F.M algoritmaları kullanımının diğer faydası A.P.V.Q denilen algoritmanın kullanılmasıdır. Bu teknik ilişkili bazı komşu girişlerde düzenli kod kitabı kullanır. Bitişik kod kelimeler arasındaki ilişki; giriş sinyalinin kod kelime adresi olduğu D.P.C.M kodlayıcı yapımında kullanılabilir. Bu teknik kayıplı adres kodlamada kullanılabilir. S.O.F.M algoritması A.P.V.Q ile bağlantı içinde bir kod kitabı hesaplamak için kullanıldığında, standart V.Q'ye göre daha iyi sonuç verir.

Finit state V.Q' da kod kelime indexi $\dot{I}_n$, kaynak Sn'in durum kod kitabından seçilir. Kaynak durum, önceki durum S_{n-1} in ve önceki index $\dot{I}_{n-1}$ in bir fonksiyonudur. Eğer durum geçiş fonksiyonu gelecek girişin iyi bir öngörücüsü ise her durum kod kitabı, bir belleksiz gruplayıcı için gerekli alandan daha az alan kullanır (Dony and Haykin, 1995).

2.6. Yapay Sinir Ağlarında Geriyansıma (Backpropagation of NN)

Bilgisayar sistemleri yüksek hızlarda mantık ve matematik problemlerini çözmek için tasarımlanmışlardır. Kullanılan bilgisayarlar bir zaman aralığında sadece bir işlem yapabilmektedirler. Bu durum onların ardışıl yapılarından kaynaklanmaktadır. İşlem hızı da tipik olarak μs mertebesindedir. Bu işlem süresi bile büyük yazılımların çalıştırılmasında problemler oluşturabilmektetedir. Bu problemin çözüm yollarından birisi, yeni bir işleme sistemi olan geriyansımalı ağ sistemidir. Bu sistemde bilgiler parelel olarak işlenir, verilen örnek tanımlamalar arasındaki ilişki kendi kendine öğretme suretiyle yeni duruma kendini uydurabilirler (Freeman and Skapura, 1992).

Geriyansımalı ağ iki aşamalı geriyansıma çevrimi kullanarak önceden tanımlı giriş-çıkış örnek çiftlerini öğrenir. Bir uyarıcı olarak ilk katmana giriş uygulandıktan sonra çıkış üretilinceye kadar herbir katmandan yansıtılır. Bu çıkış, hedef çıkış ile karşılaştırılır ve herbir çıkış birim için bir hata sinyali üretilir. Hata sinyalleri, daha sonra çıkış katmanından geriye doğru, çıkışa katkıda bulunan herbir ara katmanlardaki düğümlere iletilir. Bununla birlikte ara katmandaki herbir birim, toplam hatanın bir kısmını alır. Bu işlem toplam hata minimum seviyeye ininceye dek katmanlarda tekrarlanır. Hata sinyaline bağlı olarak bağlantı ağırlıkları her birim ile güncelleştirilir. Farklı girişlerin tümünün anlaşılması için farklı düğümler eğitilerek ara katmandaki bu düğümler kendi kendilerine eğitilirler. Geriyansıma ağının sade mimarisi üç katmandan oluşur. Bu katmanlar ileri beslemeli işlemci ile bağlıdır. Her nekadar bu mimari yaygın isede üç katmandan daha fazla olabilir. (Freeman and Skapura, 1992).

Sinyal, ağdaki farklı katmanlar arasında yayılırken her üst katmanda hazırlanan sonuç bir sonraki katmandaki birimlerce bir giriş olarak algılanır. Geriyansımalı ağlarda hata tekrarlanarak minimuma indirilmesinde "Delta kuralı" denilen ağırlıkların belirli tanımlamalarla güncelleştirilmesi yönteminden yararlanılır.

2.6.1. Genellenmiş delta kuralı

Genellenmiş delta kuralı, geriyansımalı ağ eğitilirken, hedef değerler ile çıkış değerler arasında meydana gelen hatanın bir kısmı, her bir eğitim hücresine, geri yansıtılarak ağırlıkların hataya göre değiştirilmesini ve bu işlemin belli sayıda tekrarlanarak hatanın en küçük değere ulaşmasını sağlayan bir yöntemdir. Bu kuralın adımları şu şekilde sıralanabilir:

1) Ağa bir giriş vektörü uygulanır ve ilişkili çkış değerleri hesaplanır.

2) Bulunan çıkışlar ile doğru çıkışlar karşılaştırılır ve bir hata değeri belirlenir.

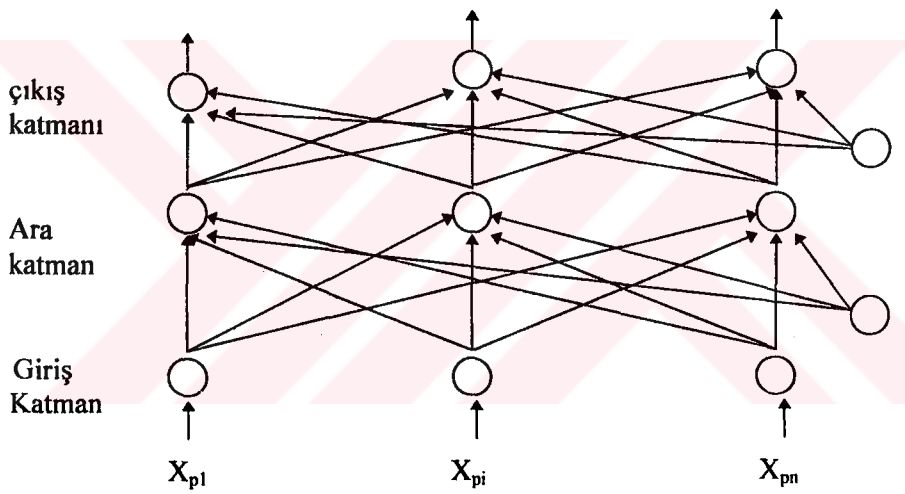
3) Hatanın azaltılması için her bir ağırlığın "+" veya "-" yönü belirlenir.

4) Her bir ağırlığı değiştirmek için genlik belirlenir.

5) Ağırlıklara düzeltmeler uygulanır.

6) 1. adımdan 5. adıma kadar olan adımlar eğitim setindeki tüm vektörler için hata istenen bir değere ulaşıncaya kadar tekrarlanır (Freeman and Skapura, 1992).

Bu adımların matematiksel ifadelerini şekil 2.14' te verilen üç katmanlı algoritma temel alınarak incelenebilir.



Şekil 2.14. Üç Katmanlı bir BPN mimarisi

Giriş vektörü $X_p = (x_{p1}, x_{p2}, \ldots x_{pn})$ ağın giriş katmanına uygulandığında, giriş birimleri bu değerleri ara katman birimlerine yayarlar. J'inci ara birim için ağ giriş:

$$net^h_{pj} = \sum_{i=1}^{N} w^h_{ji} x_{pi} + \theta^h_j \qquad (2.9)$$

Eşitlik (2.9)'de w^h_{ji} i'inci giriş biriminden j'inci arabirime olan bağlantı üzerindeki ağırlıktır. Eşitlik (2.9)'da ara katman üzerindeki nicelikler h ile tanımlanmıştır. Bu birimin aktivasyonu:

$$i_{pj} = f_j^h(net_{pj}^h) \tag{2.10}$$

olduğuna göre çıkış birimler için eşitlik ;

$$net_{pk}^o = \sum_{j=1}^{L} w_{kj}^o i_{pj} + \theta_k^o \tag{2.11}$$

$$O_{pk} = f_k^o(net_{pk}^o) \tag{2.12}$$

yazılabilir. Bu durumda ağırlık değişimi;

$$w(t+1) = w(t)_i + 2\,\mu\,\varepsilon_k\,x_{ki} \tag{2.13}$$

olarak verilir. μ pozitif sabit, x_{ki} k'ıncı eğitim vektörünün i'inci bileşeni ve ε_k çıkışın doğru değeri ile o anki çıkış arasındaki farktır:

$$\varepsilon_k = (d_k - y_k) \tag{2.14}$$

Eşitlik (2.14)'te istenilen çıkış d_k ve hesaplanan çıkış y_k dır. Fakat bu tanım BPN için yeterli olmaz. Daha genel bir ifade ile hata :

$$\delta_{pk} = (y_{pk} - o_{pk}) \tag{2.15}$$

ifade edilebilir. Burada "p", p'inci eğitim vektörünü ve "k", k'ıncı çıkış birimini tanımlar. Bu durumda y_{pk} istenilen çıkış değeri, o_{pk} k'ıncı birimde o anda oluşturulan çıkıştır. Bu hata tüm çıkış birimlerinin hatalarının karesel toplamı ile en aza indirilebilir.

$$E_p = \tfrac{1}{2} \sum_{k=1}^{M} \delta_{pk}^2 \tag{2.16}$$

eşitlik (2.16) daki ½ katsayısı sonraki türev hesaplamalarındaki kolaylık için vardır.

Ağırlık değişimindeki yönü belirlemek için E_p nin gradyantının negatifi, ∇E_p, hesaplanır. Toplam hatayı düşürerek ağırlıkların değerlerini ayarlayabiliriz. ∇E_p nin herbir bileşenlerini eşitlik (2.17)'den hesaplanır. δ_{pk} nın ifadesi:

$$E_p = \tfrac{1}{2}\sum_k (y_{pk} - o_{pk})^2 \tag{2.17}$$

$$\frac{\partial E_p}{\partial w^o_{kj}} = -(y_{pk} - o_{pk})\frac{\partial f^o_k}{\partial (net^o_{pk})}\frac{\partial (net^o_{pk})}{\partial w^o_{kj}} \tag{2.18}$$

olarak hesaplanır. o_{pk}, çıkış değeridir. f^0_k türeviyle uğraşmak yerine;

$$\frac{\partial (net^o_{pk})}{\partial w^o_{kj}} = \left(\frac{\partial}{\partial w^o_{kj}}\sum_{j=1}^{L} w^o_{kj} i_{pj} + \theta^o_k\right) = i_{pj} \tag{2.19}$$

$$-\frac{\partial E_p}{\partial w^o_{kj}} = (y_{pk} - o_{pk}) f^{o'}_k (net^o_{pk}) i_{pj} \tag{2.20}$$

yazılabilir. Görüldüğü gibi ağırlıkların değişiminin büyüklüğü negatif gradiyent ile orantılıdır. Böylece çıkış katmandaki ağırlıklar, eşitlik (2.21) ifadesi ile güncelleştirilirler.

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \Delta_p w^o_{kj}(t) \tag{2.21}$$

Eşitlik (2.22) de:

$$\Delta_p w^o_{kj}(t) = \eta (y_{pk} - o_{pk}) f^{o'}_k (net^o_{pk}) i_{pj} \tag{2.22}$$

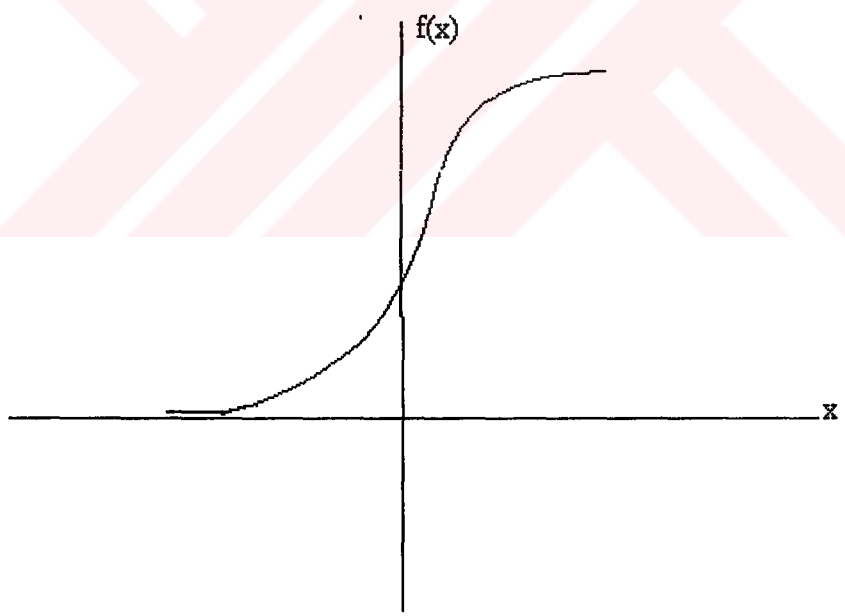
η faktörü öğrenme oran (learning rate) parametresi olarak tanımlanır. Bu değer genellikle bir'den küçük pozitif bir değerdir (Freeman and Skapura, 1992).

Transfer Fonksiyonu, her işlem elemanı için giriş sinyallerini bir çıkış sinyale dönüştüren aktarıcıdır. İlk adım bir işlem elemanı için net giriş değeri oluşturmaktır. Bazı

girişler diğerlerinden daha fazla önemli olabilirler, bu nedenle mevcut her bir giriş ile işlem elemanını ilişkilendiren bir ağırlık vardır. Bu ağırlıklar işlem elemanları arasındaki bağlantı genliğidir ve eşitlik (2.23)'de olduğu gibi gösterilebilir.

$$W_i=(w_{i1},.....w_{in}) \tag{2.23}$$

Birim tüm giriş sinyallerini aldıktan sonra toplam giriş bu ağırlıklara göre hesaplanır. Yaygın olarak kullanılan metod; toplama fonksiyonu kullanmaktır. Daha sonra herbir işlem elemanı için tanımlanan işlev içinden tanım değerleri elde edilir. Bu tanımlanan işlev basit bir doğrusal olmayan fonksiyon veya fazla karmaşık -örneğin bir sigmoid fonksiyonu-olabilir. Tanımlanan işlev sürekli ve doğrusal değildir. Yaygın olarak Şekil 2,15'deki sigmoid fonksiyonu kullanılabilir.



Şekil 2.15. Sigmoid fonksiyonunun gösterimi

Sigmoid fonksiyonu her birim için çıkış değerini hesaplamada kullanılır. Bu fonksiyon matematiksel olarak eşitlik (2.24)'de, türevi ise eşitlik (2.25)'te ve eşitlik (2.26)'da gösterilmiştir (Lippmann, 1987).

$$F(x)=1/(1+e^{-x}) \tag{2.24}$$

$$\frac{\partial F(x)}{\partial (x)} = \left(\frac{1}{1+e^{-x}}\right)\left(1-\frac{1}{1+e^{-x}}\right) \tag{2.25}$$

$$F'(x) = F(x)(1-F(x)) \tag{2.26}$$

Çıkış değerinin güncelleştirilmiş değerler;

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta(y_{pk} - o_{pk})o_{pk}(1-o_{pk})i_{pj} \tag{2.27}$$

ifade edilebilir.

$$\delta_{pk}^{o} = (y_{pk} - o_{pk})f_{k}^{o'}(net_{pk}^{o}) \tag{2.28}$$

$$= \delta_{pk} f_{k}^{o'}(net_{pk}^{o}) \tag{2.29}$$

olduğundan ağırlık güncelleştirme eşitliği;

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta\delta_{pk}^{o} i_{pj} \tag{2.30}$$

olarak tanımlanabilir (Lippmann, 1987). En küçük kareler tekniği ile burada tanımlı gradiyent azalım metodu arasındaki ilişkiyi yorumlamak gerekirse, bu metodlar birbirine benzer olabilmesi için, ağırlık değişimlerinin bütün eğitim tanımlamaları elde edilinceye kadar işlemin tekrarlanması gerekir. İşlenen her bir tanım işlendiğinde, değişiklikler yığılır, toplanır, ağırlıklar için güncelleştirme yapılır. Bu işleme hata, istenilen bir değere ulaşıncaya kadar devam edilir. Bu işlemin hatası :

$$E = \sum_{p=1}^{P} E_{p} \tag{2.31}$$

p, eğitim setindeki tanımlanma numarasıdır. Çıkış katman için yapılan hesaplamalar, ara katman için tekrarlanabilir. Ara katman birimlerinin çıkışlarının hatası ölçülmek istendiğinde bir problem ortaya çıkar. Hesaplanan çıkışlar bilinmekte fakat bu birimler için doğru çıkışlar bilinmemektedir. Toplam hata ara katmanlardaki çıkış değerleriyle ilişkili olmalıdır. Eşitlik (2.17) ye geri dönüldüğünde:

$$E_p = \tfrac{1}{2}\sum_k (y_{pk} - o_{pk})^2 \tag{2.32}$$

$$= \tfrac{1}{2}\sum_k (y_{pk} - f_k^o(net_{pk}^o))^2 \tag{2.33}$$

$$= \tfrac{1}{2}\sum_k (y_{pk} - f_k^o(\sum_j w_{kj}^o i_{pj} + \theta_k^o))^2 \tag{2.34}$$

Eşitlik (2.34)'deki i_{pj} ara katmandaki ağırlıklara bağlı olduğu bilindiğine göre, ara katman ağırlıklarının güncelleştirilmesi E_p nin gradiyentinin hesaplanmasıyla açıklanabilir:

$$\frac{\partial E_p}{\partial w_{ji}^h} = \tfrac{1}{2}\sum_k \frac{\partial}{\partial w_{ji}^h}(y_{pk} - o_{pk})^2 \tag{2.35}$$

$$= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial (net_{pk}^o)} \frac{\partial (net_{pk}^o)}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial (net_{pj}^h)} \frac{\partial (net_{pj}^h)}{\partial w_{ji}^h} \tag{2.36}$$

eşitlik (2.36)' daki faktörler önceki eşitliklerden hesaplandığında sonuç:

$$\Delta_p w_{ji}^h = \eta f_j^{h'}(net_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^o) w_{kj}^o \tag{2.37}$$

Eşitlik (2.37)'deki η öğrenme parametresidir. Hata şu şekilde tanımlanabilir:

$$\Delta_p w_{ji}^h = \eta f_j^{h'}(net_{pj}^h) x_{pi} \sum_k \delta_{pk}^o w_{kj}^o \tag{2.38}$$

Ara katman üzerindeki bütün ağırlık değişimleri çıkış katmanlarındaki hata terimlerine bağlıdır. Bu sonuç geriyansıma düşüncesidir. Çıkış katman üzerindeki bilinen hatalar uygun ağırlık değişimlerini belirlemek için ara katmanlara geri yansıtılır. Ara katman hata terimi:

$$\delta_{pj}^{h} = f_j^{h'}(net_{pj}^{h})\sum_k \delta_{pk}^{o} w_{kj}^{o} \tag{2.39}$$

Çıkış katman için yazılan eşitliğe benzer ağırlık güncelleştirme eşitliği:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta \delta_{pj}^{h} x_i \tag{2.40}$$

olarak verilebilir. Genelleştirilmiş delta kuralını özetlenirse;

1) Giriş birimlere $X_p = (x_{p1}, x_{p2}, \ldots\ldots, x_{pN})$ giriş vektörü uygulanır.

$$net_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_j^{h} \tag{2.41}$$

2) Net-giriş değerleri ara katman birimleri için uygulanır.

$$i_{pj} = f_j^{h}(net_{pj}^{h}) \tag{2.42}$$

3) Ara katmandan çıkışlar hesaplanır.

4)Çıkıştaki her bir birim için ağ girişi hesaplanır.

$$net_{pk}^{o} = \sum_{J=1}^{L} w_{kj}^{o} i_{pj} + \theta_k^{o} \tag{2.43}$$

5)Çıkış birimlerinin çıkışları hesaplanır.

$$o_{pk} = f_k^{o}(net_{pk}^{o}) \tag{2.44}$$

6)Çıkış birimleri için hatalar hesaplanır.

$$\delta_{pk}^{o} = (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^{o}) \tag{2.45}$$

7)Ara birimler için hata değerleri hesaplanır. Hesaplama yapılırken çıkış katmandaki bağlantı ağırlıkları güncelleştirilmeden önce yapılmalıdır.

$$\delta_{pj}^{h} = f_{j}^{h'}(net_{pj}^{h})\sum_{k}\delta_{pk}^{o}w_{kj}^{o} \tag{2.46}$$

8) Çıkış katman üzerindeki ağırlıklar güncelleştirilir.

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta\delta_{pk}^{o}i_{pj} \tag{2.47}$$

9) Ara katman üzerindeki ağırlıklar güncelleştirilir.

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta\delta_{pj}^{o}x_{i} \tag{2.48}$$

Ağı eğitmek için olabildiğince fazla bilgi bulunur. Mevcut eğitim bilgisinden küçük bir alt küme ağ eğitimi için gerekir. Geri kalan bilgi ağın giriş vektörleri için istenilen haritayı doğru olarak elde edip etmediğini görmek için yapılan test işleminde kullanılır. Eğer ağ bir çevre gürültüsü içerisinde eğitiliyorsa örneğin piksel-görüntü- ASCII durumunda olduğu gibi, bilgi kümesinde gürültü giriş vektörleri bulunur. B.P.N genellemede iyi bir performans sağlar. Genelleme ile kastedilen birkaç farklı vektör verildiğinde bunlara ait önemli benzerlikleri öğrenmesidir. Örneğin 5 karakter uzunluklu bir sayının tek mi çiftmi olduğunu belirlemek için bir ağ eğitilirse örneklere ait küçük bir küme eğitimde kullanılmak suretiyle B.P.N bu sayı içindeki son önemli bit değerinin temeli üzerine sınıflandırma yaparak, ağırlıklarını buna göre ayarlayacaktır. Eğer fonksiyon olarak sigmoid kullanılırsa çıkışların normalize edilmesi gerekir (Freeman and Skapura, 1992).

2.6.2. Ağ genişliği

Bir problemi çözmek için ne kadar işlem birimine ihtiyaç vardır? Üç katman her zaman için yeterlimidir? Bu tür sorulara kesin cevap verilemez. Genellikle üç katman yeterli olmasına rağmen problemin çözümü için birden çok ara katmana ihtiyaç duyulabilir. Bu durumda ağ eğitimi daha hızlı olur. Giriş katmanın işlemci eleman sayısı uygulamanın

durumuna göre belirlenebilir. Çıkış düğümlerinin sayısı çıkış birimleri üzerindeki bilginin analog veya sayısal olup olmamasına göre karar verilerek belirlenir.

2.6.3. Eğitim parametreleri ve hata

Geri yansımalı ağ eğitilirken, eğitime yarımcı olacak veya hatanın istenilen değere ulaşmasını daha kısa sürede gerçekleştirebilecek eğitim parametrelerinden yararlanılır. Bu parametrelerin en önemlileri; eğitim parametresi, eğilimleme parametresi ve momentum parametresidir. Eğilimleme (bias) terimi, θ, +0.5 ile -0.5 arasında bir değer ile tanımlanabilir. Bu terim eşitlik (2.49)'da gösterilmektedir.

$$net^{o}_{pk} = \sum_{j=1}^{L} w^{o}_{kj} i_{pj} + \theta^{o}_{k} \qquad (2.49)$$

$\theta^{o}_{k} \equiv w^{o}_{k(L+1)}$ ve $i_{p(L+1)} \equiv 1$ olarak tanımlandığında ;

$$net^{o}_{pk} = \sum_{j=1}^{L+1} w^{o}_{kj} i_{pj} \qquad (2.50)$$

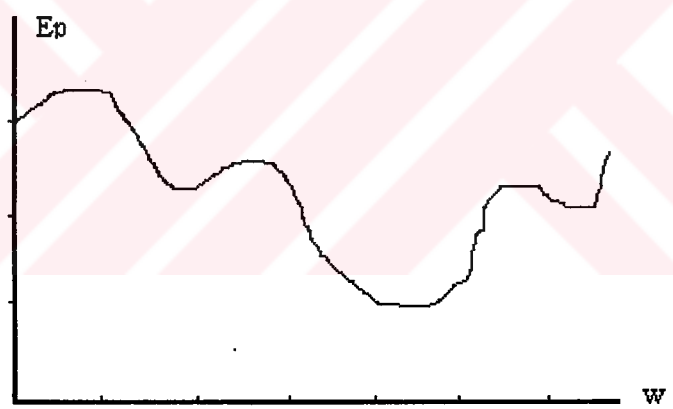
olarak yazılabilir. Eşitlik (2.49)'daki θ^{o}_{k}; her bir işlem elemanı için ilişkili eşik değeridir ve eğitim işleminde bir ağırlık olarak düşünülebilir. Diğer bir deyişle işlem elemanı ağırlıklı ağ giriş eşik değerinden büyükse çıkış sinyali üretecek aksi halde işlemci eleman sinyal üretmeyecektir. Bununla birlikte eğitim işleminde bias terimleri kullanılmayabilir. Eğitim oran parametresi (learning rate parameter), η, ağ performansı üzerinde önemli bir etkiye sahiptir. Genellikle bu paremetre için küçük değerler verilmelidir. Örneğin 0.05 ile 0.25 arasında değerler verilebilir. Öğrenme parametresi ağın güvenli olarak bir problemi çözmesi için gereklidir. η değerinin küçük olarak seçilmesi ağın çok sayıda iterasyon yapması anlamına gelir. Bunun sebebi ağın doğru sonuca yakınsaması küçük adımlar ile gerçekleşmesinden kaynaklanmaktadır. Eğitime başlarken η'nün boyutunu sürekli artırmak mümkündür. Hata değeri düşerken η değerinin artırılması, hatanın hızlı olarak en aza indirilmesine yardım eder. Fakat bu durumun sakıncası; ağ, en küçük değerden çok uzağa atlayabilir. Yakınsamayı hızlandırmak için başka bir yol ağırlık değişim değerini hesaplarken

önceki değişikliklerin küçük bir parçasının eklenmesidir. Bu ekleme işleminde momentum terimi denilen bir parametreden yararlanılır. Bu parametre aynı yöndeki ağırlık değişimlerini korumayı sağlar. Çıkış katmandaki momentum parametreli ağırlık değişim eşitliği :

$$w^{o}_{kj}(t+1) = w^{o}_{kj}(t) + \eta\delta^{o}_{pk} i_{pj} + \alpha\Delta_{p} w^{o}_{kj}(t-1) \qquad (2.51)$$

ile ifade edilir. Eşitlik (2.51)'de α, momentum parametresidir ve genellikle 1 den daha küçük bir değerdir. Eğilimleme teriminde olduğu gibi, momentum terimini de kullanmak isteğe bağlıdır (Freeman and Skapura, 1992).

Bir önemli konu hata-ağırlık uzayında belirli noktalarda, hatanın yerel en küçük değere yakınsama olasılığıdır. Bu durum şekil 2,16'da gösterilmiştir.

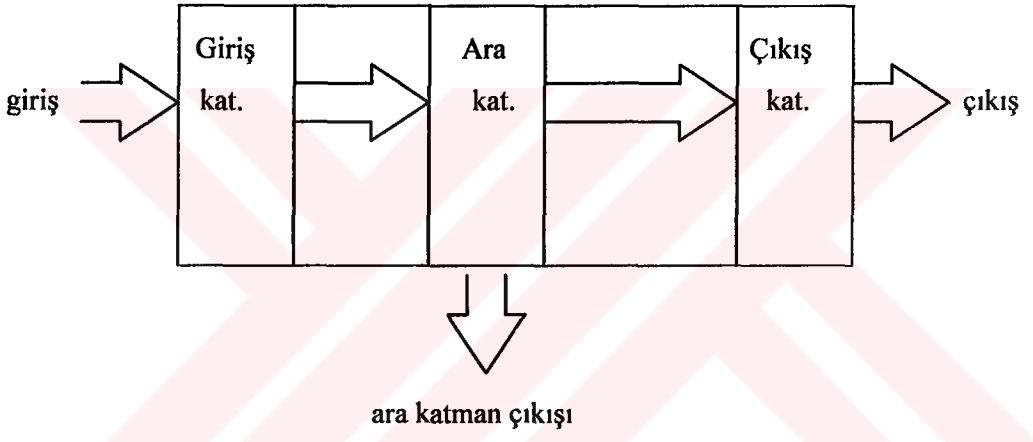


Şekil 2.16. Hatanın ağırlıklara göre değişiminin gösterimi

Eğitim işlemi yapılırken ağ, yerel veya genel olup olmadığı bilinmeyen bir en küçük değere ulaşır. Eğer hata değeri yerel en küçük değere ulaşmışsa, bir süre sonra ağ çıkışlarındaki hata istenmeyen şekilde yükselebilir. Bu problem genellikle fazla bir zorluğa neden olmaz. Eğer bir ağ istenilen bir çözüme ulaşmadan durursa, ara birimlerde veya eğitim parametrelerinde değişiklik yapılarak veya başlangıç ağırlık değerleri farklı bir değer aralığında seçilmesi ile problem çözülebilir. Bir ağ istenilen bir çözüme ulaştığında, bu çözümün genel en küçük hata değeri olduğu tam olarak söylenemez. Fakat ulaşılan bu değerin genel en küçük değer mi, yoksa yerel enküçük değer mi olduğu önemli değildir (Lippman, 1987).

3. GERİYANSIMALI AĞIN GÖRÜNTÜ SIKIŞTIRMADA KULLANILMASI

Geriyansımalı yapay sinir ağı, hedeflenen çıkışlar ile hesaplanan çıkışlar arasındaki farkı, her bir birime geri yansıtan ve bu işlemi tekrarlayarak, eğitimi sağlayan yapay sinir ağıdır. Bilgisayarda görüntü, n x m boyutlu piksellerin matrisi olarak algılanır. Bu matris değerleri, 256 farklı monokrom seviye ile işaretlenmektedir. Bilgisayar ile görüntü sıkıştırılırmada, görüntü matrisi, 8 x 8'lik matrislere bölünerek 64 elemanlı vektörler elde edilir. Bu vektörler yapay ağ için, giriş ve hedef çıkış vektörleri olarak belirlenmektedir. Böyle bir yapay sinir ağı yapısına ilişkin blok diyagramı şekil 3,1'de verilmiştir.

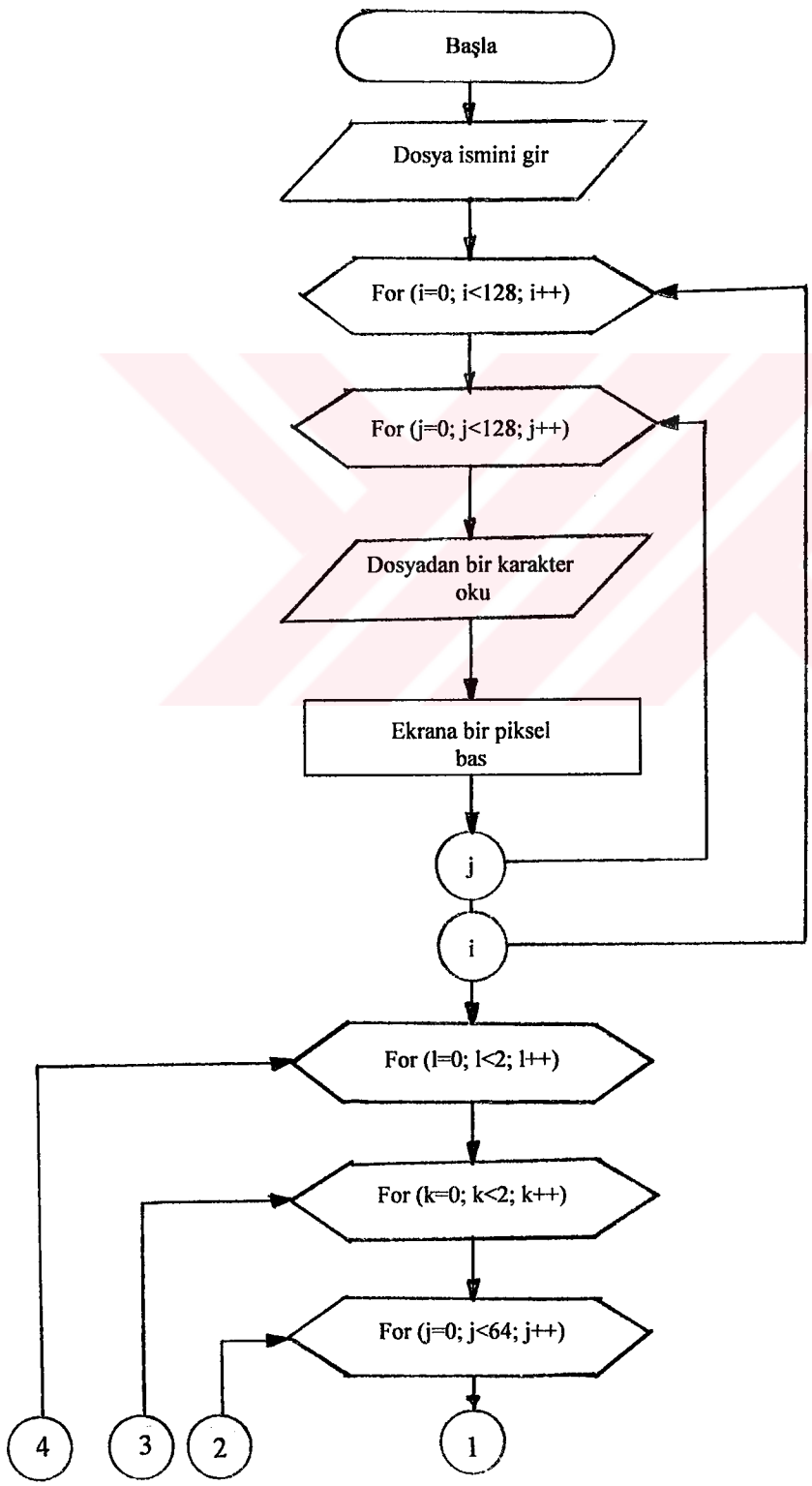


Şekil 3.1. 64 giriş, 4 ara, 64 çıkış birimli yapay sinir ağının blok diyagramı.

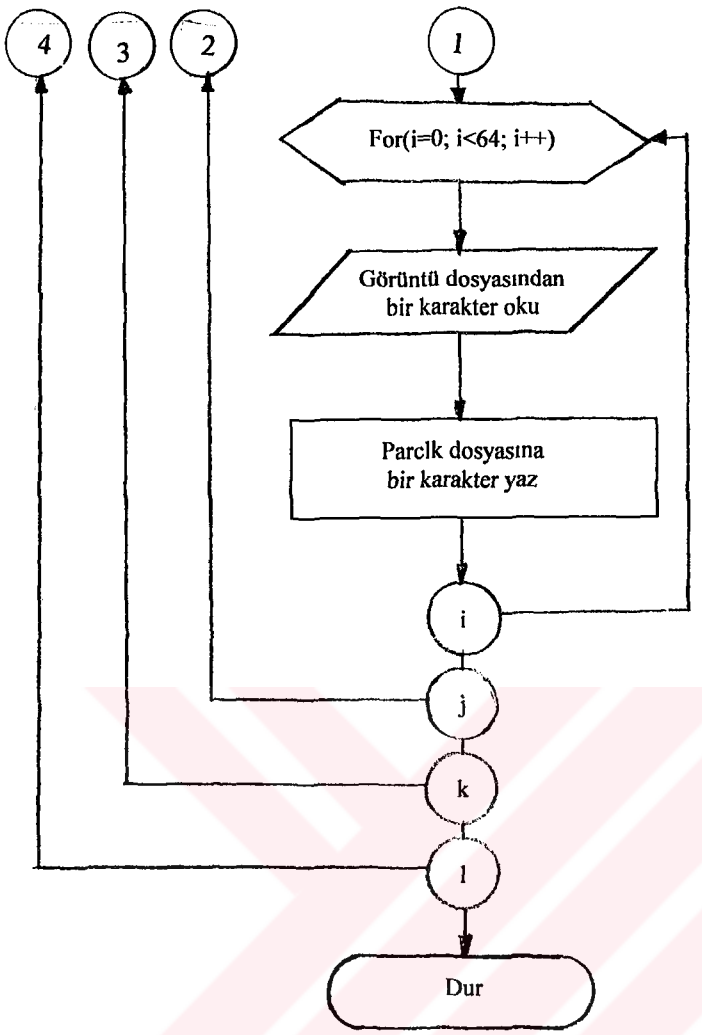
Görüntü sıkıştırma probleminin çözümünde ilk adım, hedeflenen sıkıştırmayı sağlayacak uygun bir ağ yapısının bulunmasıdır. Seçilen bu mimari ağ, hedeflenen bir bilgi azalımını sağlamalıdır. Çıkışlar ara katmandan alınacağı için, böyle bir ağ mimarisi, giriş katman birimlerinden daha az ara katman birimi kullanılarak tasarımlanabilir. Şekil 3,1'de gösterildiği gibi, yapay sinir ağı için giriş ve çıkış katmanlarında 64 işlem elemanı, ara katmanda 4 işlem elemanı kullanıldığında, sıkıştırılmış bilgiler ara katmandan elde edilebilir. Ara katman birimlerinin çıkışı, çıkış katmana doğru yayılmasıyla kodlanmış bilgiden asıl görüntünün elde edilmesi mümkündür.

Görüntüden vektörlerin elde edilmesi bilgisayar yardımıyla yapılabilmektedir. Programlama dilleri genellikle, bir programın çalışması için 64 KB alan ayırdıkları için, görüntü boyutları da 64 x 64'lük boyutlarda sınırlandırımalıdır. Böylece görüntü 64 x 64

boyutlu parçalara ayrılmaktadır. Yapılan çalışmalarda 128 x 128 boyutlu görüntüler kullanıdığından, bu görüntüler dört parçaya ayrılarak yapay sinir ağlarında eğitim işlemleri sürdürülmektedir. Böyle bir işlemi bilgisayar destekli olarak yapılabilmesi için oluşturulması gereken akış diyagramı şekil 3,2'de verilmektedir.

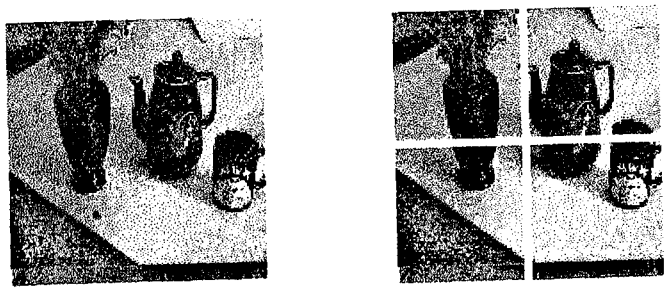


Şekil 3.2. Parca64.cpp programının akış diyagramı.



Şekil 3.2 Paca64.cpp programının akış diyagramı (devamı).

Şekil 3,2'de verilen algoritmada 128 x128 boyutlu görüntü bilgisi karakter karakter okunarak ekrana yazdırılarak asıl görüntü ekranda gösterilmekte ve daha sonra görüntü bilgisinden birer karakter okunarak sırasıyla Parc00, Parc01, Parc10, Parc11 adlı dosyalar elde edilmektedir. Bu görüntü parçaları 64 x 64 boyutlu piksel değerlerinin matrisleridirler. Örnek alınan çiçek görüntüsünün eş dört parçaya ayrılması şekil 3,3'de verilmiştir.



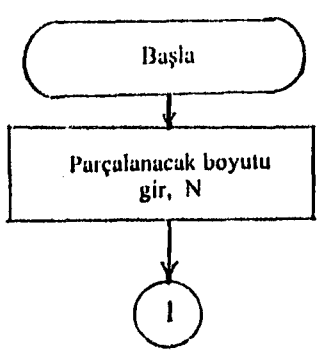
Şekil 3.3 Cicek.img görüntüsünün 64 x 64 lük parçalara ayrılması.

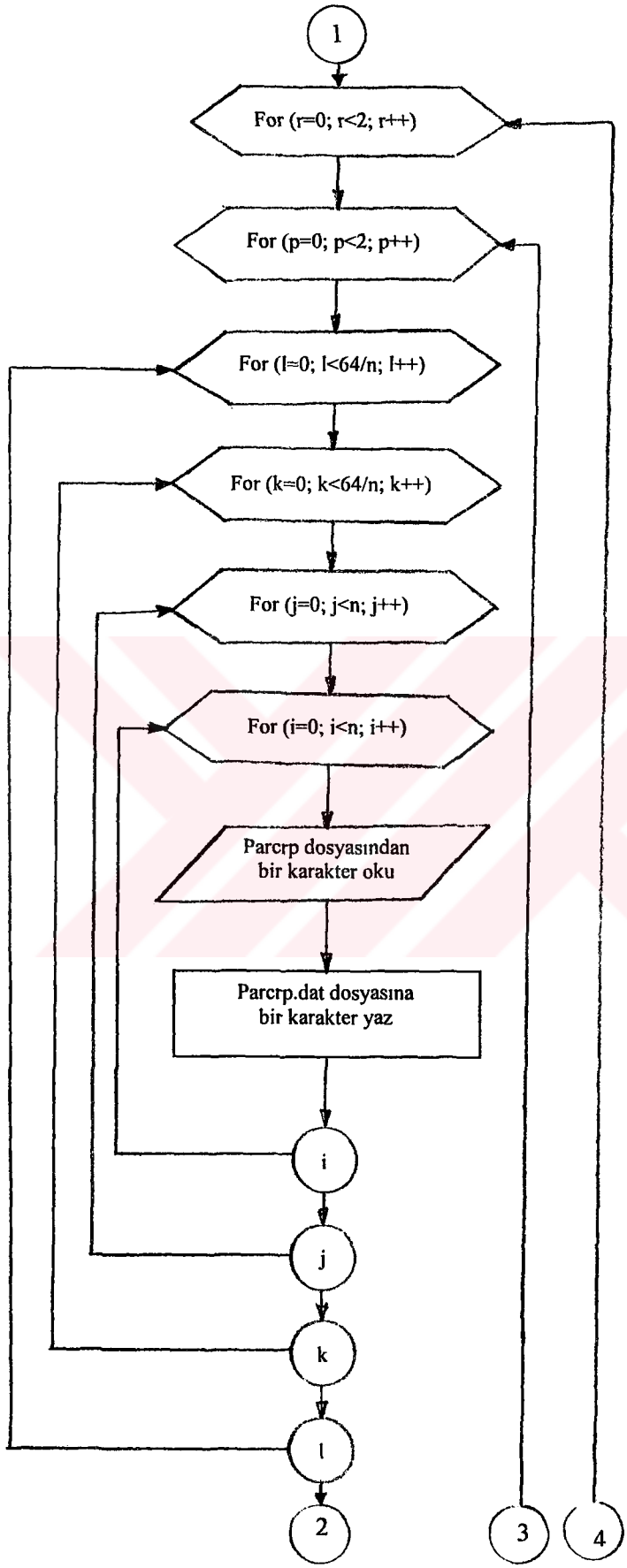
Şekil 3,3' de verilen 128 x 128 boyutlu görüntünün bilgisayara 64 x 64 olarak aktarılabilmesi (işlenebilmesi) için parc00, parc01, parc10, parc11 parçaları elde edilmiştir. Bu görüntü parçaları asıl görüntü dosyasının alt dosyalarıdır. Bilgisayarda bu alt dosyalar ile görüntü işlenecektir. Bu dosyaların özellikleri çizelge 3,1'de gösterilmiştir.

Çizelge 3.1. 128 x 128 boyutlu görüntü dosyasından oluşturulan 64 x 64 boyutlu alt dosyalar.

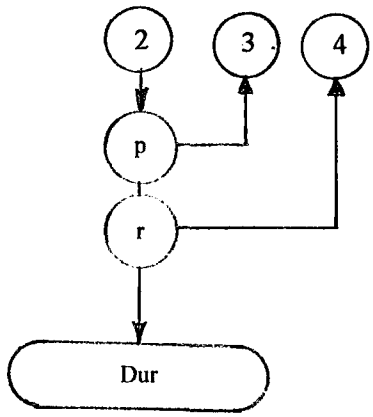
Dosya ismi	Boyutu	Bilgi Tipi	Uzunluk (Byte)
Parc00	64 x 64	Char	4096
Parc01	64 x 64	Char	4096
Parc10	64 x 64	Char	4096
Parc11	64 x 64	Char	4096

Çizelge 3,1'de özellikleri belirtilen görüntü parçalarına ait dosyalar, eğitimde kullanılabilmesi için 8x8'lik alt vektör gruplarına ayrılması gereklidir. Bu işlem için Prodat.cpp programı kullanıldı. Prodat.cpp programının algoritması şekil 3,4'de verilmektedir. Bu program ile, her bir dosya 64 elemanlı 64 vektör olarak tanımlanmaktadır. Bu işlemle elde edilen vektör elemanları [0,1) aralığında normalize edilmektedirler.





Şekil 3.4. Prodat.cpp programının akış diyagramı (devamı).



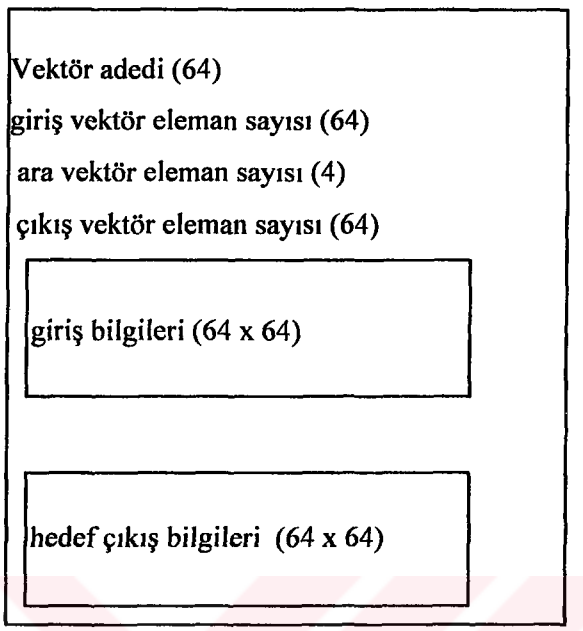
Şekil 3.4 Prodat.cpp programının akış diyagramı (devamı).

Ek..2'de Prodat64.cpp programı verilmiştir. Böylece çizelge 3.2 de verilen .dat uzantılı dosyalar elde edilebilmektedir.

Çizelge 3.2. Prodat.cpp Programıyla elde edilen dosyalar.

Dosya ismi	Boyutu	Bilgi Tipi	Uzunluğu (Byte)
Parc00.dat	64 x 64	Float	16384
Parc01.dat	64 x 64	Float	16384
Parc10.dat	64 x 64	Float	16384
Parc11.dat	64 x 64	Float	16384

Çizelge 3,2'de göterilen dosyalar, geriyansımalı ağ için oluşturulan Bp.c programında eğitilebilmesi için, editör yardımıyla şekil 3,5'de verilen biçime dönüştürülmektedirler. Bu dosyada, eğitimde kullanılan vektör adedi, giriş vektör eleman sayısı, arakatman eleman sayısı , çıkış katman eleman sayısı ile birlikte giriş bilgileri ve çıkış bilgileri yer almaktadır. Bp.c programı bu dosyadan aldığı bilgilere göre bir ağ oluşturur. Oluşan bu yapay sinir ağında, dosyada kayıtlı bulunan giriş ve hedef çıkış değerlerini kullanarak yapay sinir ağının eğitilmesini sağlanmaktadır.



Şekil 3.5. Eğitim için hazır duruma getirilmiş dosyalar

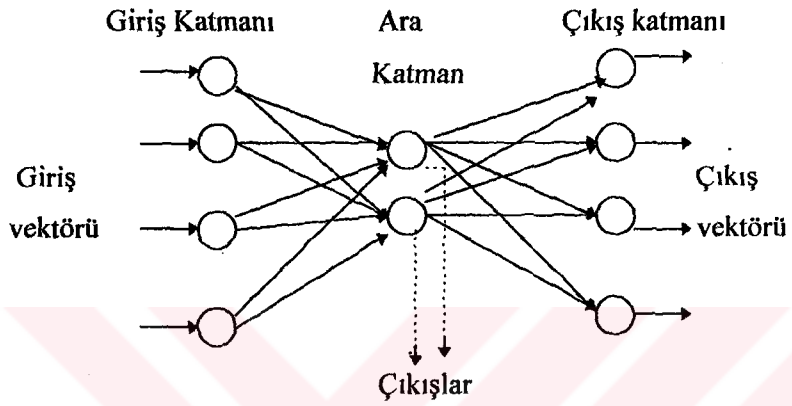
Şekil 3,5'de sembolik gösterimi verilen Pacbp00.dat, Parcbp01.dat, Parcbp10.dat, Parcbp11.dat dosyalarının özellikleri çizelge 3,3'de verilmiştir.

Çizelge 3.3. Bp.c Programında yapay sinir ağı girişine uygun hale getirilmiş (eğitilebilir) dosyalar.

Dosya İsmi	**Boyutu**	**Bilgi Tipi**	**Uzunluğu (Byte)**
Parcbp00.dat	64x128	Float	32768
Parcbp01.dat	64x128	Float	32768
Parcbp10.dat	64x128	Float	32768
Parcbp11.dat	64x128	float	32768

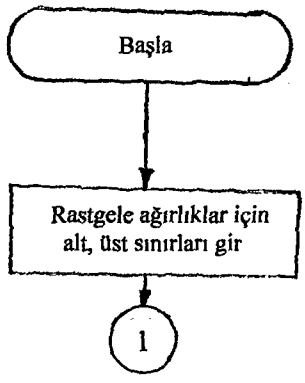
Çizelge 3,3'te verilen dosya özelliklerinden görüleceği gibi, bu dosyaların eğitilebilmesi için 64 giriş düğümlü bir yapay sinir ağı gerekmektedir. Girişler, aynı zamanda hedeflenen çıkışlar olarak kullanılacağından, ağ çıkış katmanında da 64 düğüm kullanılmalıdır. Böyle bir ağ yapısında eğitim yapılabilmesi için Bp.c programı oluşturuldu. C kaynak kodları

kullanarak gerçekleştirilen bu program, görüntünün eğitilerek sıkıştırılmasında temel program olarak kullanıldı. Programda, ağ mimarisi olarak üç katmanlı ağ kullanılmış ve böyle bir ağ yapısı şekil 3,6'de verilmiştir. Ağ çıkışları ise ara katmandan alınmaktadır. Bu düşünce ile herhangi bir görüntünün istenilen (n x n) boyutlu alt piksel kümeleri elde edilebilmektedir.

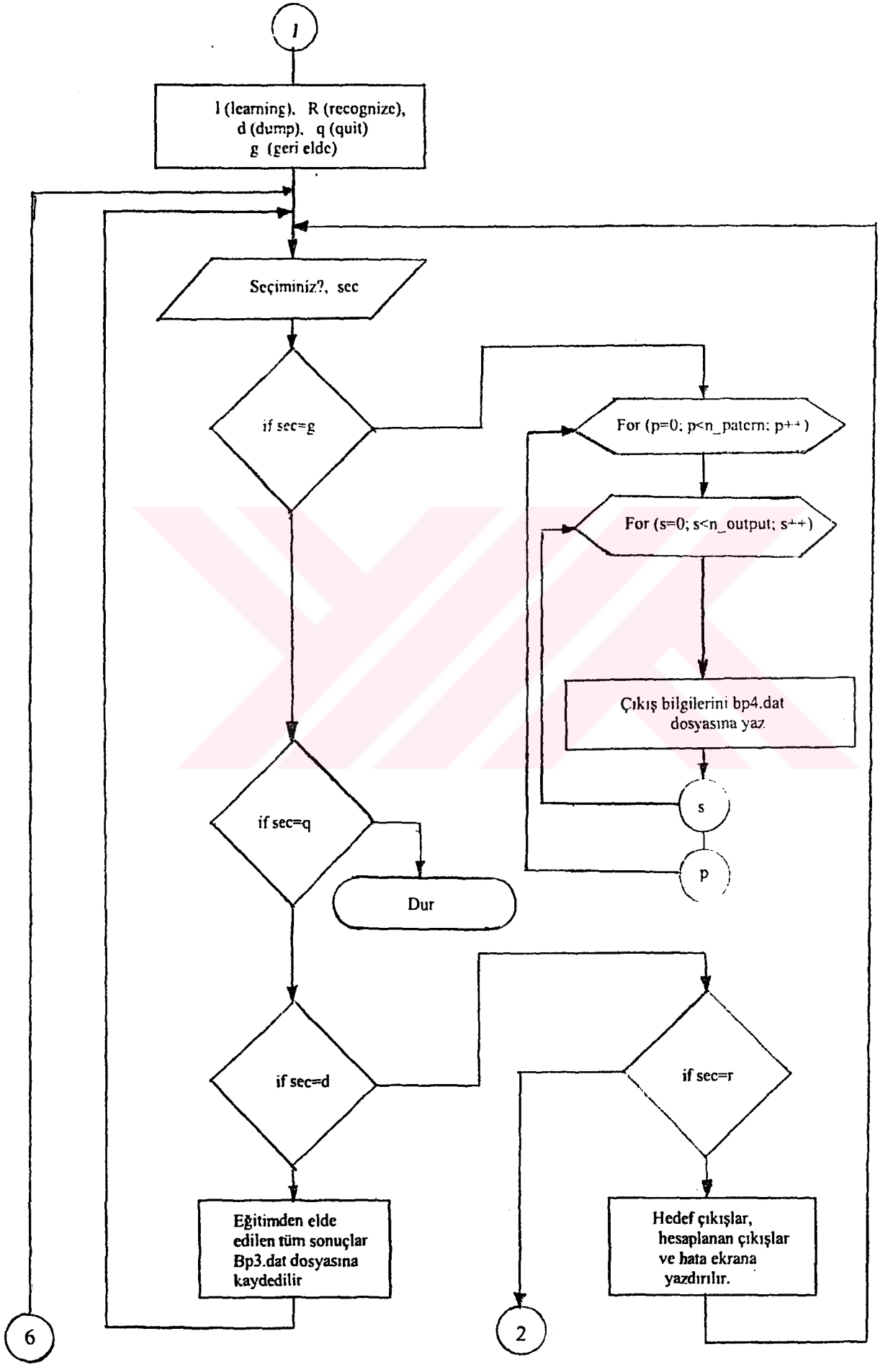


Şekil 3.6. Üç katmanlı geriyansımalı yapay sinir ağı.

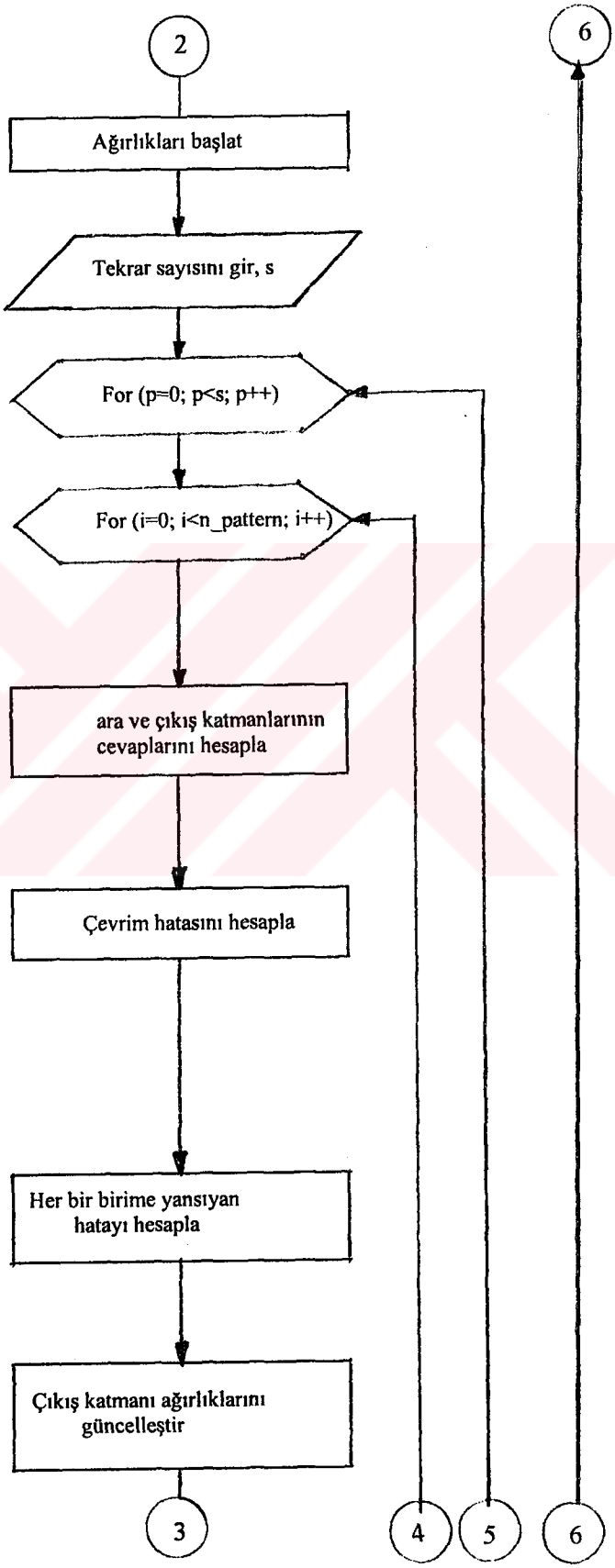
Ağın eğitimi sıkıştırılmış bilgilerden ara katman-çıkış katmanı arasındaki ağırlıkları kullanarak asıl görüntüye en yakın görüntüyü elde etmektir. Hedef çıkış bilgileri olarak asıl görüntü bilgileri olan giriş bilgileri alınmaktadır. Çıkış katmanından elde edilecek bilgiler, hedef bilgilere ne kadar benzerse, o oranda hatasız sıkıştırılmış bilgi ve ara katman-çıkış katman arasında bulunan ağırlık bilgileri de elde edilmiş olmaktadır. Her bir 64 x 64'lük görüntü bilgilerini ayrı ayrı ele alarak verilen tekrar sayısınca eğiten ve sonuçta hedefe en yakın sıkıştırılmış değerleri, ağırlık değerlerini, RMS hata değerini bularak istenildiğinde tüm eğitim işlevini bp3.dat dosyasına aktaran Bp.c programının akış diyagramı şekil 3,7'de verilmektedir.



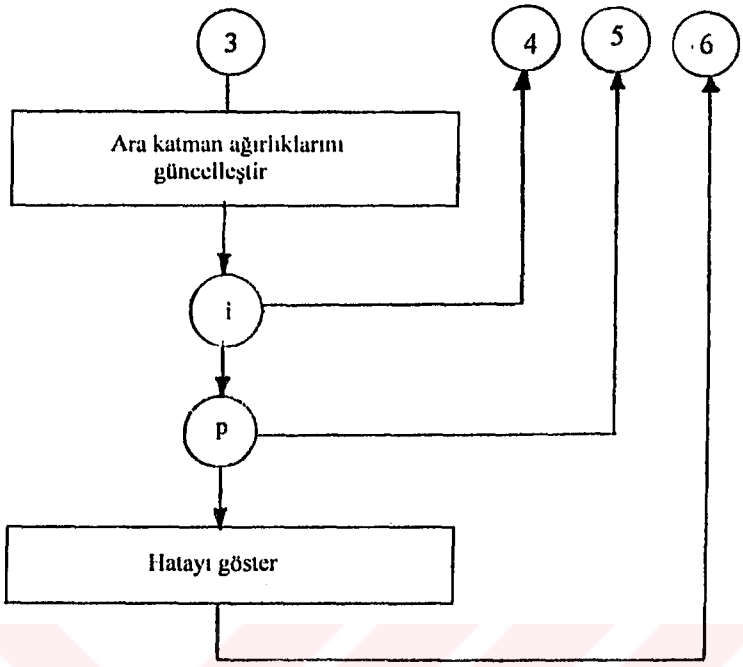
Şekil 3.7. Bp.c programının akış diyagramı.



Şekil 3.7. Bp.c programının akış diyagramı (devamı).



Şekil 3.7. Bp.c programının akış diyagramı (devamı).



Şekil 3.7. Bp.c programının algoritması (devamı).

Şekil 3,7'da akış diyagramı verilen Bp.c programının C kaynak kodu Ek.3' te verilmektedir. Bp.c programında geri yansımalı algoritmanın yapısı gereği eğitimde belirli oranda bir hata oluşmaktadır. Bu hata sonucunda, görüntü bilgisinin geri elde edileceği bilgilerle asıl görüntü bilgilerinde farklılıklar oluşmakta ve sonuçta kayıplı bir sıkıştırma oluşmaktadır. Belirli sayılarda eğitimin tekrar edilmesi ile hatanın azaltılması sonucunda elde edilen görüntü farklılıklarının tekrar sayısına göre değişimi şekil 3,8'de verilmiştir.

a) b) c) d)

Şekil 3.8. Asıl görüntünün farklı sayılarda eğitilmesi ile oluşan sıkıştırılmış görüntüden elde edilen görüntüler. a) 1000 eğitim tekrarı ile eğitilmiş görüntü. b) 2000 eğitim

tekrarı ile eğitilmiş görüntü. c) 4000 eğitim tekrarı ile eğitilmiş görüntü. d) 6000 eğitim tekrarı ile eğitilmiş görüntü.

Şekil 3,8'de hatanın küçülmesi ile görüntü bilgisinin asıl bilgiye yaklaştığı görülmektedir. Elde edilen görüntünün asıl görüntüye yaklaşmasında ağda kullanılan ara katman eleman sayısı önemli bir etkiye sahiptir. Ara katman eleman sayısı arttıkça hata minimum seviyeye inmekte, azaldıkça ağın eğitimi sonucunda oluşan hata artmaktadır. Geri yansımalı ağın yapısı gereği sıkıştırılmış bilgiler ara katmandan alınmaktadır.Ağın eğitiminde kullanılan Bp.c programında ara katman sıkıştırılmış bilgiler, hidden matris değişkeni içerisine yığılmıştır. Bu değişkende saklanan değerlere ait özellikler çizelge 3,4'de verilmiştir.

Çizelge 3.4. Hidden matris değişkenindeki sıkıştırılmış bilgiler.

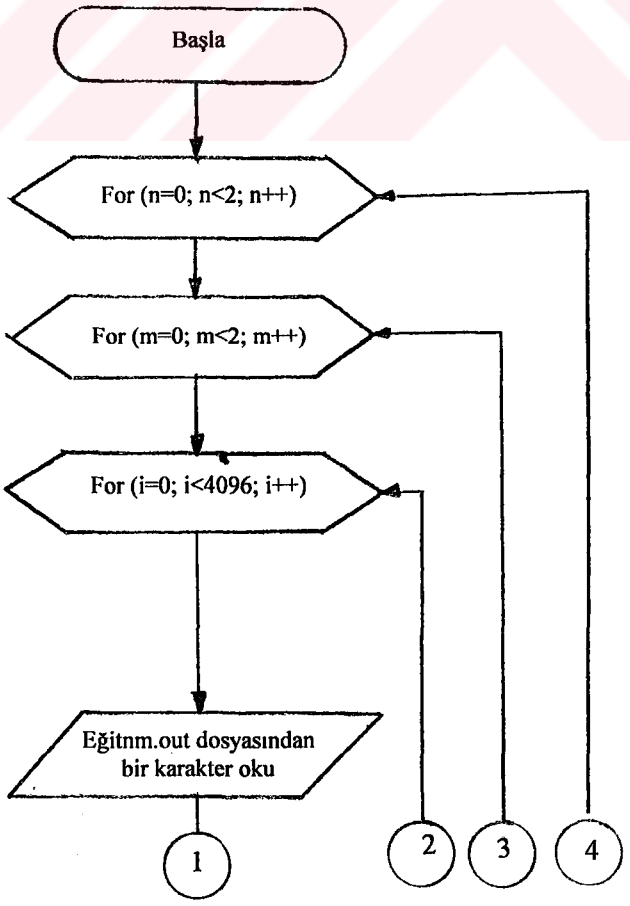
Dosya İsmi	**Boyutu**	**Sıkıştırılmış Boyut**	**Bilgi tipi**	**Uzunluk (Byte)**
Parcbp00.dat	64 x 64	4 x 64	Float	1024
Parcbp01.dat	64 x 64	4 x 64	Float	1024
Parcbp10.dat	64 x 64	4x 64	Float	1024
Parcbp11.dat	64 x 64	4 x 64	Float	1024

Belirli sayıda tekrarlanan eğitim sonucunda hata istenilen değere ulaşabilmektedir. Bp.c programındaki geri elde edim seçeneği ile, ara katmandaki sıkıştırılmış bilgiler, ara katman-çıkış katmanı ağırlık değerleri ile çarpılıp, bulunan sonuçlar çıkış katmanındaki işlem elemanlarında toplanmaktadır. Daha sonra elde edilen sonuçlar sigmoid fonksiyonuna aktarılarak 64x64 lük Egit00.out, Egit01.out, Eğit10.out ve Eğit11.out dosyaları elde edilmektedir. Bu dosyalarda, görüntüyü tekrar elde edebilmek için gerekli olan görüntü bilgilerinin normalize edilmiş değerleri bulunmaktadır. Bu dosyalar ile ilgili özellikler çizelge 3,5'de verilmiştir.

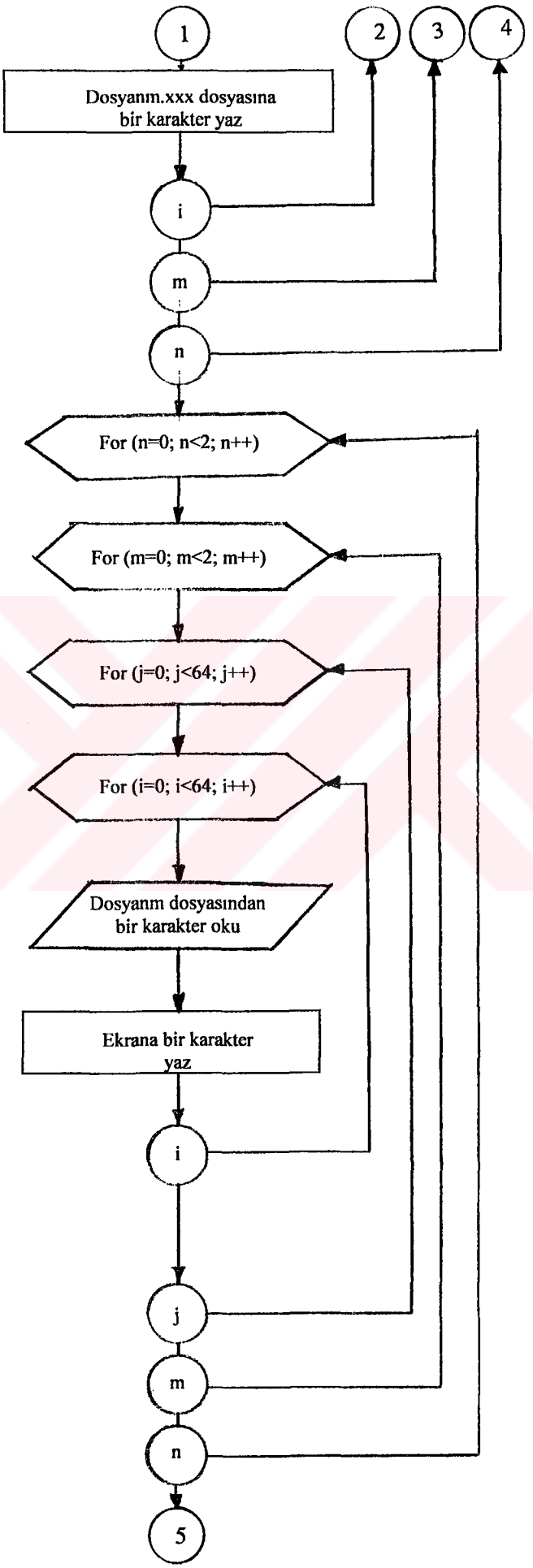
Çizelge 3.5. Sıkıştırılmış bilgilerden elde edilen görüntü dosyaları.

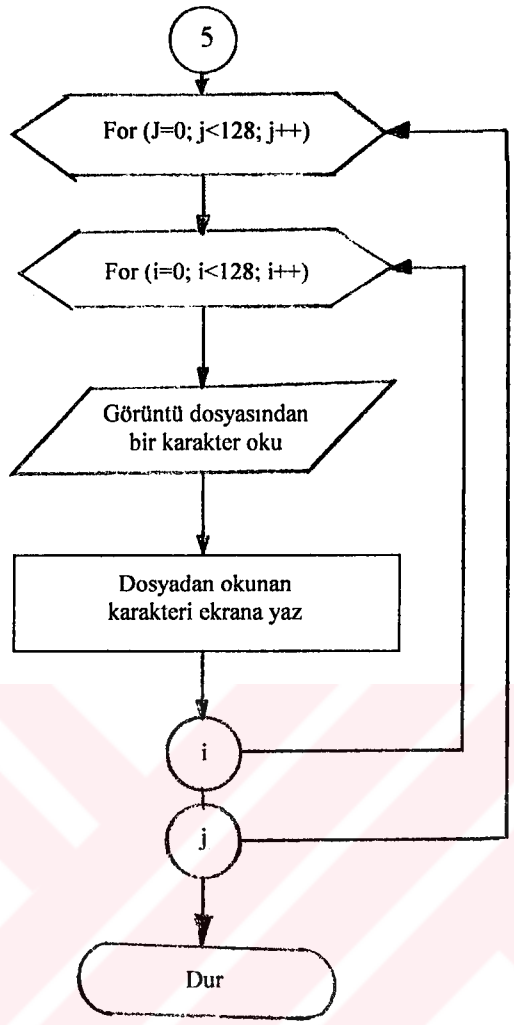
Dosya İsmi	**Boyut**	**Bilgi tipi**	**Uzunluk (Byte)**
Egit00.out	64 x 64	Float	16384
Egit01.out	64 x 64	Float	16384
Egit10.out	64 x 64	Float	16384
Egit11.out	64 x 64	Float	16384

Çizelge 3,5'de verilen sıkıştırılmış dosyalardan elde edilen eğitilmiş bilgiler ile geri yansımalı sıkıştırılmış görüntü sonucu ekranda elde edilebilir. Bu işlem için geliştirilen Gerihbp.cpp isimli program, geri elde edilen görüntü ile asıl görüntüyü ekrana yansıtmakta ve böylece sonucun karşılaştırılması aynı anda yapılabilmektedir Ek.4'te Gerihbp.cpp programının C kaynak kodu sunulmuştur. Gerihbp.cpp porgramının akış diyagramı ise şekil 3,9'de verilmektedir.



Şekil 3.9. Gerihbp.cpp programının akış diyagramı.


1
2
3
4
Dosyanm.xxx dosyasına bir karakter yaz
i
m
n
For (n=0; n<2; n++)
For (m=0; m<2; m++)
For (j=0; j<64; j++)
For (i=0; i<64; i++)
Dosyanm dosyasından bir karakter oku
Ekrana bir karakter yaz
i
j
m
n
5



Şekil 3.9. Gerihbp.cpp programının akış diyagramı (devamı).

Böylece geri yansımalı yapay sinir ağı kullanarak sıkıştırılmış görüntüden eğitilerek elde edilen görüntü ekranda oluşur. Görüntüde oluşan kayıp, hata değeri ile ters orantılıdır. Hata değeri görüntü boyutları ile doğrudan ilişkilidir. Görüntü boyutları arttıkça eğitim süresi ve hata değeri artmakta ve dolayısıyle çok kayıplı sonuçlar elde edilebilmektedir. Ara katmandaki işlem eleman sayısının düşük tutulması durumunda kazanç oranı artmakta, fakat buna parelel olarak eğitim süresi ve hata değeri de artmaktadır. Örnek olarak çizelge 3,6'da farklı boyutlarda ve değişik yapılarda elde edilen sonuçlar verilmektedir

Çizelge 3.6. Geriyansımalı algoritma kullanarak elde edilen sonuçlar.

Boyutlar	Ağ	Tekrar sayısı	Sıkıştırma oranı	Hata (%)
16 x 32	4 : 3 : 4	32000	15/16	0.01
16 x 32	4 : 2 : 4	32000	5/8	0.02
16 x 32	4 : 1 : 4	32000	5/16	0.04
32 x 32	16 : 4 : 16	32000	5/16	0.05
32 x 32	16 : 8 : 16	32000	5/8	0.02
16 x 16	16 : 4 :16	64000	1/2	0.03

Çizelge 3,6'da görüleceği gibi geriyansımalı yapay sinir ağı kullanarak görüntünün sıkıştırılması işlemini minimum hata ile sonuçlandırmak için eğitimin tekrar sayısının artırılması gerekmektedir. Fakat eğitim sigmoid fonksiyonu kullanarak yapılırsa, hata, artan süreyle daha az oranlarda azalacaktır. Hatanın minimum seviyeye indirilmesi için yapılması gereken işlemlerden biri de ağ mimarisinde öngörülen arakatman işlem eleman sayısının artırılması olabilmektedir. Bu durumda hatada belirli bir düşüş görülecektir. Bu işlemin sakıncası ise; çıkışlar ara katmandan alındığı için sıkıştırma oranını azaltmasıdır.

Çizelge 3.6 ve çizelge 3.7'de verilen sıkıştırma oranları eşitlik (2.52)'den hesaplanmaktadır;

$$\mathbf{M_T = \frac{n_h \times n_v \times u_h + w_n^h \times u_w}{d_i \times u_i}} \qquad (2.52)$$

eşitlik (2.52)'de M_T sıkıştırma oranı, n_h ara katman sayısı, u_h ara katman birimi çıkışının byte cinsinden uzunluğu, $\mathbf{w_n^h}$ ara katman ağırlık sayısı, u_w bir ağırlık değerinin byte cinsinden uzunluğu, d_i görüntü boyutu, u_i görüntü pikselinin byte cinsinden uzunluğudur. Çizelge 3.7'de elde edilen sıkıştırma oranları, float tipindeki ara katman birimi çıkışları ile ağırlık değerlerinin kayıpsız sıkıştırma yapan bir programla (örneğin Zıp.exe programı) sıkıştırılarak iletildiği varsayılarak hesaplanmıştır. Böylece float tipindeki karakterler 2 byte uzunluklu karakterler olarak değerlendirilebilirler. Ek.5'te orjinal görüntüleri ve sıkıştırılmış değerlerden geri elde edilen, lenna.img, Cicek.img, Tel.img görüntüleri sunulmuştur. Bu görüntülerle ilgili bilgiler çizelge 3.7'de gösterilmiştir.

Çizelge 3.7. Geriyansımalı algoritma kullanarak elde edilen görüntü bilgileri.

Asıl görüntü	Parçalı görüntü	Boyutları	Tekrar sayısı	Sıkıştırma oranı	Hata (%)
LENNA	lenna00	64 x 64	10000	1/4	2.41
	lenna01	64 x 64	10000	1/4	3.2
	lenna10	64 x 64	10000	1/4	4.1
	lenna11	64 x 64	10000	1/4	3.1
ÇİÇEK	Cicek00	64 x 64	10000	1/4	6.1
	Cicek01	64 x 64	10000	1/4	7.2
	Cicek10	64 x 64	10000	1/4	4.2
	Cicek11	64 x 64	10000	1/4	5.3
TELEFON	Tel00	64 x 64	10000	1/4	3.2
	Tel01	64 x 64	10000	1/4	4.1
	Tel10	64 x 64	10000	1/4	4.3
	Tel11	64 x 64	10000	1/4	6.2

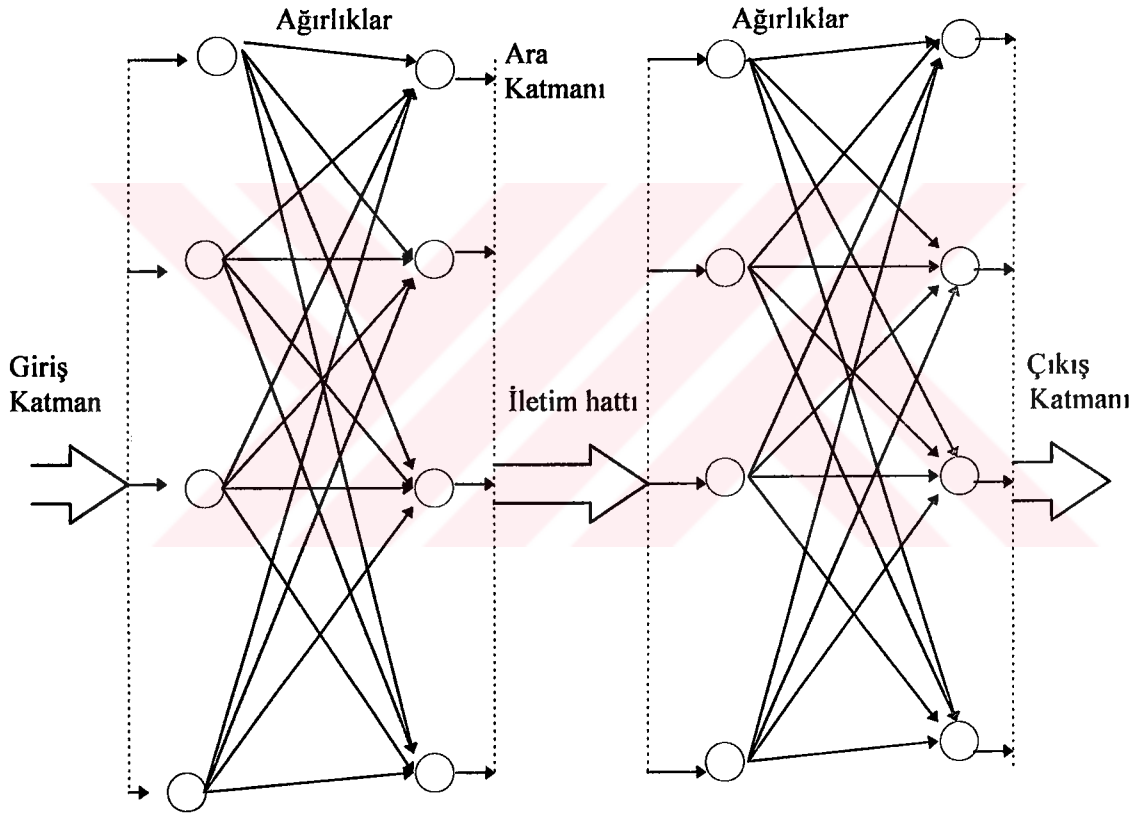
4. SONUÇ VE ÖNERİLER

Tanımlama işlemlerinin bilgisayarla yapabilmesi için, pentium 100 mikroişlemcili, 8 MB RAM belleğe sahip bilgisayar kullanılmıştır. 128 x 128 boyuttan daha büyük görüntüler için, eğitilecek parça sayısı fazla olacağından ve bu durum da eğitim süresinin uzamasına neden olacağı için, sıkıştırma işlemlerinde tercih edilmemiştir. Bununla birlikte, fazla işlemci biriminin kullanıldığı ağlarda, işlemci sayısına paralel olarak, fazla sayıda hesaplamalar yapılacağından eğitim süresi uzun zaman alabilmektedir. Aynı şekilde, giriş bilgi kümesinde fazla sayıda vektör kullanılması da eğitim süresini uzatmakta ve hatayı da artırmaktadır. Bu gibi nedenlerden dolayı yapay sinir ağlarının -64:4:64 gibi- küçük boyutlu olanları ve görüntülerin de 64 x 64 lük boyutları tercih edilmelidir. Bu nedenle bu yöntemle hatanın düşürülmesi en son düşünülmelidir.

Geriyansımalı çalışmalarda ağ yapısı önemlidir. Ara katman sayısı ile ilgili sınırlamalar yoktur; bir ağ sadece bir veya daha fazla ara katmana sahip olabilmektedir. Eğer geriyansımalı ağların çalışması arakatmanlar açısından karşılaştırılırsa, ağlar,en geniş ara katmanlı olanlar dar olanlara göre daha iyi performans sağlar. Geriyansımalı yapay sinir ağlarında öğrenme süresi, doğrudan doğruya ağın büyüklüğüne ve vektör sayısına bağlıdır. Geniş ağlarda vektördeki örnek sayısı fazla olduğundan öğrenme süresi uzamaktadır. Öğrenme parametresinin seçimi bir başka önemli konudur. Genellikle 0,1 aralığında seçilir. Bütün ağ içindeki işlemci elemanlar aynı öğrenme parametresi kullanabildiği gibi her katman için bu değer farklı olabilmektedir. Bu değer tüm öğrenme periyodu üzerinde aynı kalabilir, veya belirli bir periyottan sonra değiştirilebilmektedir. Eğitim periyodunun başlangıcında öğrenme parametresi büyük değerlerde tutularak hatanın minimuma yaklaşma oranı artırılabilmektedir. Belirlenen eğitim süresinden sonra, öğrenme parametresi küçültülebilir.

Bir önemli parametre de momentum değeridir. Momentum parametresi de öğrenme değeri gibi değişken olarak kullanılabilmektedir. Bunun yanısıra öğrenme değeri ile momentum değeri birbiriyle ilişkili olarak değiştirilmek suretiyle hatanın değişimi de gözlenebilmektedir.

Eğitim işlemleri, herbir problemin kabul edilebilir hata değerine ulaştığında işlem tamamlanır. Geri yansıma işleminin sıkıştırma işleminde kullanılmasında, ağ ara birimlerinin asıl boyutu içerisine sıkıştırılmış bilginin saklanabilmesi için uygun ağırlık değerlerinin bulunması gerekir. Ağırlıklar, eğitim esnasında, her iterasyonda değiştirilerek en uygun ağırlık değerleri bulunmaya çalışılır. Eğer ara katman birimlerinde sıkıştırılmış bilgi, belirli bir sistem içerisinde iletilecek olursa, bu bilgi, iletilen yerde tekrar asıl görüntünün elde edilmesi için kullanılabilir. Bu durum şekil 4,1'de verilmiştir.



Şekil 4.1. Sıkıştırılmış bilgilerin iletimi ve orjinal bilginin elde edilmesi .

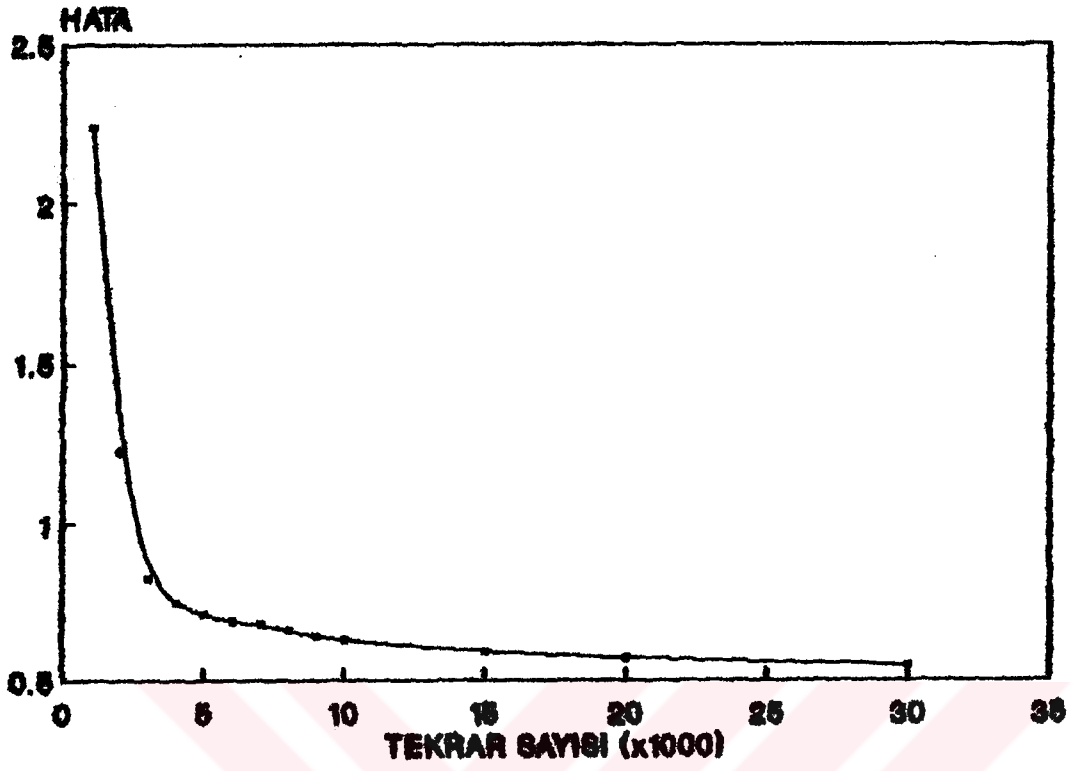
Çizelge 4,1'de geriyansımalı ağ kullanarak, değişik boyutlardaki dosyalar üzerinde yapılan eğitim işlemlerinin sonuçları verilmiştir. Çizelge 4,1'de öncelikle, eğitimde kullanılan dosyaların boyutları, vektör sayıları, her bir vektörün eleman sayısı gibi dosya özellikleri verilmiştir. Daha sonra bu dosyaların eğitilmesi için kullanılan farklı ağ mimarilerine ait bilgiler verilerek giriş katman, ara katman ve çıkış katmanlardaki düğüm sayıları da verilmektedir. Ayrıca her bir dosya için eğitim tekrar sayıları ile birlikte, bazı dosyalar için

belirli tekrar sayılarında yapılan eğitimin nekadar zamanda yapıldığı verilmektedir. Bütün bu bilgiler doğrultusunda elde edilen kazanç ve hata değerleri de çizelge 4,1'de yeralmaktadır.

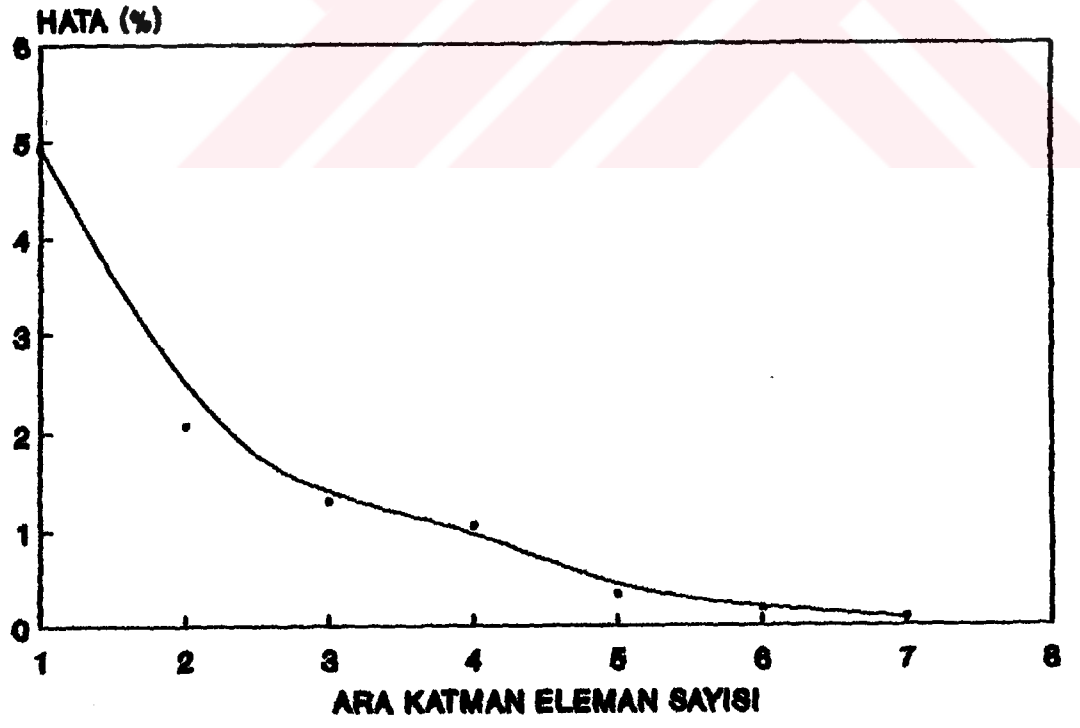
Çizelge 4.1. Geriyansımalı ağ kullanılarak elde edilen görüntü değerleri.

Dosya	Vek. elman sayısı	Vek. sayısı	Boyut	Giriş kat.	Ara kat. elm.	Çıkış kat. elm.	Hata (%)	Sıkıştırma oranı	t (s)	Tekr. Sayısı (x1000)
eh32.dat	64	16	1024	64	4	64	2.24	3.2	30	1
eh32.dat	64	16	1024	64	4	64	1.22	3.2	60	2
eh32.dat	64	16	1024	64	4	64	0.825	3.2	90	3
eh32.dat	64	16	1024	64	4	64	0.75	3.2	120	4
eh32.dat	64	16	1024	64	4	64	0.71	3.2	150	5
eh32.dat	64	16	1024	64	4	64	0.69	3.2	180	6
eh32.dat	64	16	1024	64	4	64	0.68	3.2	210	7
eh32.dat	64	16	1024	64	4	64	0.66	3.2	240	8
eh32.dat	64	16	1024	64	4	64	0.64	3.2	270	9
eh32.dat	64	16	1024	64	4	64	0.63	3.2	300	10
eh32.dat	64	16	1024	64	4	64	0.59	3.2	450	15
eh32.dat	64	16	1024	64	4	64	0.57	3.2	600	20
eh32.dat	64	16	1024	64	4	64	0.55	3.2	900	30
eh32.dat	64	16	1024	64	1	64	4.93	12.8	---	5
eh32.dat	64	16	1024	64	2	64	2.08	6.4	---	5
eh32.dat	64	16	1024	64	3	64	1.3	4.266	---	5
eh32.dat	64	16	1024	64	4	64	1.25	3.2	---	5
eh32.dat	64	16	1024	64	5	64	0.33	2.56	---	5
eh32.dat	64	16	1024	64	6	64	0.19	2.133	---	5
eh32.dat	64	16	1024	64	7	64	0.1	1.828	---	5
eh164.dat	64	1	64	64	4	64	0.0002	0.264	---	5
eh264.dat	64	2	128	64	4	64	0.0004	0.484	---	5
eh464.dat	64	4	256	64	4	64	0.06	0.941	---	5
eh864.dat	64	8	512	64	4	64	0.31	1.77	---	5
eh32.dat	64	16	1024	64	4	64	0.71	3.2	---	5
parcoo.dat	64	64	4096	64	4	64	2.1	8	---	5

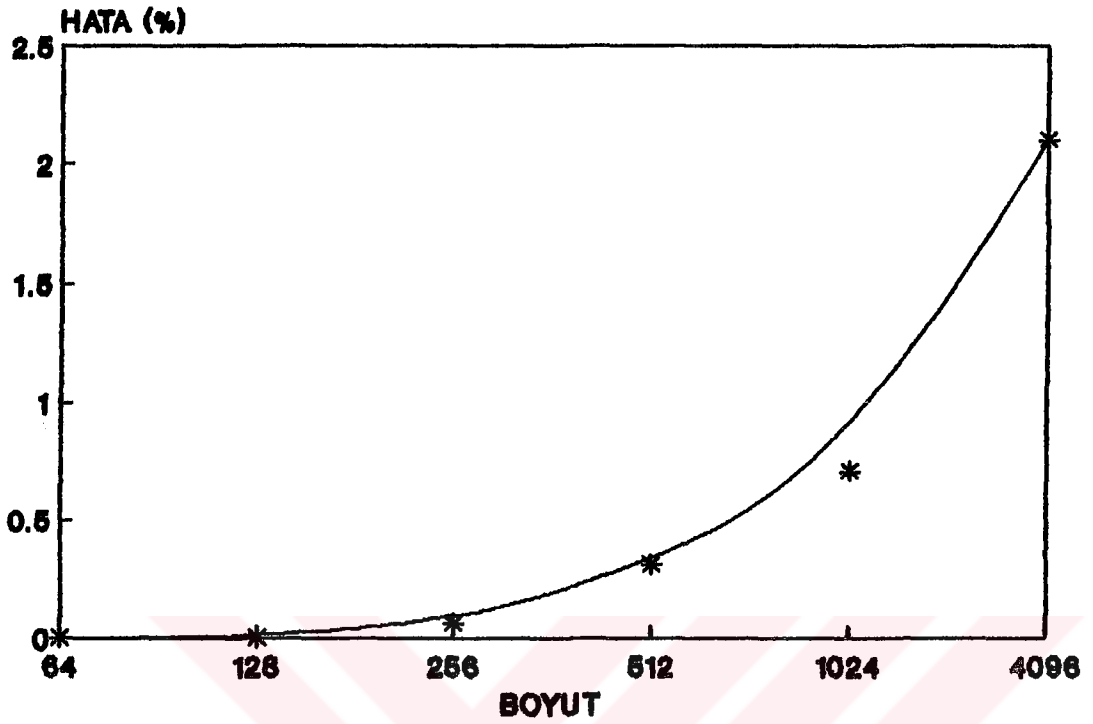
Önemli bir parametre olan hatanın, kazanç, zaman, tekrar sayısı, boyut gibi değişkenlere göre değişimi elde edilebilir. Elde edilecek bu değişimler yardımıyla sıkıştırma işleminde optimum noktanın bulunması sağlanabilecektir. Çizelge 4,1'deki sonuçlardan yararlanarak; şekil 4,2'de hata-tekrar sayısı, şekil 4,3'de hata-ara katman, şekil 4,4'de hata-boyut, şekil 4,5'de hata-kazanç, şekil 4,6'da hata-zaman değişimleri verilmiştir.



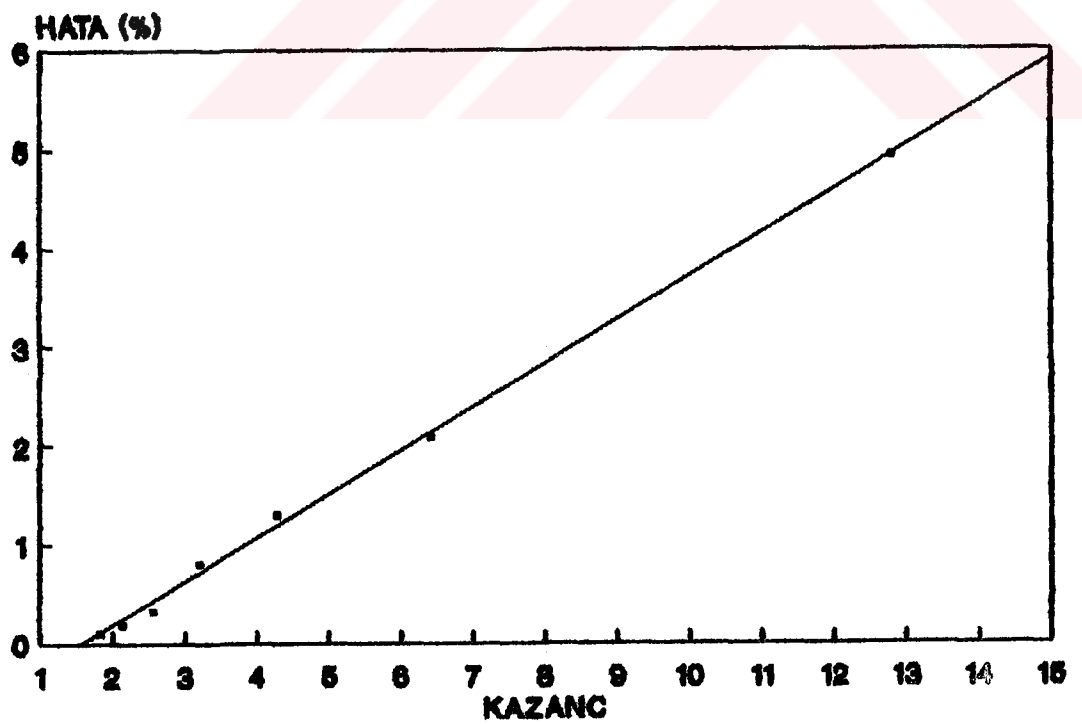
Şekil 4.2. Hata-tekrar sayısı değişimi



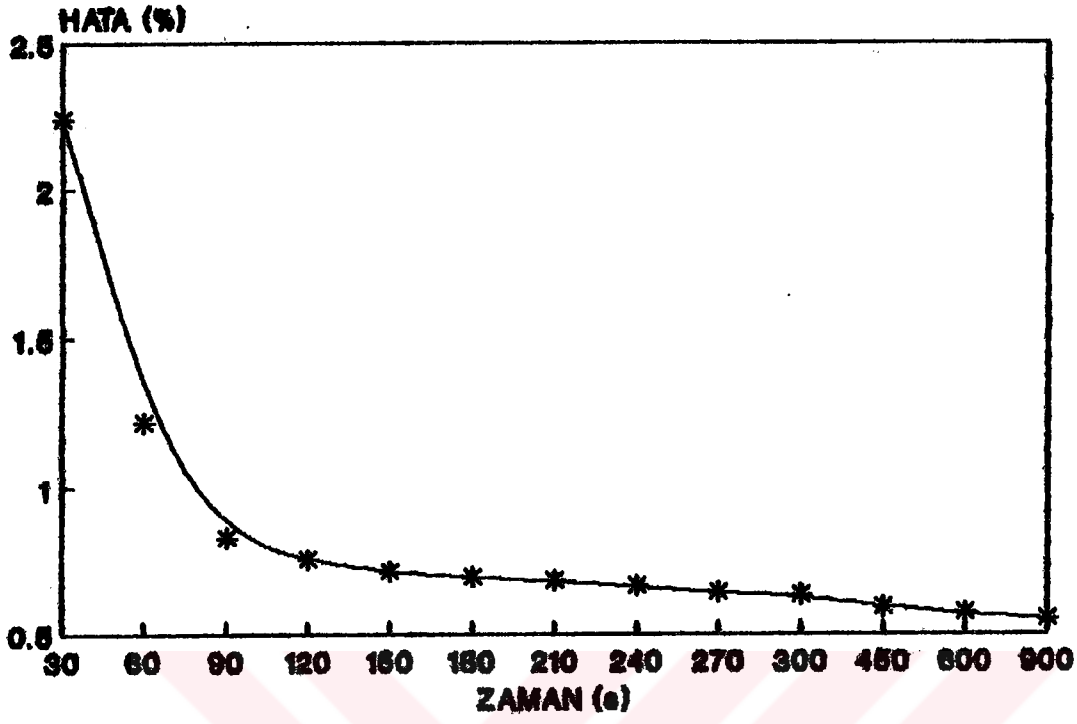
Şekil 4.3. Hata-ara katman eleman sayısı değişimi



Şekil 4.4. Hata-boyut değişimi



Şekil 4.5. Hata-sıkıştırma oranı değişimi



Şekil 4.6. Hata zaman değişimi

Yukarıdaki değişim şekilleri içerisinde yer alan şekil 4,2'de görüleceği gibi hata, tekrar sayısı arttıkça azalmakta fakat bu azalım değeri belirli eğitim tekrar sayısından sonra çok küçük oranlarda değişerek logaritmik bir azalım göstermektedir. Ara katman eleman sayısı görüntü bilgisinin işaretleneceği sıkıştırılmış değerlerin sayısıyla doğru orantılı olduğu için, bu sayı mümkün olduğunca küçük tutulmalıdır. Fakat şekil 4,3'de görüleceği gibi, ara katman eleman sayısı azaldıkca hata değeri yükseldiği için, hatanın kabul edilebilir seviyelerde oluştuğu ara katman eleman sayıları tercih edilmelidir. Nitekim şekil 4.5'de ara katman sayıları değiştirilerek elde edilen hata-kazanç değişiminde, ara katman eleman sayısının azaltılmasıyla elde edilen kazanç artışının, hata değerini de artırdığı görülmektedir. Bu sonuçlar ışığında ara katmanın 4 elemanlı seçilmesi, uygun bir seçim olarak görülmektedir. Bir önemli konu da eğitilecek görüntü matrisinin boyutuyla (eleman sayısıyla) hatanın değişimini gözlemlemektir. Şekil 4,4'deki Hata-boyut değişiminde, görüntü matris boyutları arttıkça hatanın da arttığı görülmektedir. Bu nedenle görüntü bilgisinin kabul edilebilir değerlerdeki hata ile eğitilebilmesi için, görüntü boyutları da dikkate alınmalıdır. Eğitim işlemlerinin optimizasyonunda dikkat edilmesi gereken önemli bir faktör de, zaman faktörüdür. Uygun yapıda bir ağın seçiminde, ağın kabul edilebilir hataya ulaşabilmesi önemli olduğu gibi, bu hataya ulaşıncaya kadar geçen eğitim zamanı da önemlidir. Tekrar sayısı ile doğrudan ilişkili olan eğitim zamanı arttıkça hata belirli oranlarda azalmaktadır. Fakat bu azalımdaki değişim,

belirli bir zamandan sonra, küçük oranlarda oluşmaktadır. Verilen şekil 4,6'da bu değişim gösterilmiştir.

Geri yansımalı yapay sinir ağı kullanarak görüntü sıkıştırma işleminde eğitim zamanını azaltmak için yüksek hızlı mikroişlemciye sahip bilgisayarlar kullanılabilir. Böylece tanımlanan yöntemle, durgun görüntülerin sıkıştırılmasının yanısıra hareketli görüntü denilen video görüntülerinin de yüksek kalitede sıkıştırılıp gerçek zamanlı iletilmesi mümkün olabilecektir. Ayrıca yüksek hızlı bilgisayarlar geliştirildiğinde eğitim hatasını minimum seviyelere indirmek daha az zaman alacağı için ara katman eleman sayısı en küçük değerlerde alınarak sıkıştırma oranı artırılabilecektir. Bir de geri yansımalı yapay sinir ağı ile görüntü sıkıştırma yöntemi, diğer görüntü sıkıştırma yöntemleriyle birlikte kullanılarak düşük hata değeri ile yüksek sıkıştırma oranlı sistemler geliştirilebilecektir.

KAYNAKLAR DİZİNİ

Alonso,T., 1992, Digital image compression, Lehigh University,96p

Di Ruocco, N., Vitale, A. and Vitulano, S., 1992, Artificial intelligence in vision, 11 th International Conference on Pattern Recognition (IAPR), Netherlands, 453-456

Freeman, J.A. and Skapura, D.M., 1992, Neural network algoritms, applications, Addisson-Wesley, 401p

Haralik, R.M. and Shapiro, L.G., 1985, Computer and robot vision volume 1, Addison-Wesley Publishing

Haykin, S. and Dony, R.D., 1995, Neural network approaches to image compression, Proceedings of the IEEE,Vol. 83. No. 2, 288-303

Held, G., 1991, Data compression, Jhon Wiley. Sons, 475p

Hopfield, j.j. and Tank, D., 1985, Neural computation of decisions in optimizations problems, Biological Cybernetics, 52, 143-152

Jain, A.K., 1981, Image data compression: A review, Proc. IEEE, vol. 69, 349-389

Khoenen, T., 1990, The self-organizing map, Proc. IEEE, vol. 78, 1464-1480

Li, j. and Manikopoulos, C.N., 1990, Nonliner predictor in image coding DPCM, Elektron. Lett., Vol. 26, 1357-1359

Lim, J.S., 1994, Two-dimensional signal and image processing, 700p

Linde, Y., Buzo, A. and Gray, R.M., 1980, IEEE Trans. Commun., vol. C-28, 84-95

Lippmann,R.P., 1987, An introduction to computing with neural nets, IEEE ASP Magazine, vol. 4, 4-24

Nasrabadi, N.M. and King, R.A., 1988, Image coding using vector quantization: A review, IEEE Trans. Commun., vol. 36, pp. 957-971

KAYNAKLAR DİZİNİ (devam)

Öztemel, E., 1996, Bilgisayarda öğrenme ve yapay sinir ağları, Otomasyon Dergisi, 134-140s

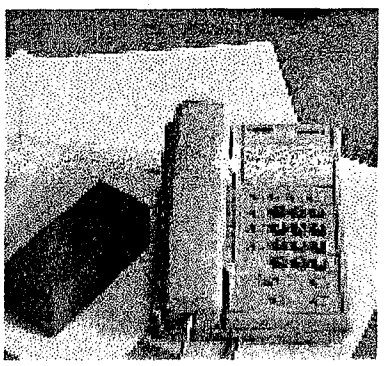
Sayood, K., 1996, İntroduction to data compression, Morgan-kaufmann-Publihers, 475p

Simpson, P., 1990, Artificial neural systems, Pergamon Press, .

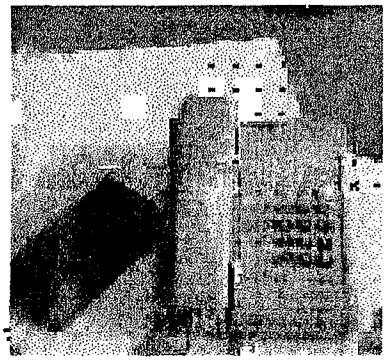
EKLER

128 x 128'lik çiçek orjinal görüntüsü

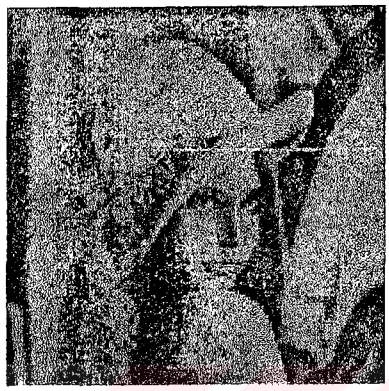
128 x 128'lik çiçek görüntüsünün sıkıştırıldıktan sonra elde edilen görüntüsü



128 x 128'lik orjinal telefon görüntüsü



128 x 128'lik telefon görüntüsünün sıkıştırıldıktan sonra elde edilen görüntüsü



128 x 128'lik lenna orjinal görüntüsü.

128 x 128'lik Lenna görüntüsünün sıkıştırıldıktan sonra elde edilen görüntüsü.

```
/*****************************************************************/
/*  Proram ismi: bp.c                                            */
/*                                                               */
/*****************************************************************/
#include <string.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "error.h"
#include "random.h"
#include <alloc.h>
#include <malloc.h>
#define GRAPH 0
#if GRAPH==1
#include"plot.h"
#endif
#ifdef ECO
#include<malloc.h>
#endif
#define U(x) (unsigned int)(x)
#define SQ(x) ((x)*(x))
/* fonksion prototipleri */
void getdata        (FILE *bp1,FILE *bp2);
void getpattern     (FILE *bp1,int,int,double *);
void allocate_memory (void);
void init_weights   (int,int,double *);
void learn          (long int);
void foreward       (int,int,double *,double *,double *);
void recognise      (void);
void calc_delta_o (int,int,double *,double *,double *);
void calc_delta_h (int,int,double *,double *,double *,double *);
void calc_descent (int,int,double,double,double *,double *,double *);
void correct_weight  (int,int,double *,double *);
double activate     (double);
double pattern_error (int,int,double *,double *);
void print_scale    (void);
void get_seed(void);
void get_limits(void);
void dump(int);
void geri_elde(void);
double *input;
double *output;
double *target;
double *weight_h;
double *weight_o;
```

```
double *hidden;
double *delta_o;
double *delta_h;
double *descent_h;
double *descent_o;
int n_pattern;
int n_input;
int n_hidden;
int n_output;
double learning_rate;
double momentum;
FILE *bp3;
FILE  *bp4;
int main()
{
  FILE *bp1;
  FILE *bp2;
        char buff[10];
  int choice;
        int p;
      long int cycles;
  if((bp1=fopen("parcbp00.dat","r"))==NULL){
    error(0,FATAL);
  }
  if((bp2=fopen("bp2.dat","r"))==NULL){
    error(1,FATAL);
  }
  if((bp3=fopen("bp3.dat","w"))==NULL){
    error(2,FATAL);
  }
  if((bp4=fopen("bp4.dat","wb"))==NULL){
    error(2,FATAL);
  }
 getdata(bp1,bp2);
        allocate_memory();
        getpattern(bp1,n_pattern,n_input,input);
              getpattern(bp1,n_pattern,n_output,target);
            get_seed();
      get_limits();
  init_weights(n_input,n_hidden,weight_h);
  init_weights(n_hidden,n_output,weight_o);
  for(;;){
 printf("\nBack Propagation Delta Kuralı öğrenme Programı\n"          printf("
   learning\n                  Recognise\n");
printf("        dump\n        quit\n   geri elde\n ");
    printf("Seçiminiz:");
    choice = getch();
    putchar(choice);
            switch(choice){
```

```
                case 'l':
                case 'L':
        printf("\nNe kadar çevrim yapılacak?\n");
                        cycles=atol(gets(buff));
                        if(cycles<1)cycles=1;
                        learn(cycles);
                        break;
                case 'g':
                case 'G':
                  geri_elde();
                        break;
                case 'r':
                case 'R':
                        recognise();
                        break;
                case 'd':
                case 'D':
                        for(p=0;p<n_pattern;p++)dump(p);
    printf("\n ağ değişkenleri bp3.dat dosyasına atılır");
                        break;
                case 'q':
                case 'Q':
                        fclose(bp1);
                        fclose(bp2);
                        fclose(bp3);
                        fclose(bp4);
                        exit(0);
                default:
    break;
   }
  }

}
    void getdata(
  FILE *bp1,
  FILE *bp2
)
{
  if(fscanf(bp1,"%d",&n_pattern)==EOF){
    error(3,FATAL);
  }
  if(fscanf(bp1,"%d",&n_input)==EOF){
    error(3,FATAL);
  }
if(fscanf(bp1,"%d",&n_hidden)==EOF){
          error(3,FATAL);
   }
if(fscanf(bp1,"%d",&n_output)==EOF){
    error(3,FATAL);
```

```
  }
if(fscanf(bp2,"%lf",&learning_rate)==EOF){
    error(4,FATAL);
  }
     if(fscanf(bp2,"%lf",&momentum)==EOF){
    error(4,FATAL);
  }
}
void allocate_memory()
{
  if((input=(double *)calloc(U(n_pattern*n_input),
  sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((target=(double *)calloc(U(n_pattern*n_output),
  sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((output=(double *)calloc(U(n_pattern*n_output),
  sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((hidden=(double *)calloc(U(n_hidden),sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((delta_h=(double *)calloc(U(n_hidden),sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((delta_o=(double *)calloc(U(n_output),sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((weight_h=(double *)calloc(U((n_input+1)*n_hidden),
  sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((weight_o=(double *)calloc(U((n_hidden+1)*n_output),
  sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((descent_h=(double *)calloc(U((n_input+1)*n_hidden),
  sizeof(double)))==NULL){
    error(6,FATAL);
  }
  if((descent_o=(double *)calloc(U((n_hidden+1)*n_output),
  sizeof(double)))==NULL){
    error(6,FATAL);
  }
     }
void getpattern(
```

```
  FILE *data,
  int n_pattern_vector,
  int n_units,
  double *matrix
)
{
  int p;
  int i;
  for(p=0;p<n_pattern_vector;p++){
    for(i=0;i<n_units;i++){
      if(fscanf(data,"%lf",matrix+(p*n_units+i))<=NULL){
        error(3,FATAL);
      }
    }
  }
}
void init_weights(
  int n_input_units,
  int n_output_units,
  double *weight
)
{
  int i;
  int j;
  for(j=0;j<n_output_units;j++){
    *(weight+j*(n_input_units+1)+n_input_units) = d_rand();
    for(i=0;i<n_input_units;i++){
    *(weight+j*(n_input_units+1)+i) = d_rand();
    }
  }
}
void learn(
      long int n_cycle
)
{
   long int learning_cycle;
      int p;
      int z;
      double eski,yeni;
      #if GRAPH==1
   double x_scale;
      double y_scale;
   #endif
   #if GRAPH==1
            init_graph();
   #else
      print_scale();
   #endif
   for(learning_cycle=0;learning_cycle<n_cycle;learning_cycle++){
```

```
            #if GRAPH!=1
            if(learning_cycle%10==0)printf(".");
            #endif

      if(learning_cycle==39999)
            printf(".");
      for(p=0;p<n_pattern;p++){
 foreward(n_input,n_hidden,weight_h,input+p*n_input,hidden);
 foreward(n_hidden,n_output,weight_o,hidden,output+p*n_output);
 calc_delta_o(p,n_output,delta_o,target,output+p*n_output);
 calc_delta_h(n_output,n_hidden,delta_o,delta_h,hidden,weight_o);
  calc_descent(n_hidden,n_output,learning_rate,momentum,
              descent_o,delta_o,hidden);
      calc_descent(n_input,n_hidden,learning_rate,momentum,
           descent_h,delta_h,(input+p*n_input));
       correct_weight(n_hidden,n_output,weight_o,descent_o);
      correct_weight(n_input,n_hidden,weight_h,descent_h);
     yeni=pattern_error(n_pattern,n_output,target,output);
            }
            #if GRAPH==1
            if(learning_cycle==0){
  set_scales(pattern_error(n_pattern,n_output,target,output),
                              n_cycle,&x_scale,&y_scale);
            }
            if((learning_cycle+1)%10==0){
        point(pattern_error(n_pattern,n_output,target,output),
                              learning_cycle+1,x_scale,y_scale);
            }
      #endif
      }
      #if GRAPH==1
      point(pattern_error(n_pattern,n_output,target,output),
      learning_cycle,x_scale,y_scale);
      close_graph();
      #endif
 printf("RMS error = %f ",
 pattern_error(n_pattern,n_output,target,output));
}
void recognise ()
{
  int p;
  int i;
  int k;
  for(p=0;p<n_pattern;p++){
  foreward(n_input,n_hidden,weight_h,(input+p*n_input),hidden);
   foreward(n_hidden,n_output,weight_o,hidden,output+p*n_output);
            printf("\ninput  ");
            for(i=0;i<n_input;i++){
                  printf(" %3.6f ",*(input+(p*n_input+i)));
```

```
    }
    printf("\n");
    printf("output  ");
    for(k=0;k<n_output;k++){
      printf("%f  ",*(output+p*n_output+k));
    }
    printf("\n");
    printf("target  ");
    for(k=0;k<n_output;k++){
      printf("%f  ",*(target+(p*n_output+k)));
    }
    printf("\n");
  }
 printf("RMS error = %f ",
 pattern_error(n_pattern,n_output,target,output));
  printf("         ilerlemek için herhangi bir tuşa basın\n");
  getch();
}
void foreward (
  int n_input_units,
  int n_output_units,
  double *weight,
  double *unit_in,
  double *unit_out
)
{
  int i;
  int j;
  double sum;
  for(j=0;j<n_output_units;j++){
    sum = 0.0;
    for(i=0;i<n_input_units;i++){
   sum = sum + (*(unit_in+i))*(*(weight+(j*(n_input_units+1)+i)));
       }
    sum = sum + (*(weight+(j*(n_input_units+1)+n_input_units)));
    *(unit_out+j) = activate(sum);
  }
}
double activate(
  double sum
)
{
  double activation;
            activation = 1.0/(1.0+exp(-sum));
  return activation;
}
          void calc_delta_o(
  int p,
  int n_output_units,
```

```
  double *delta,
  double *unit_target,
  double *unit_out
)
{
  int j;
  double temp;
        for(j=0;j<n_output_units;j++){
    *(delta+j) = ((*(unit_target+(p*n_output_units)+j))-
    (*(unit_out+j)))*
     (*(unit_out+j))*(1-(*(unit_out+j)));
       }
}
      void calc_delta_h(
  int n_output_units,
  int n_hidden_units,
  double *delta_out,
  double *delta_hid,
  double *unit_hid,
  double *weight
)
{
  int j;
  int k;
  double sum;
        for(j=0;j<n_hidden_units;j++){
    sum = 0.0;
    for(k=0;k<n_output_units;k++){
   sum = sum+(*(delta_out+k))*(*(weight+k*(n_hidden_units+1)+j));
    }
    *(delta_hid+j) = (*(unit_hid+j)) * (1-(*(unit_hid+j))) * sum;
  }
}
            void calc_descent(
  int n_input_units,
  int n_output_units,
  double rate,
  double moment,
  double *descent,
  double *delta,
  double *unit_in
)
{
  int i;
  int j;
        for(j=0;j<n_output_units;j++){
    for(i=0;i<n_input_units;i++){
*(descent+j*(n_input_units+1)+i) = rate*(*(delta+j))*(*(unit_in+i))+
         moment*(*(descent+j*(n_input_units+1)+i));
```

```
  *(descent+j*(n_input_units+1)+n_input_units) = rate*(*(delta+j))+
    moment*(*(descent+j*(n_input_units+1)+n_input_units));
  }
}
void correct_weight(
  int n_input_units,
  int n_output_units,
  double *weight,
  double *descent
)
{
  int i;
  int j;
  for(i=0;i<n_input_units+1;i++){
    for(j=0;j<n_output_units;j++){
 *(weight+j*(n_input_units+1)+i) = *(weight+j*(n_input_units+1)+i) +
      *(descent+j*(n_input_units+1)+i);
    }
  }
}
                double pattern_error(
  int n_pattern_vectors,
  int n_output_units,
  double *unit_target,
  double *unit_out
)
{
  int p,j;
  double temp;
  temp = 0.0;
  for (p=0;p<n_pattern_vectors;p++){
    for(j=0;j<n_output_units;j++){
    temp = temp + SQ((*(unit_target+(p*n_output_units)+j))-
              (*(unit_out+p*n_output_units+j)));
    }
  }
  return sqrt(temp);
}
void print_scale()
{
  printf("\n       100      200      300      400");
  printf("       500      600      700      800");
  printf("---------+---------+---------+---------+");
  printf("---------+---------+---------+---------+");
}
void  dump( int p )
{
  int i,j,k;
```

```
 if(fprintf(bp3,"\n")==NULL)
   error(5,WARN);
 if(fprintf(bp3,"input pattern no. %d\n",p)==NULL)
   error(5,WARN);
 if(fprintf(bp3,"\n")==NULL)
   error(5,WARN);
   if(fprintf(bp3,"input pattern\n")==NULL)
   error(5,WARN);
 for(i=0;i<n_input;i++){
   if(fprintf(bp3,"%f ",*(input+(p*n_input+i)))==NULL)
     error(5,WARN);
 }
 if(fprintf(bp3,"\n")==NULL)
   error(5,WARN);
            if(fprintf(bp3,"hidden pattern\n")==NULL)
error(5,WARN);
 for(j=0;j<n_hidden;j++){
   if(fprintf(bp3,"%f ",*(hidden+j))==NULL)
     error(5,WARN);
 }
 if(fprintf(bp3,"\n")==NULL)
   error(5,WARN);
     if(fprintf(bp3,"deltas for hidden pattern\n")==NULL)
   error(5,WARN);
 for(j=0;j<n_hidden;j++){
   if(fprintf(bp3,"%f ",*(delta_h+j))==NULL)
     error(5,WARN);
 }
 if(fprintf(bp3,"\n")==NULL)
   error(5,WARN);
            if(fprintf(bp3,"output pattern\n")==NULL)
   error(5,WARN);
 for(k=0;k<n_output;k++){
   if(fprintf(bp3,"%f ",*(output+p*n_output+k))==NULL)
     error(5,WARN);
 }
 if(fprintf(bp3,"\n")==NULL)
   error(5,WARN);
      if(fprintf(bp3,"deltas for output pattern\n")==NULL)
   error(5,WARN);
 for(k=0;k<n_output;k++){
   if(fprintf(bp3,"%f ",*(delta_o+k))==NULL)
     error(5,WARN);
 }
 if(fprintf(bp3,"\n")==NULL)
   error(5,WARN);
            if(fprintf(bp3,"target pattern\n")==NULL)
   error(5,WARN);
 for(k=0;k<n_output;k++){
```

```
    if(fprintf(bp3,"%f ",*(target+(p*n_output+k)))==NULL)
      error(5,WARN);
  }
  if(fprintf(bp3,"\n")==NULL)
    error(5,WARN);
      if(fprintf(bp3,"ara birimler i‡in a§□rl□klar\n")==NULL)
     error(5,WARN);
  for(j=0;j<n_hidden;j++){
    for(i=0;i<n_input+1;i++){
      if(fprintf(bp3,"%f ",*(weight_h+j*(n_input+1)+i))==NULL)
        error(5,WARN);
    }
  if(fprintf(bp3,"\n")==NULL)
    error(5,WARN);
  }
  if(fprintf(bp3,"ara  ağırlıklar için descentler \n")==NULL)
    error(5,WARN);
  for(j=0;j<n_hidden;j++){
    for(i=0;i<n_input+1;i++){
      if(fprintf(bp3,"%f ",*(descent_h+j*(n_input+1)+i))==NULL)
        error(5,WARN);
    }
  if(fprintf(bp3,"\n")==NULL)
    error(5,WARN);
  }
            if(fprintf(bp3,"weights to output units\n")==NULL)
    error(5,WARN);
  for(k=0;k<n_output;k++){
    for(j=0;j<n_hidden+1;j++){
      if(fprintf(bp3,"%f ",*(weight_o+k*(n_hidden+1)+j))==NULL)
        error(5,WARN);
    }
  if(fprintf(bp3,"\n")==NULL)
    error(5,WARN);
  }
  if(fprintf(bp3,"çıkış için descentler \n")==NULL)
    error(5,WARN);
  for(k=0;k<n_output;k++){
    for(j=0;j<n_hidden+1;j++){
      if(fprintf(bp3,"%f ",*(descent_o+k*(n_hidden+1)+j))==NULL)
        error(5,WARN);
    }
  if(fprintf(bp3,"\n")==NULL)
    error(5,WARN);
  }
}
void geri_elde()
     {   int p;
    int s;
```

```
      for(p=0;p<n_pattern;p++)
  {    for(s=0;s<n_output;s++)
     { // ch=fgetc(dosya);fprintf(dat,"%5.3f ",ch/256.0);
       if(fprintf(bp4,"%5.3f ",*(output+p*n_output+s))==NULL)
       error(5,WARN);
     }
     if(fprintf(bp4,"\n")==NULL)
     error(5,WARN);
  }
              }
            void get_seed()
{
  char buff[10];
          long int s;
  printf("\n\rBack Propagation Delta kuralı eğitim Programı\n\r");
  printf("Enter seed \n\rDefault = 1\n\rseed: ");
  s = atol(gets(buff));
  if(s<1) s = 1;
  s_seed(s);
  printf("\n\rseed = %ld",s);
}
           void get_limits()
{
     double upper,lower;
  char buff[10];
  printf("\n\r ağırlıklar için sınırları gir\n\r");
printf("defaults:\n\r  üst limit = 1.0\n\r   alt limit = -1.0\n\r");
  printf(" üst limiti gir: ");
  upper = atof(gets(buff));
  printf("alt limiti gir: ");
  lower = atof(gets(buff));
  if(lower>=upper){
    printf(" alt limit üst limitten büyük olamaz; ");
    printf("default selected\n\r");
    upper = 1.0;
    lower = -1.0;       }
  printf("üst limit = %4.2f\n\ralt limit = %4.2f",upper,lower);
     s_limits(upper,lower);        }
```

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <dos.h>
FILE *dosya,*dos;
struct palettetype pal;
unsigned int i,j,k,m,n;
unsigned char ch;
float deger;
char st[20];
void main()
{int gd=0,gm;
 initgraph(&gd,&gm,"");
 cleardevice();
 getpalette(&pal);
 for(i=0;i<pal.size;i++)setrgbpalette(pal.colors[i],i*4,i*4,i*4);

 st[0]='P';st[1]='A';st[2]='R';st[3]='T';st[6]='.';st[10]=0;
 for(n=0;n<4;n++)
 {if(n==0)st[4]='0';else if(n==1)st[4]='1';
  else if(n==2)st[4]='2';else st[4]='3';
 for(m=0;m<4;m++)
 {if(m==0)st[5]='0';else if(m==1)st[5]='1';
  else if(m==2)st[5]='2';else st[5]='3';
  st[7]='O';st[8]='U';st[9]='T';
  dosya=fopen(st,"rb");
  st[7]='X';st[8]='X';st[9]='X';
  dos=fopen(st,"wb");
  for(i=0;i<1024;i++)
  {fscanf(dosya,"%f",&deger);
   ch=deger*256;
   if((ch==9)||(ch==10)||(ch==13))ch=8;
   fputc(ch,dos);
  }
  fcloseall();
 }
}
for(n=0;n<4;n++)
{if(n==0)st[4]='0';else if(n==1)st[4]='1';
 else if(n==2)st[4]='2';else st[4]='3';
for(m=0;m<4;m++)
{if(m==0)st[5]='0';else if(m==1)st[5]='1';
 else if(m==2)st[5]='2';else st[5]='3';
 st[7]='X';st[8]='X';st[9]='X';
 dosya=fopen(st,"rb");
 st[7]='O';st[8]='O';st[9]='O';
 dos=fopen(st,"wb");
 for(k=0;k<8;k++)for(j=0;j<4;j++)for(i=0;i<8;i++)
 {fseek(dosya,4*32*k+4*j+16*i,SEEK_SET);
  ch=fgetc(dosya);fputc(ch,dos);
  ch=fgetc(dosya);fputc(ch,dos);
```

```
   ch=fgetc(dosya);fputc(ch,dos);
   ch=fgetc(dosya);fputc(ch,dos);
  }
  fcloseall();
 }
}
for(n=0;n<4;n++)
{if(n==0)st[4]='0';else if(n==1)st[4]='1';
  else if(n==2)st[4]='2';else st[4]='3';
 for(m=0;m<4;m++)
 {if(m==0)st[5]='0';else if(m==1)st[5]='1';
  else if(m==2)st[5]='2';else st[5]='3';
  st[7]='X';st[8]='X';st[9]='X';
  remove(st);
 }
}
outtextxy(25,18,"GERI ELDE EDILEN GORUNTU");
for(n=0;n<4;n++)
{if(n==0)st[4]='0';else if(n==1)st[4]='1';
  else if(n==2)st[4]='2';else st[4]='3';
 for(m=0;m<4;m++)
 {if(m==0)st[5]='0';else if(m==1)st[5]='1';
  else if(m==2)st[5]='2';else st[5]='3';
  st[7]='O';st[8]='O';st[9]='O';
  dos=fopen(st,"rb");
  for(j=0;j<32;j++)for(i=0;i<32;i++)
  {ch=getc(dos);putpixel(33*m+i+40,33*n+j+40,ch/16);}
  fcloseall();
 }
}

dosya=fopen("lena128.img","rb");
for(j=0;j<128;j++)for(i=0;i<128;i++)
{ch=fgetc(dosya);putpixel(300+i,40+j,ch/16);
}
fclose(dosya);



 getch();
 closegraph();
}
```

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <dos.h>
#include <ctype.h>
FILE *dosya,*dos;
struct palettetype pal;
int i,j,k,l;
char filename[20],st[7];
unsigned char ch;
void main()
{do
 {clrscr();
  printf("Okunup Parcalanmas□n□ istedigin dosya ad□ :");
  scanf("%s",&filename);
  dosya=fopen(filename,"rb");
  ch=0;
  if(dosya==0)
  {printf("Oyle bir dosya yok\n");
   printf("\nDevam etmek istermisin ?(E/H):");
   ch=toupper(getch());
   if(ch!='E')break;
  }
}while(dosya==0);
if(dosya!=0)
{int gd=0,gm;
 initgraph(&gd,&gm,"");cleardevice();getpalette(&pal);
 for(i=0;i<pal.size;i++)setrgbpalette(pal.colors[i],i*4,i*4,i*4);
 for(j=0;j<128;j++)for(i=0;i<128;i++)
 {ch=fgetc(dosya);putpixel(10+i,10+j,ch/16);}
 st[6]=0;st[0]='P';st[1]='A';st[2]='R';st[3]='C';
 for(l=0;l<2;l++)
 {if(l==0)st[4]='0';else if(l==1)st[4]='1';
  for(k=0;k<2;k++)
  {if(k==0)st[5]='0';else if(k==1)st[5]='1';
   dos=fopen(st,"wb");
   fseek(dosya,64*k+64*128*l,SEEK_SET);
   for(j=0;j<64;j++)
   {for(i=0;i<64;i++)
    {ch=fgetc(dosya);fputc(ch,dos);putpixel(200+i+68*k,10+j+68*l,ch/16);}
    if(!((l==1)&&(j==63)))fseek(dosya,64,SEEK_CUR);
   }
   fclose(dos);
  }
 }

 fclose(dosya);
}

getch();
closegraph();
```

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <dos.h>
FILE *dosya,*dat,*pro;
struct palettetype pal;
unsigned int i,j,k,l,n,m,p,r;
unsigned char ch;
char st[11];
void main()
{clrscr();
 TEKRAR:printf("N 1,2,4,8,16,32 degerlerinden biri olabilir\n");
 printf("N degerrini gir :");
 scanf("%u",&n);
 if(!((n==1)||(n==2)||(n==4)||(n==8)||(n==16)||(n==32)))
 {printf("Yanlis girdin. Tekrar gir\n\n");goto TEKRAR;}
 int gd=0,gm;
 initgraph(&gd,&gm,"");
 cleardevice();
 getpalette(&pal);
 for(i=0;i<pal.size;i++)setrgbpalette(pal.colors[i],i*4,i*4,i*4);

 st[0]='P';st[1]='A';st[2]='R';st[3]='C';

 for(r=0;r<2;r++)
 {if(r==0)st[4]='0';else if(r==1)st[4]='1';

  for(p=0;p<2;p++)
  {if(p==0)st[5]='0';else if(p==1)st[5]='1';

   st[6]=0;
   dosya=fopen(st,"rb");
   st[6]='.';st[7]='D';st[8]='A';st[9]='T';st[10]=0;
   dat=fopen(st,"wb");
   st[7]='P';st[8]='R';st[9]='O';
   pro=fopen(st,"wb");
   for(l=0;l<64/n;l++)for(k=0;k<64/n;k++)
  /* {for(m=0;m<=1;m++)*/
    {for(j=0;j<n;j++)
     {fseek(dosya,64*j+n*k+64*n*l,SEEK_SET);
      for(i=0;i<n;i++)
      {ch=fgetc(dosya);fprintf(dat,"%5.3f ",ch/256.0);
       if(m==0)fprintf(pro,"%5.3f\n",ch/256.0);
      }
     }
    fprintf(dat,"\n");
   }
  /* }*/
  fcloseall();
 }
}
```

```
closegraph();
}
```