İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞLARI KULLANILARAK EKG VERİLERİNİN SIKIŞTIRILMASI

YÜKSEK LİSANS TEZİ

Müh. Yücel KOÇYİĞİT

Tezin Enstitüye Verildiği Tarih : 10 Ocak 1996

Tezin Savunulduğu Tarih : 2 Şubat 1996

Tez Danışmanı : Doç. Dr. Mehmet KORÜREK

Diğer Jüri Üyeleri : Prof. Dr. Erdal PANAYIRCI

Doç. Dr. Cüneyt GÜZELİŞ

ŞUBAT 1996

ÖNSÖZ

Tez çalışmam sırasında bana her türlü yardımda bulunan danışman hocam Sayın Doç.Dr. Mehmet KORÜREK ve Sayın Yrd. Doç. Dr. Bekir KARLIK'a, arkadaşım Araş. Gör. S. Vakkas ÜSTÜN'e ve bana destek olan tüm arkadaşlarıma teşekkür ederim.

İstanbul, 1996 Yücel KOÇYİĞİT

İÇİNDEKİLER

ÖZET

Elektrokardiyografik işaretler için veri azaltma yöntemleri, işarette herhangi klinik bilgi kaybına neden olmadan, depolama, gönderme veya analiz işlemleri için veri hacminin küçültülmesi ve transmisyon kanal band genişliğinin küçültülmesi ile veri aktarım hızının arttırılmasına yönelik olarak son yıllarda artan bir ilgi ile kullanılagelmektedir. Bu nedenle son 30 yıl içerisinde elektrokardiyogramın (EKG) sıkıştırılması için birçok algoritma önerildi. Bu tezde güncel bir konu olan yapay sinir ağları (YSA) ile EKG verilerinin sıkıştırılması ve yeniden yapılandırılması yazılımla gerçekleştirildi. YSA'nın çalışma prensibinden yararlanarak eğitme aşamasında, EKG işaretleri YSA'nın hem girişine hem de çıkışına uygulandı. Böylece elde edilen ağırlık parametreleri gelecek yeni EKG verileri için baz oluşturdu. Sıkıştırma ve yeniden yapılandırma, iki ayrı ağırlık parametre seti için yapıldı.

Tezin ilk bölümünde, veri sıkıştırmanın gerekliliği ve çok kullanılan veri sıkıştırma teknikleri hakkında genel bilgi verildi.

İkinci bölümde çalışma da kullanılan yapay sinir ağlarının yapısı, fonksiyonları ve çeşitleri hakkında geniş bilgilere yer verildi.

Bulgular ve sonuçlar bölümünde çalışmadan elde edilen sonuçlar açıklandı. Son olarak, ek kısmında gerçekleştirilen yazılım verildi.

SUMMARY

ECG DATA COMPRESSION BY ARTIFICIAL NEURAL NETWORKS

The continuing proliferation of computerized electrocardiogram (ECG) processing systems along with the increased feature performance requirements and demand for lower cost medical care have mandated reliable, accurate, and more efficient ECG data compression techniques. The practical importance of ECG data compression has become evident in many aspects of computerized electrocardiography including: a)increased storage capacity of ECG's as databases for subsequent comparison or evaluation, b) feasibility of transmitting real-time ECG's over the public phone network, c) implementation of cost effective real-time rhythm algorithms, d)economical rapid transmission of off-line ECG's over public phone lines to a remote interpretation center, and e) improved functionality of ambulatory ECG monitors and recorders.

The main goal of any compression technique is to achieve maximum data volume reduction while preserving the significant signal morphology features upon reconstruction. Conceptually, data compression is the process of detecting and eliminating redundancies in a given data set. Redundancy in a digital signal exists whenever adjacent signal samples are statistically dependent and/or the quantized signal amplitudes do not occur with equal probability. However, the first step towards ECG data compression is the selection of minimum sampling rate and wordlength. Consequently, further compression of the ECG signal can be achieved by exploiting the known statistical properties of the signal.

Data compression techniques have been utilized in a broad spectrum of communication areas such as speech, image, and telemetry transmission. Data

compression methods have been mainly classified into three major categories: a) direct data compression, b) transformation methods, and c) parameter extraction techniques. Data compression by the transformation or the direct data compression methods contain transformed or actual data from the orginal signal. Whereby, the orginal data are reconstructed an inverse process. The direct data compressors base their detection of redundancies on direct analysis of the actual signal samples. In contrast, transformation compression methods mainly utilize spectral and energy distribution analysis for detection redundancies. On the other hand, the parameter extraction method is an irreversible process with which a particular characteristic or parameter of the signal is extracted. The extracted parameters (e.g., measurement of the probability distribution) are subsequently utilized for classification based on a priori knowledge of the signal features.

Existing data compression techiques for ECG signals lie in two of the three categories described: the direct data and transformation methods. The direct data compression techniques are: ECG differential pulse code modulation and entropy coding, AZTEC, Turning-point, CORTES, Fan and SAPA algorithms. Some of the transformation methods are: Fourier, Walsh and Karhunen Loeve transforms. Direct data compression techniques for ECG signals have shown a more efficient performance than the transformation techniques in regard particularly to processing speed and generally to compression ratio.

While it is done any comparison among ECG data compression techniques, the following properties are determined:

1) Signal sampling frequency (f_0):

In analog/digital converter utilized to convert ECG signals to digital the sampling frequency can be different frequencies in accordance with purpose but generally has been selected at 500 Hz.

2) Bit numbers in digital samples (p,"precision"):

Bit numbers which show resolution of stored ECG data can be 8 or 12 bits.

3) Compression ratio (CR):

This is one of the important parameters in data compression algorithms and the large value of this ratio shows success of any algorithm.

$$CR = \frac{\text{The number of samples before compression}}{\text{The number of samples after compression}} \qquad (1)$$

4) Performance index (PRD, "Percent Root Mean Square Difference"):

PRD is other one of the important parameters of any algorithms and the small value of PRD shows success of algorithm.

$$PRD = \sqrt{\frac{\sum_{i=1}^{n}\left[X_{org}(i) - X_{rec}(i)\right]^2}{\sum_{i=1}^{n}\left[X_{org}(i)\right]^2}} \cdot 100 \qquad (2)$$

where X_{org} and X_{rec} are samples of the orginal and reconstructed data sequences.

5) The higher speed of any algorithm is, the quicker the process can be done and real-time studies can be done.

6) Most of the databases utilized in evaluating ECG compression algorithms are nonstandart. However the algorithm results can be different in accordance with databases.

Let's determine artificial neural networks (ANN).

ANN is an information processing system which information spreads parallel on. This system consist of processing elements connected by single sided signal channels (connections).The number of output signals is one, but it can be increased. ANN can determine its conditions and adjust itself to provide different responses by using inputs and desired outputs which are given to the system.

Recently, it has been shown that neural networks have abilities to solve various complex problems. On the other hand, multilayered feedforward network has a better ability to learn the correspondence between input pattern and teaching values

from many sample data by the error backpropagation algorithm. Therefore, in this thesis, we used three-layered feedforward neural networks and taught them by error backpropagation. Fig. 1 shows a general structure of a neural network.

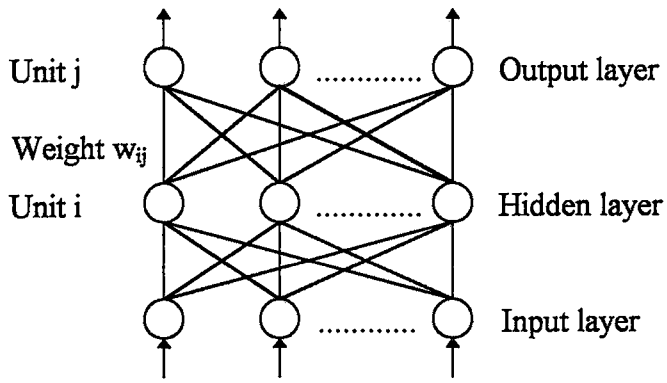


Fig.1. Ordinary type neural network.

The output o_j of each unit j is defined by,

$$o_j = f(net_j), \quad net_j = \sum_i^j w_{ji} o_i + \theta_j \tag{3}$$

where o_i is the output of unit i, w_{ji} is the weight of the connection from unit j to unit i, θ_j is the bias of unit j, Σ_i is a summation of every unit i whose output flows into unit j, and f(x) is monotonously increasing function. In practice, a logistic activation function f(x)=1/(1+exp(-x)) is used.

When the set of m-dimensional input patterns $\{i_p = (i_{p1}, i_{p2}, \ldots, i_{pm}); p \in P\}$ where P denotes set of presented patterns, and their correesponding desired n-dimensional output patterns $\{t_p = (t_{p1}, t_{p2}, \ldots, t_{pm}); p \in P\}$ are provided, the neural network is taught to output ideal patterns as follows. The squared error function E_p for a pattern p is defined by,

$$E_p = \frac{1}{2} \sum_{j \in \text{output layer}} (t_{pj} - o_{pj})^2 \tag{4}$$

The purpose is to make $E = \Sigma_p E_p$ small enough by choosing appropriate w_{ji} and θ_j. To realize this purpose, a pattern $p \in P$ is chosen successively and randomly, and the w_{ji} and θ_j are changed by

$$\Delta_p w_{ji} = -\varepsilon\left(\partial E_p / \partial w_{ji}\right) \tag{5}$$

$$\Delta_p \theta_j = -\varepsilon\left(\partial E_p / \partial \theta_j\right) \tag{6}$$

where ε is a small positive constant. By calculating the right hand side of (5) and (6), it follows that

$$\Delta_p w_{ji} = \varepsilon \delta_{pj} o_{pi} \tag{7}$$

$$\Delta_p \theta_j = \varepsilon \delta_{pj} \tag{8}$$

where

$$\delta_{pj} = \begin{cases} f'(net_j)(t_{pj} - o_{pj}) \text{ (when j belongs to the output layer.)} \\ f'(net_j)\sum_k \delta_{pk} w_{kj} \text{ (otherwise)} \end{cases} \tag{9}$$

Note that k in the above summation represents every unit k whose output follows into unit j. In order to accelerate the computation, the momentum terms are added on (7), (8),

$$\Delta_p w_{ji}(n+1) = \varepsilon \delta_{pj} o_{pi} + \alpha \Delta_p w_{ji}(n) \tag{10}$$

$$\Delta_p \theta_j(n+1) = \varepsilon \delta_{pj} + \alpha \Delta_p \theta_j(n) \tag{11}$$

where n represents the number of learning cycles, and α is a small positive value.

In this thesis, using artificial neural network (ANN) compression and reconstructed of the some ECG's are done. These ECG's are from MIT/BIH database tapes 100, 107 and 109. Tapes 100, 107 and 109 are respectively normal, paced beat and left bundle branch block ECG's. Orginal data sampled at 360 Hz and this sampling rate is decreased to 200 Hz in this thesis.

Compression processing which is run on computer with Pentium-75 microprocessor in Borland C is provided, as above mentioned, with 200 input nodes, 10 and 5 hidden nodes and 200 output nodes. At the same time, these values give compression ratio, i.e. 20:1 and 40:1. Therefore the utilized ANN structures are as 200:10:200 and 200:5:200. Generally, 20 cycles from ECG's are used in training

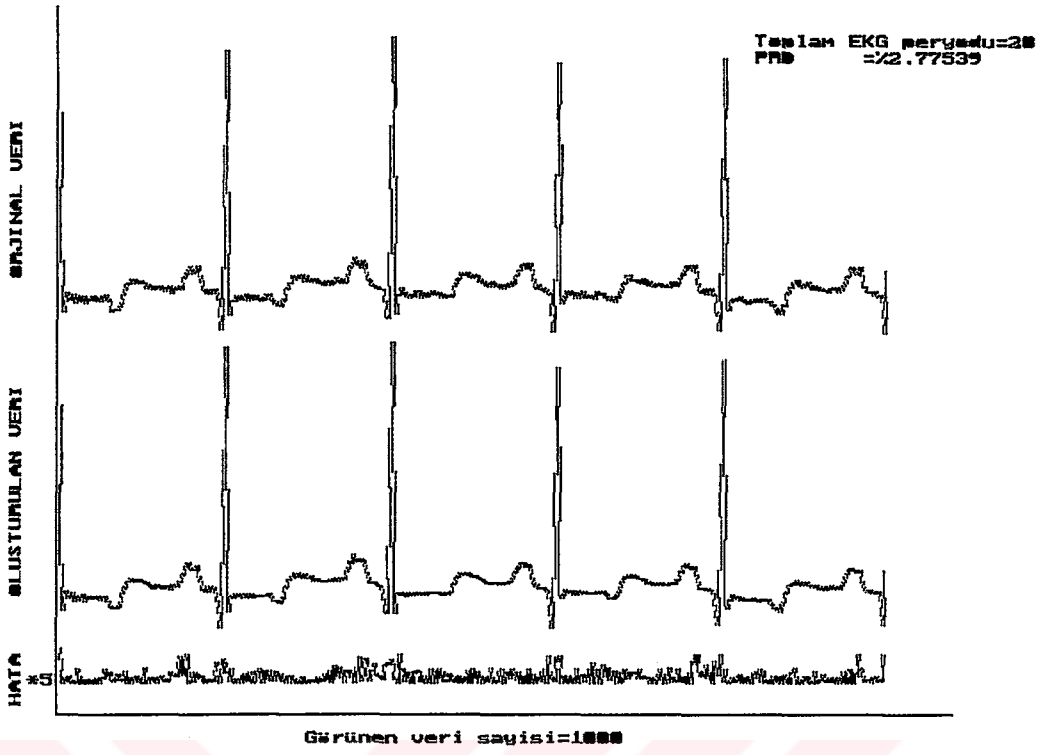


Fig. 2. Graphics of the orginal and reconstructed ECG from MIT/BIH tapes 100 for 10 hidden nodes, and graphics of difference (5 times) between them

process. As above mentioned, learning coefficient (ε) and momentum (α) must be selected to minimize the error in training process.

Reconstructed processing for compressed ECG is provided with 10 input nodes (because of 10 hidden nodes in compression) or 5 input nodes (because of 5 hidden nodes in compression) and 200 output nodes. Number of hidden nodes are selected to minimize the error along with ε and α. For normal ECG, orginal and reconstructed ECG is shown in Fig 2. An advantage of this technique is to compress large numbers of data, i.e. 10 cycles (samples) from ECG's are used for training and 100 cycles (samples) can be compressed.

BÖLÜM 1

GİRİŞ

Elektrokardiyogram (EKG) kalbin elektriksel aktivitesini sunan bir grafiktir. EKG işareti, vücut yüzeyinden elektrotlarla alınır ve teşhis etme amacına yönelik olarak doktorlara yardımcı olur. Elektrokardiyografik işaretler için veri azaltma yöntemleri, işarette herhangi klinik bilgi kaybına neden olmadan, depolama, gönderme veya analiz işlemleri için veri hacminin küçültülmesi ve transmisyon kanal band genişliğinin küçültülmesi ile veri aktarım hızının arttırılmasına yönelik olarak son yıllarda artan bir ilgi ile kullanılagelmektedir.

Öznitelik performans geneksinimlerinin artmasıyla ve düşük fiyatlı medikal tedavi talebiyle bilgisayarlaşmış EKG işleme sistemlerinin hızla çoğalması, güvenilir, doğru ve daha verimli EKG veri sıkıştırma tekniklerini mecbur kılar. EKG veri sıkıştırmanın uygulamadaki önemi, bilgisayarlaşmış elektrokardiyografinin aşağıdaki durumları için açıktır:

a) Sonraki karşılaştırma veya geliştirmeler için veri tabanı olarak EKG' lerin artan bellek kapasitesinin karşılanması,

b) Genel telefon şebekesi üzerinde EKG'leri gerçek zamanda gönderme imkanı,

c) Maliyeti açısından etkin gerçek-zaman ritim algoritmalarının gerçekleştirilmesi,

d) Genel telefon şebekesi üzerinden hat-dışı EKG'lerin, uzaktan, yorumlama merkezine ekonomik ve hızlı olarak gönderilmesi,

e) Gezici ("Ambulatory") EKG monitörlerinin ve kaydedicilerinin fonksiyonlarını geliştirme.

Herhangi bir sıkıştırma tekniğinin hedefi, işaretin morfolojik özniteliklerini, yeniden yapılanma için saklarken maksimum veri miktarı azalımını sağlamaktır. Kavramsal olarak , veri sıkıştırma, verilen bir veri seti içerisindeki gereksiz verileri sezme ve eleme işlemidir. Veri sıkıştırmada temel ilke, yeni işaret örneği bir öncekine bağımlı olur olmaz ve/veya örnekler arası olasılıklar eşit olur olmaz, dijital işarette farklılıklar varolur. Ancak EKG veri sıkıştırmada ilk adım minimum örnekleme hızının ve kelime uzunluğunun seçimidir. Bu nedenle, EKG işaretlerinin ileri derece sıkıştırılması, işaretin bilinen istatistik özellikleri işletilerek sağlanabilir.

Cox ve arkadaşları, 1968, ritim analizlerinde gerçek zamanlı EKG'lerin önişlemesi için AZTEC (ampitude zone time epoch coding) algoritmasını geliştirdi. Bu algoritma, EKG izleme ve veri tabanları için kullanışlı bir veri azaltma algoritması oldu. Ancak yeniden yapılanan işaret açık bir şekilde süreksizlik ve distorsiyon gösterir. Özellikle işaret distorsiyonlarının çoğu, yavaş değişen eğimlerinden dolayı P ve T dalgalarının yeniden yapılanmasında oluşur. 200 Hz, 12 bit örnekleme şartları için sıkıştırma oranı (CR) 10 ve performans indeksi (PRD) %28 elde edilmiştir.

Mueller, 1978, büyük genlikli QRS'lerin genliğini azaltmadan 200 Hz'lik EKG işareti örnekleme frekansını 100 Hz'e indirgeyen dönüm noktası tekniğini (TP,"turning point") geliştirdi. TP yöntemi 2:1 oranında sabit bir sıkıştırma oranı sağlar. TP yönteminin dezavantajı, yeniden yapılanma için sakladığı noktaların eşit aralıklarla dizilmiş zaman aralıklarını temsil edememesidir. 200 Hz, 12 bit örnekleme şartları için PRD %5,3 elde edilmiştir.

Tompkins ve arkadaşları, 1982, CORTES adı verilen AZTEC ve TP algoritmalarının karmasından oluşan algoritmayı geliştirdiler. Bu algoritma heriki yöntemin üstünlüklerini içeriyordu. 200 Hz, 12 bit örnekleme şartları için CR 4,8 ve PRD %7.0 elde edilmiştir.

Gardenhire, 1964, FAN adı verilen interpolasyon algoritmasını geliştirdi. Bu yöntemde gönderilen en son nokta ile işlenen nokta arasındaki bütün gerçek verileri

depolamaya ihtiyaç bırakmayıp belirlenmiş bir hata toleransı içerisindeki orta örneklerden oluşan başlangıç ve bitiş noktalarını en uzun çizgiyle birleştirir.

Ishijima ve arkadaşları, 1983, EKG işarelerini düz-çizgi segmentleriyle sunan üç SAPA (scan-along polygonal approximation) tekniğini ileri sürdüler. Bunlardan en iyi sonucu veren SAPA-2 tekniği büyük oranda FAN tekniğine benziyordu. Aradaki tek fark FAN iki eğim hesabı yaparken SAPA-2 üç eğim hesabı yapıyordu. 250 Hz örnekleme için CR 3.0 ve PRD %4.0 elde edilmiştir.

Wolf ve arkadaşları, 1972, delta kod sistemini önerdiler. Delta kod algoritması, ard arda gelen örnekler arasındaki genlik farkının, örnek genliklerinin kendisinden daha küçük olması esasına dayanır. Bu yöntemde lineer prediktörler ve interpolatörler kullanılmaktadır. Ancak bunların kullanılmasının daha verimli olup olmayacağı bilinmemesine rağmen ikinci dereceden daha yüksek derecelileri kullanıldığında EKG veri sıkıştırmada belli bir artış kaydetmeyeceğini araştırmacılar göstermişlerdir. 300 Hz, 8 bit örnekleme şartları için CR 4.0 elde edilmiştir.

Huffmann, 1952 yılında, entropi kodlamasının temelini atmıştır. Bu teknikte EKG sabit uzunluktaki kod kelimesiyle kodlama yerine değişik uzunluktaki kod kelimelerine kodlanır. Değişik uzunluktaki kodlamanın dezavantajı, aktarma hatalarından dolayı ciddi kod çözme hatalarına sahip olabilmesidir. 500 Hz, 8 bit örnekleme şartları için CR 7.8 ve PRD %3.5 elde edilmiştir.

1960'ların başlarında transformasyon yöntemiyle pekçok sıkıştırma tekniği ortaya atıldı: Fourier (FT), ortonormal eksponansiyeller ve Karhunen Loeve Transformu (KLT). Daha sonra KLT ve FT 'nin [1] kullanılmaları üzerine çalışmalar yapıldı. Haar Transformu (HT) ve Kosinüs Transformu (CT) bile [2]'de kullanıldı. Bu EKG transformasyon teknikleri arasında, çok-uçlu EKG verileri için en yüksek sıkıştırma oranı KLT kullanılarak sağlandı.

Ahmed ve arkadaşları [2], KLT, CT, ve HT'yi tek-uçlu EKG'lere uyguladı. KLT'nin diğerlerine göre 400 Hz örnekleme frekansı için en yüksek sıkıştırma

oranını(3:1) verdi. Reddy ve Murthy [1] ortogonal iki (X,Y) EKG uçlarının sıkıştırması için iki boyutlu FT' yi çalıştırdılar. 12 bit doğruluklu 250 Hz'te örneklenmiş EKG için sıkıştırma oranı 7,4:1 olarak gerçekleşti.

Bu çalışmada yapay sinir ağları ile yeni bir sıkıştırma algoritması denendi. 200Hz'de örneklenmiş EKG'lerin herbiri için ağırlık parametreleri hesaplandı ve buna göre EKG'ler yeniden yapılandırıldı. Sıkıştırma oranı 20:1 ve 40:1 olarak gerçekleştirildi.

BÖLÜM 2

VERİ SIKIŞTIRMA TEKNİKLERİ

Veri sıkıştırma metodları üç ana kategoriye ayrılabilir:

a) direkt veri sıkıştırma,

b) transformasyon metodları,

c) parametre çıkarma teknikleri.

Transformasyon veya direkt veri sıkıştırma yöntemleriyle yapılan veri sıkıştırma, orjinal işaretten transfer edilmiş veya gerçek verileri içerir. Buna karşılık, orjinal veri ters işlemlerle yeniden yapılandırılır. Direkt veri sıkıştırıcılar, gerçek işaret örneklerinin direkt yöntemlerle analizleri yapılırken fazlalıkların tespit edilmesi prensibiyle çalışır. Buna karşılık, transformasyon sıkıştırma yöntemleri, fazlalıkları tespit için spektral ve enerji dağılım anallizlerini kullanırlar. Diğer taraftan, parametre çıkarma yöntemi işaretin özel karakteristiğinin veya parametresinin çıkarıldığı tersinmez ("irreversible")bir işlemdir. Çıkarılan parametreler (örneğin olasılık dağılımı) daha sonra işaretin özniteliklerinin priori bilgisi üzerine dayanan sınıflama için kullanılır.

EKG işaretleri, veri sıkıştırma tekniklerindeki bu üç kategoriden ikisiyle sıkıştırılır: direkt veri sıkıştırma ve transformasyon metodları. EKG işaretleri için direkt veri sıkıştırma teknikleri, transformasyon tekniklerinden özellikle işleme hızı ve genellikle sıkıştırma oranı bakımından daha verimli performans gösterir.

EKG veri sıkıştırma teknikleri arasında karşılaştırma yapılırken aşağıda sıralanan özellikler dikkate alınmaktadır[3]:

1) İşaret örnekleme frekansı (f_0): EKG işaretlerini sayısala çevirmekte kullanılan analog/sayısal çeviricilerde örnekleme frekansı amaca göre çeşitlilik gösterip yaygın olarak 500Hz'lik örnekleme frekansı kullanılmaktadır.

2) Sayısal örneklerdeki bit sayısı (p): Saklanacak EKG sayısal bilgilerinin çözünürlüğünün (rezolüsyonunun) göstergesi olan bit sayısı 8 veya 12 olabilmektedir.

3) Veri sıkıştırma oranı (CR, "compression ratio"): Veri sıkıştırma algoritmasının önemli parametrelerinden biri olup bu oranın büyüklüğü, algoritmanın üstünlüğünü gösterir.

$$CR = \frac{\text{Sıkıştırılacak örneklerin sayısı}}{\text{Sıkıştırılmış örneklerin sayısı}} \qquad (2.1)$$

4) Performans indeksi (PRD, "percent rootmean square differrence"): Orjinal EKG işareti ,X_{org}, ile sıkıştırılmışının yeniden yapılanması, X_{rec} ,(yeniden yapılandırılmışı) arasındaki farkın bağıl karesel ortalamasının kare kökü olarak tanımlı olup yeniden yapılanmış (oluşturulmuş) olan EKG işaretinin orjinaline ne derece benzer olduğunun ölçüsü olarak kullanılmaktadır. PRD, veri sıkıştırma algoritmalarının önemli karşılaştırma parametrelerinden bir diğeri olup PRD'nin küçüklüğü algoritmanın başarı derecesini gösterir.

$$PRD = \sqrt{\frac{\sum_{i=1}^{n}\left[X_{org}(i) - X_{rec}(i)\right]^2}{\sum_{i=1}^{n}\left[X_{org}(i)\right]^2}} \cdot 100 \qquad (2.2)$$

5) Algoritmanın hızı ne kadar yüksekse o oranda hızlı işlem yapabilir ve gerçek zaman çalışmaları gerçekleştirilebilir.

6) EKG sıkıştırma algoritmalarında kullanılan veri tabanlarının çoğu standart dışıdır. Oysa kullanılacak veri tabanlarına göre de algoritma sonuçları farklı olabilmektedir.

2.1. Direkt Veri Sıkıştırma Yöntemleri

2.1.1. Değişken Eşikli Adaptif Algoritma Tekniği

Sözkonusu tekniğin akış mantığı aşağıdaki adımlarla anlatılabilir:

a) Bu algoritma, işlenmemiş EKG verilerini ele alıp AZTEC algoritmasında olduğu gibi kısa çizgiler üretir [3], [4]. Orjinal işaret örnek dizisi ,X_i (i=1,2,...) ile ve örneklerin maksimum ve minimum değerleri ise X_{maks} ve X_{min} ile belirtilsin. Başlangıç X_{maks} ve X_{min} değerleri, ilk işaret örneği olan X_1'e eşit kılınır. X_2, X_3 gibi sonraki örnekler devamlı olarak X_{maks} ve X_{min} değerleri ile karşılaştırılırlar. Eğer yeni örnek X_{maks}'tan büyük ise bu örnek X_{maks}'a atanır, yok eğer bu örnek X_{min}'den küçükse X_{min}'e atanır. Bunun dışındaki durumda X_{maks} ve X_{min} değerleri değiştirilmez. Bu işlem, X_{maks}-X_{min} farkının T gibi bir eşik değerinden büyük olmasına kadar devam edilir [5].

b) Sözkonusu X_{maks}-X_{min} > T koşulu sağlandığında çizgi, veri ($\overline{X}$,t) ikilisi kullanılarak kaydedilir. Burada $\overline{X}$, X_i örneğinin hesaplanmış ortalama genlik değeri ve t ise çizginin uzunluğu yani o bölgedeki örnek sayısını göstermektedir.

c) T eşiği, işaretin tabiatına bağlı olarak değişir. Tabanhattı gibi yavaş değişim gösteren bölgelerdeki eşik değerleri, P, T, ST, QRS gibi hızlı değişim gösteren bölgelerdeki eşik değerlerinden daha yüksek olacaktır. Böylece algoritma, bir çeşit kendinden adaptasyon içermektedir.

d) Böyle bir bilgi-bölgesinden diğerine atlayabilmek için algoritma gerçek zamanda bir takım istatistiki bilgileri hesaplar. Bunları şöyle sıralayabiliriz:

1) Ortalama değer;

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \qquad (2.3)$$

Burada X_i, sıkıştırılacak orjinal işaret örnekleridir.

2) İşaretlerin standart sapması σ ;

$$\sigma = \left[\frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2\right]^{1/2} \qquad (2.4)$$

3) İşaretin üçüncü momenti M;

$$M=\left[\frac{1}{n}\sum_{i=1}^{n}\left(X_i-\overline{X}\right)^3\right]^{1/3} \qquad (2.5)$$

Standart sapma ve 3. moment hesapları, yavaş değişim bölgelerinden hızlı değişim bölgelerine geçişi daha hassas olarak fark etmek amacıyla hesaplanmaktadırlar. Ancak gerçek zaman çalışmalarında bu parametrelerin dinamik olarak rekürsif (ardışık) biçimde hesaplanmaları gerekir. Bunun için de aşağıdaki formüller çıkarılmıştır:

1) Ortalama değer:

$$\overline{X}_k=\frac{(k-1)\overline{X}_{k-1}+X_k}{k} \qquad (2.6)$$

2) Standart sapma:

$$\sigma_k=\left[\frac{(k-1)\sigma^2_{k-1}+\left(X_k-\overline{X}_k\right)^2}{k}\right]^{1/2} \qquad (2.7)$$

3) Üçüncü moment:

$$M_k=\left[\frac{(k-1)M^3_{k-1}+\left(X_k-\overline{X}_k\right)^3}{k}\right]^{1/3} \qquad (2.8)$$

e) Her örnekten sonra bir kriter fonksiyonu KF_i aşağıdaki şekilde hesaplanır:

$$KF_k=C_1\,(\sigma_k+M_k) \qquad (2.9)$$

Daha sonra bu kriter fonksiyonuna dayanarak, devamlı yenilenecek şekilde eşik değeri hesap edilir:

$$T_k=T_{k-1}-C_2\,(kF_k-KF_{k-1})\,T_{k-1} \qquad (2.10)$$

C_1 ve C_2 sabitleri deneylerden edinilen tecrübeye göre 1 ve 0,008 civarında seçilebilirler. Eşik değeride (T_{min}, T_{maks}) gibi sınırlar arasında tutulabilir.

Böylece standart sapma veya 3. momentteki herhangi bir değişme eşik değerine yansımış ve yavaş değişim bölgeleri büyük eşik değerlerine sahip olmuş ve sıkıştırma oranı yükselmiş olacaktır.

Sıkıştırma algoritması, belli bir başlangıç eşik değeri ile başlar ve sonra (2.9) ve (2.10) eşitlikleri, her örnekten sonra T'yi yenilemek amacıyla $(X_{maks}-X_{min}) > T_k$ oluncaya kadar hesaplanır. Belirtilen eşitsizlik koşulu sağlandığında $(\overline{X},k)$ değerleri yardımıyla çizgi saklanır ve işlem devam eder.

Yeniden Yapılanma ("recontruction") Algoritması:

Sıkıştırılmış veri, $(\overline{X},k)$ gibi veri çiftleri dizisinden oluşmuştur. Her çift, çizginin genlik ve uzunluğunu belirtir. Örneğin: (42,5) = 5 adet 42 birim genlikli örnekten oluşan çizgiyi temsil etmektedir. Veri bu şekilde açılır ve gözlenirse, bu, kabul edilebilir bir görüntü oluşturmaz. Onun için bir eğri yumuşatma algoritması bu diziye uygulanır. Standart en az karesel polinom eğrisi uydurma tekniği burada yumuşatma filtresi olarak uygulanır. Bu teknik, en iyi uygunluğu, işaretin her biri 7. örnekten oluşan setlerine tatbik edilmesiyle en iyi sonucu verir.

Burada iki adım mevcuttur:

1) Elde edilmiş doğru çizgi verileri alınıp ayrım veri noktaları şekilde açılır;

Örneğin: (42,5) = (42, 42, 42, 42, 42) gibi.

2) Her yedi noktaya yumuşatma filtresi uygulanır;

$$y_k = \frac{1}{21}\left(-x_{k-3} + x_{k-2} + 6x_{k-1} + 7x_k + 6x_{k+1} + 3x_{k+2} - 2x_{k+3}\right) \qquad (2.11)$$

Burada y_k, yeni veri noktası ve X_{k+n}, orjinal veri noktalarını temsil etmektedirler.

2.1.2. AZTEC Tekniği ("Amplitude Zone Time Epoch Coding"):

Esasında AZTEC, yukarıda anlatılan değişken eşikli adaptif algoritmanın bir alt algoritması olarak kabul edilebilir. İkisi arasındaki tek fark AZTEC 'te belirlenen eşiğin, işlem boyunca değişmeden sabit kalması ve dolayısıyla standart sapma ve üçüncü moment gibi hesapların yapılmasına gerek kalmamasıdır.

AZTEC algoritması, orjinal EKG örnek noktalarını yatay ve eğimli çizgilere dönüştürür[4]. AZTEC platosu (yatay çizgiler), sıfırıncı mertebe interpolatörü (ZOI) kullanılarak üretilirler. Her platonun bellekte saklanan değerleri, çizginin genlik değerleri ve uzunluğudur. Bir AZTEC eğiminin başlaması için (plato oluşturabilmek için) eşik kapsamındaki örnek sayısının 3'ten az olması lazımdır. Eğimin saklanan değerleri, süresi ve son örnek noktasının genliğidir. İşaretin yeniden yapılanması, AZTEC plato ve eğimlerine ait veri noktalarının ayrık dizilere açılmasıyla elde edilir.

Yeniden yapılanan EKG'deki süreksizliklerden ötürü AZTEC, yüksek oranda veri azaltması sağlasa da, kardiyologlar tarafından kabul edilecek düzeyde değildir. Bu süreksizliklerin büyük oranda azaltılması, yumuşatma yapan bir parabolik filtrenin kullanılmasıyla olur. Bu işlemin kusuru ise EKG dalga biçimine genlik distorsiyonun katılmasıdır.

Bu durumda, AZTEC algoritmasının ZOI kısmı için kullanılan hata eşiğinin değişen EKG işaretlerine adapte edilmesiyle elde edilen yeni modifiye AZTEC algoritması önerilmiştir [5]. Bu adaptiflik, işaretin ilk üç saniyesinin ardışık (recursive) hesaplanması temeline dayandırılmıştır. Bu teknik, ortalama karesel fark kökü yüzdesine (PRD) bakıldığında, aynı sıkıştırma oranı (CR) için, AZTEC algoritmasına göre daha üstündür. Süreksizliği azaltmak amacıyla AZTEC'e dayalı diğer bir teknik geliştirilmiştir. Platoları üretmek üzere ZOI kullanmak yerine, eğim çizgilerini oluşturmak üzere Fan tekniği kullanılmıştır. Daha sonra EKG işareti, bu eğim çizgileri ve AZTEC eğimlerine bağlanarak tekrar yapılanır. Bu teknikle ilgili hesaplamalar, AZTEC algoritması ile karşılaştırıldığında göstermiştir ki, sıkıştırma oranında bir artış ve işaret performans indeksinde (PRD) %50'lik bir iyileşme olmuştur.

2.1.3. Dönüm Noktası Tekniği (TP, "Turning Point"):

Bu teknik, büyük genlikli QRS 'lerin genliğini azaltmadan 200 Hz 'lik EKG işareti örnekleme frekansını 100 Hz 'e indirmeyi amaçlamıştır [6].

Algoritma aynı anda 3 veri noktasını işler; X_0 referans veri noktası ve bunu takip eden X_1 ve X_2 veri noktaları. Ya X_1 yada X_2 saklı tutulacaktır. Bu ise hangi noktanın orjinal 3 noktanın eğimini koruduğuna bağlıdır. Bunu matematiksel bir dille şöyle ifade edilebilir:

$$(X_2 - X_1)\cdot(X_1 - X_0) < 0 \text{ ise } X_0 = X_1$$

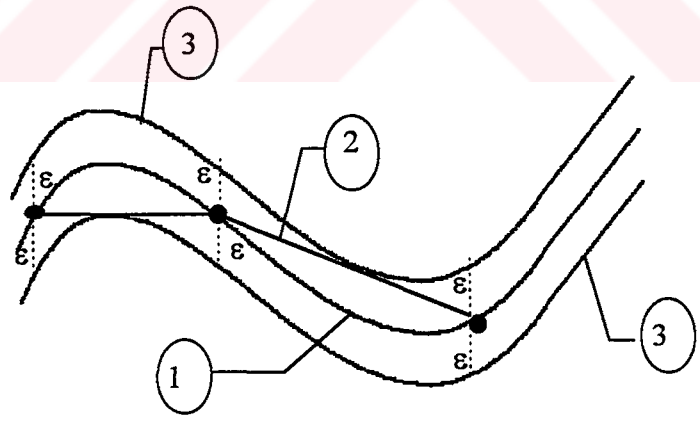
$$(X_2 - X_1)\cdot(X_1 - X_0) \geq 0 \quad \text{ise} \quad X_0 = X_2 \quad \text{olarak}$$

tutulacaktır.

TP algoritması 2:1'lik sabit bir sıkıştırma oranı sağlar ki burada yeniden yapılanan işaret orjinal işarete bazı gürültüleri içerdiği halde benzer. TP yönteminin bir kusuru, kayıt edilmiş noktaların eşit aralıklarla dizilmiş zaman aralıklarını temsil etmemesidir.

2.1.4. SAPA Teknikleri ("Scan Along Polygonal Approximation"):

Üç adet SAPA (Poligonal Yaklaşım İzleme) algoritması vardır. SAPA-2 bunlardan en iyi sonuç verenidir. Bu algoritmanın teorik temeli, doğru çizgiler (approximated) ile orjinal işaret arasındaki sapmanın, hata toleransından hiçbir zaman



Şekil 2.1. SAPA tekniğinin ana fikri.

1-Orjinal İşaret

2-Orjinal İşaretin Poligonal Yaklaşımı

3-Orjinal Hata Sınırı

büyük olmamasıdır. SAPA-2 ve Fan algoritmaları arasındaki tek fark, SAPA-2'nin orijin örnek noktası ve gerçek sonraki örnek noktsı arasında 3. bir eğim (merkez eğim, center slope) hesaplamasıdır. Merkez eğim, iki nokta arasında değişen işaretin belli hata sınırları dışına taştığı zaman, sondan bir önceki örnek noktası kalıcı (sürekli) örnek noktası olarak dikkate alınır. Başka bir deyişle, SAPA-2 algoritması, örneğin kalıcı mı yoksa geçici mi olduğunu belirlemek için merkez-eğim kriterini kullanır (Şekil2.1.). Oysa Fan'da, gerçek örnek değeri kriteri kullanılmaktadır [6].

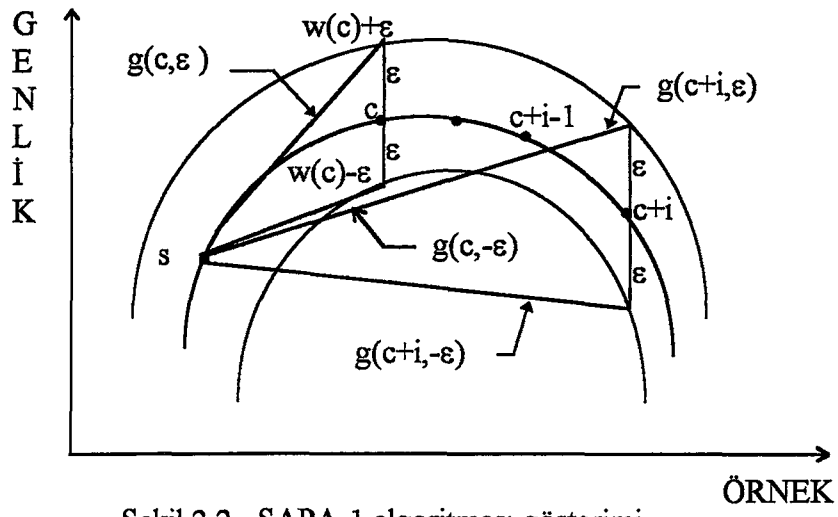
2.1.4.1. SAPA-1

Orjinal örneklenmiş veri w(k) gibi ayrık dizi şeklinde gösterilsin. Burada k örnek numarası olsun. ε, kullanıcı tarafından belirlenecek olan hata aralığını temsil etsin. Tekrar yapılanmış işaret $\hat{w}(k)$ ayrık dizisi ile gösterilsin, öyle ki bu dizi $|w(k)-\hat{w}(k)| \le \varepsilon$, k=1,2,3,... şartını sağlasın. Orjinal veri (k,w(k), k=1,2,3,...) ile ve azaltılmış veri ise (k, $\hat{w}(k)$, k=s_1 ,s_2, ...) şeklinde temsil edilsin. (Örnek numarası genlik) çifti şeklinde gösterilen bu azaltılmış veriye orjinal verinin verteksleri denir. İki verteks arası düz bir çizgi çekmek suretiyle yaklaşıklık yapılır.

Algoritmaya başlarken ilk veri noktası verteks olarak seçilir. k=c'deki, daha sonra gelen örneğin alınmasından sonra normalize edilmiş iki doğrunun eğimleri hesaplanır.
Bunlar;

$$g(c,\varepsilon) = \frac{w(c)+\varepsilon - w(s)}{c-s}$$
$$g(c,-\varepsilon) = \frac{w(c)-\varepsilon - w(s)}{c-s} \qquad (2.12)$$

şeklindedir. Şekil 2.2.'de bu eğimler gösterilmiştir. Her yeni veri noktası işlendiğinde, o anki en küçük $g(c,\varepsilon)$ m_1 ve en büyük $g(c,-\varepsilon)$ değeri m_2 olarak saklanır. Böylece m_1 ve m_2, s'ten gelen ve $\pm\varepsilon$ aralığı içinde bulunan doğruların maksimumu ve minimum eğimlerini göstermiş olur. Ne zaman $m_2 > m_1$ koşulu sağlanırsa (Şekil 2.1.'de k=c+i noktasında olduğu gibi), hemen bir önceki veri noktası yeni verteks olarak seçilir [7].



Şekil 2.2. SAPA-1 algoritması gösterimi

2.1.4.2. SAPA-2

Bu algoritma SAPA-1'in biraz daha iyileştirilmişi olup, ekstra bir kontrol mekanizması eklenmesiyle elde edilmiştir. Burada g(c,ε) ve g(c,-ε) eğimlerine ek olarak,

$$g(c,0) = \frac{w(c) - w(s)}{c - s} \qquad (2.13)$$

hesaplaması katılmıştır.

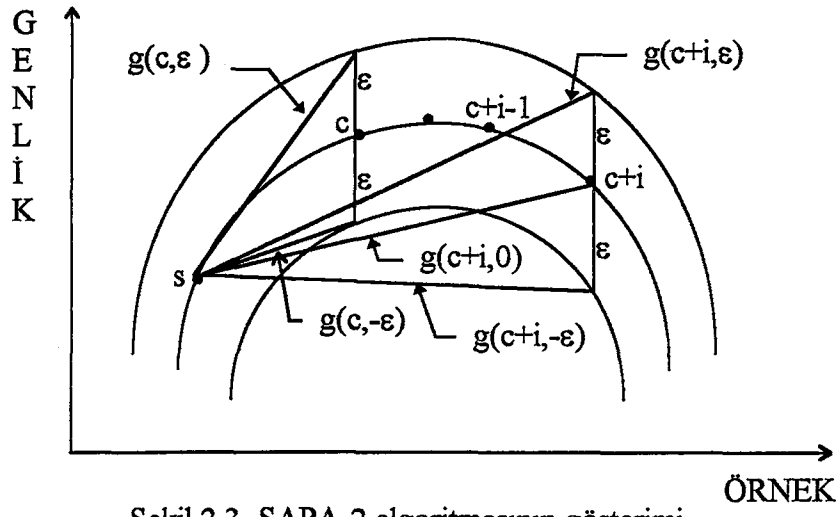
k=s ve k=c+i noktaları arasındaki noktaların uygun şekilde yaklaşıklıklarının yapılıp yapılmadığını anlamak için de,

$$g(c+i,0) \leq m_1 \qquad (2.14)$$

ve

$$g(c+i,0) \geq m_2 \qquad (2.15)$$

test hesaplamaları yapılmıştır (Şekil 2.3). Bu iki test şartlarından biri bozulduğunda, hemen bir önceki veri verteks olarak tutulur.



Şekil 2.3. SAPA-2 algoritmasının gösterimi

2.1.5. CORTES Tekniği

CORTES ("Coordinate Reduction Time System"), Dönüm Noktası ve AZTEC algoritmasının hibritidir. Böylece her iki yöntemin güçlü noktalarının avantajları sağlanır. CORTES yönteminde AZTEC, EKG işaretinde izoelektrik bölgeye ve TP, klinik olarak yüksek frekans bölgelerine (örneğin, QRS kompleksi, P ve T dalgaları) uygulanır. Deneysel olarak belirlenen T_{ln} çizgi uzunluk eşiğini kullanır. AZTEC platosu (AZTEC çizgisinin uzun peryodu) üretildiğinde CORTES, platonun T_{ln}'den daha uzun olması halinde AZTEC verilerini, eşit veya daha küçük olması halinde TP verilerini saklar. AZTEC platoları üretildiğinde eğim üretilmez. CORTES, AZTEC gibi aynı veri azalmasını sağlarken TP gibi aynı rekonstrükte hatasını üretir.

2.1.6. DPCM ile EKG Veri Sıkıştırma

Diferansiyel darbe kod modülasyonu (DPCM, differantial pulse code modulation), gerçek ilişkili işareti, ilişkisiz işarete çevirme esasına dayanan veri kodlama tekniğidir. Sıkıştırma, orjinal işaretten daha küçük varyansa sahip tahmini hata dizisini kodlamak suretiyle gerçekleştirilir. Kayıpsız ve kayıplı DPCM sıkıştırma

yöntemleri vardır. DPCM kodlayıcı estimatörü, polinom veya lineer prediktör [8] gibi tahmin algoritmaları olabilir.

2.1.7. Entropi Kodlama

Entropi kodlamayla veri sıkıştırma, değişken uzunluklu kod kelimelerini, sözkonusu frekanslara göre belirlenmiş kuantalama veri dizisine ayırma yoluyla elde edilir. Huffman kodlama, değişken uzunluklu kodları oluşturma yöntemidir. Bu kodlama, daha kısa (daha uzun) kod uzunluklarını, daha yüksek (daha düşük) olasılıkla meydana gelen değerlere ayırır [9]. Pratikte entropi kodlayıcılar, diğer kodlama teknikleri ile, örneğin genlik değerlerinin dağılımını esas alarak kodlayan DPCM tahmini hata dizisi ile kullanılır [10].

2.2. Transformasyon Teknikleri

Transformasyon teknikleri, lineer ortogonal transformasyon yöntemiyle giriş işaretinin önişlemesini, transform edilen çıkışın (açılım katsayıları) kodlamasını ve orjinal işareti uygun sunmayı gerektiren veri miktarını azaltmayı içerir. İşaretin yeniden yapılanmış hali ters transformasyon işlemiyle gerçekleştirilir ve orjinal işaret belli bir hata ile oluşturulur. Ancak oran (karar kriteri), seri açılım (transform) teknikleri kullanarak transformasyon katsayılarıyla veri zincirlerinin uygun şekilde ifadesidir.

Dijital işaret ifadelerinde Karhunen-Loeve Transformu(KLT), Fourier(FT), Kosinüs (CT), Walsh(WT), Haar(HT) gibi çok sayıda ayrık ortogonal transformlar [2], [11]-[13] geliştirildi. En optimum transform, verilen bir ortalama karesel hata için giriş işaretini ifade etmesi gereken en az sayıda ortonormal fonksiyonlu KLT'dir. Üstelik KLT ilişkisiz transform katsayılarını (diagonal kovaryans matrisi) netice verir ve diğer transformlarla karşılaştırılan toplam entropiyi minimize eder. Ancak, KLT temel vektörlerini (fonksiyonlarını) hesaplamak için gereken hesaplama zamanı çok fazladır. Bu gerçekten dolayıdır ki KLT temel vektörleri, büyük simetrik matris

olabilen orjinal işaretin kovaryans matrisinin özdeğerlerini ve buna bağlı özvektörlerini incelemeye dayalıdır. KLT'nin işleme ihtiyacının uzunluğu, hızlı algoritmalı (yani FT,WT, CT, HT, vs.) suboptimum transformlarının kullanımına önderlik etti. KLT gibi olmayan bu suboptimum transformalarının temel vektörleri girişten bağımsızdır (ön incelemeli). Örneğin, FT'deki temel vektörler basit sinüs ve kosinüstür. Buna karşılık WT temel vektörleri, farklı dizilerin kare dalgalarıdır. Sonuçta, performans açısından hiçbirisi KLT'yi geçemez [14].

Transformasyon yöntemlerinde (parametre çıkartma yöntemleri) parametre seti işaretten çıkartılır ve sıkıştırma için bir "ön bilgi" olarak düşünülür. Bundan dolayıdır ki gerçek zamanda EKG veri sıkıştırma ve gönderimi bu yöntemlerde uygun değildir[5].

BÖLÜM 3

YAPAY SİNİR AĞLARI (YSA)

3.1. YSA'nın Tanımı

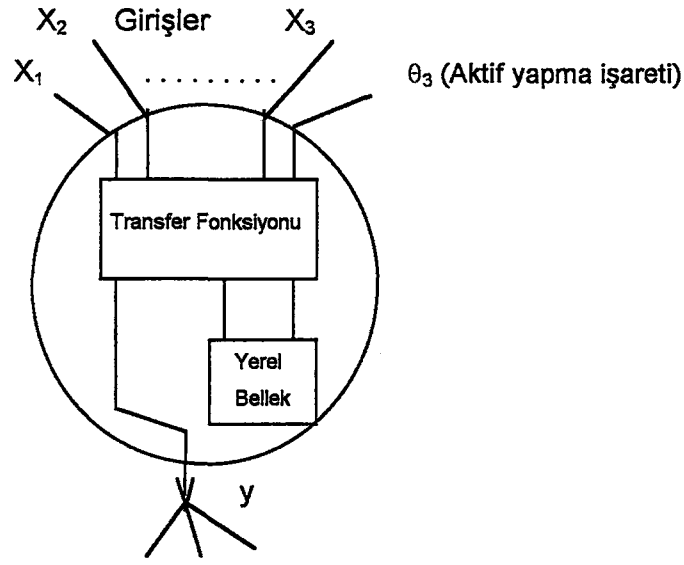
YSA paralel dağılmış bir bilgi işleme sistemidir. Bu sistem tek yönlü işaret kanalları (bağlantılar) ile birbirine bağlanan işlem elemanlarından oluşur. Çıkış işareti bir tane olup isteğe göre çoğaltılabilir. YSA yaklaşımının temel düşüncesiyle, insan beyninin fonksiyonları arasında benzerlik vardır. Bu yüzden YSA sistemine insan beyninin modeli denilebilir. YSA çevre şartlarına göre davranışlarını şekilleyebilir. Girişler ve istenen çıkışların sisteme verilmesi ile kendisini farklı cevaplar verebilecek şekilde ayarlayabilir. Ancak son derece karmaşık bir içyapısı vardır. onun için bugüne kadar gerçekleştirilen YSA; biyolojik fonksiyonların temel nöronlarını örnek alarak yerine getiren kompoze elemanlardır [15].

3.2. YSA'nın Yapısı ve İşlem Elemanı

YSA temel olarak, basit yapıda ve yönlü bir graf biçimindedir. Her bir düğüm hücre denilen n. dereceden lineer olmayan bir devredir. Düğümler işlem elemanı olarak tanımlanır. Düğümler arasında bağlantılar vardır. Her bağlantı tek yönlü işaret iletim yolu (gecikmesiz) olarak görev yapar. Her işlem elemanı istenildiği sayıda giriş bağlantısı ve tek bir çıkış bağlantısı alabilir. Fakat bu bağlantı kopya edilebilir. Yani bu tek çıkış birçok hücreyi besleyebilir. Ağ'daki tek gecikme, çıkışları ileten bağlantı yollarındaki iletim gecikmeleridir. İşlem elemanının çıkışı istenilen matematiksel tipte olabilir. Kısmen sürekli çalışma konumunda "aktif" halde eleman bir çıkış işareti üretir. Giriş işaretleri YSA'na bilgi taşır. Sonuç ise çıkış işaretlerinden alınabilir. Şekil3.1. 'de genel bir işlem elemanı (nöron, düğüm) gösterilmiştir.

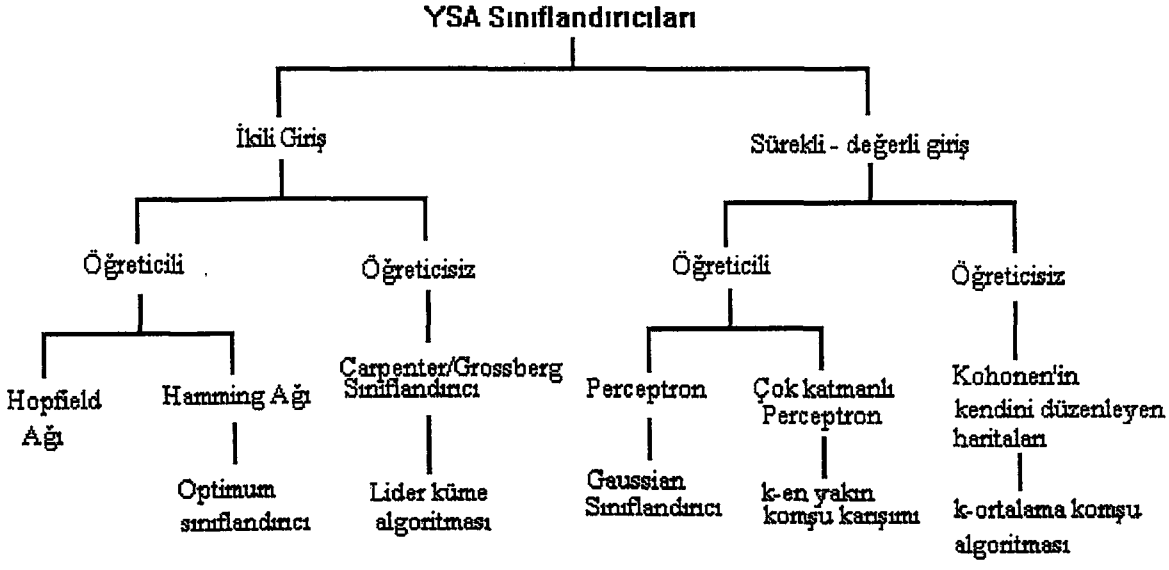
Her işlem elemanı kendisine verilen yerel veriye göre, kendisini ayarlayacak bütün YSA'nın enformasyon bölgesinin öğrenmesini sağlar. (Enformasyon bölgesi olasılık-yoğunluk fonksiyonu ile de tanımlanabilir). Enformasyon bölgesi birçok uygulamada, gerçek değerin "0" ile "1" arasında normalize edilmesi gerekir.

(Normalize etmek:gerçek değeri 85 olan bir girişi 0.85 şeklinde ağa uygulamaktır.) Normalizasyon aynı anda bütün girişlere uygulanabilir.



Şekil 3.1. Genel işlem elemanı yapısı

YSA birtakım alt kümelere ayrılabilir. Bu alt kümelerdeki elemanların transfer fonksiyonları aynıdır. Bu küçük gruplara katman ("layer") adı verilir. (örn: "multilayer perceptron, MLP") Ağ katmanların birbirlerine hiyerarşik bir şekilde bağlanmasından oluşmuştur. Dış dünyadan alınan bilgi, giriş katmanı ile taşınır. Giriş katmanlarının bir transfer fonksiyonları yoktur. YSA transfer fonksiyonu ve yerel bellek elemanı bir



Şekil 3.2. YSA 'nın sınıflandırıcıları

öğrenme kuralı ile giriş çıkış işareti arasındaki bağıntıya göre ayarlanır. Aktif yapma girişi için bir zamanlama fonksiyonu tanımlaması gerekebilir.

Bir işlem elemanına gelen girişler matematiksel tiplerine göre etiketlendirilerek sınıflandırılır. YSA, giriş veri tiplerine göre ikili giriş (0,1) ve sürekli değerli giriş olmak üzere Şekil 3.2.'deki gibi sınıflandırılır.

Bu tezde giriş işareti, sürekli-değerli (reel sayı) olduğundan dolayı, öğreticili öğrenmeye sahip olan çok katmanlı idrak ("perceptron", almaç) kullanılmıştır.

3.3. Bağlantı Geometrileri

Bağlantılarda taşınan işaret verisinin cinsi tanımlanmalıdır. Bağlantı geometrisi YSA için çok önemlidir ve bağlantı işareti her cinsten olabilir. Bağlantının nerede başlayıp nerede bittiğinin bilinmesi gerekir. 1'den N'e kadar olan bir işlem elemanı kümesinin bağlantıları aşağıda tanımlandığı gibi NxN boyutlu matris biçiminde gösterilebilir.

$w_{ij} = w_{ji} = 1 \Leftarrow i.$ işlem elemanı j. işlem elemanına bağlı,
$w_{ij} = w_{ji} = 0 \Leftarrow$ *bağlı değil*

En fazla N^2 bağlantı olur. Bağlantılar çeşitli geometrik bölgeler arasında demetler halinde düşünülebilir. Bu bağlantı demetlerinin uyması gereken kurallar şunlardır:

1- Bağlantı demetini oluşturan işlem elemanları aynı bölgeden çıkmalıdır.
2- Bağlantı demetinin işaretleri aynı matematiksel tipten olmalıdır.
3- Bağlantı demetinin işaretleri aynı sınıftan olmalıdır.
4- Bağlantı demetinin bir seçim fonksiyonu (σ) olmalıdır.

$\sigma : T \rightarrow 2^S$ T:Hedef belgesi S:kaynak bölgesi

Hedef bölgesindeki her işlem elemanı kaynak bölgesindeki her elemana giderse, tam ("full") bağlıdır. (örn:çok katmanlı almaç). Eğer her hedef bölgesi elemanı N kaynak bölgesi elemanına bağlı ise düzgün ("uniform") dağılımlıdır denir. Ayrıca her bir elemana, yine bir kaynak elemanı bağlı ise buna da, bire-bir bağlı, denir.

3.4. Eşik Fonksiyonları

Transfer veya işaret fonksiyonları olarak da adlandırılan eşik fonksiyonları, muhtemel sonsuz domen girişli işlem elemanlarını önceden belirlenmiş sınırda çıkış olarak düzenler. Üç tane yaygın eşik fonksiyonu vardır. Bunlar, rampa, basamak ve sigmoid fonksiyonudur. Bu çalışmada eşik fonksiyonu olarak kullanılan S biçimindeki sigmoid fonksiyonu; seviyeli, lineer olmayan çıkış veren, sınırlı, monoton artan fonksiyondur.

3.5. Ağırlık Uzayı

Bir çok YSA öğrenme işlemi, işlem elemanlarının ağırlığı değiştirilerek sağlanır. Böylece tanımlanan ağırlık değiştirilerek öğrenmede iyi bir model kullanıp, ağırlıkların bu modele göre değiştirilmesi esastır. Basit bir matematiksel model olarak herbir işlem elemanının "n" adet gerçek ağırlığı olduğu düşünülerek ve N adet işlem elemanı gözönüne alınırsa;

$$w = \left(w_1, w_{12}, \ldots, w_{1n}, w_{21}, w_{22}, \ldots, w_{2n}, \ldots\ldots, w_{N1}, w_{N2}, \ldots, w_{Nn}\right)^T$$

$$w = \left(w_1{}^T, w_2{}^T, w_3{}^T, \ldots\ldots, w_N{}^T\right)$$

w_1, w_2,..., w_N: işlem elemanlarının ağırlık vektörleridir.

$$w_1 = \begin{bmatrix} w_{11} \\ w_{12} \\ \cdot \\ \cdot \\ w_{1n} \end{bmatrix} \qquad \ldots\ldots\ldots\ldots\ldots w_N = \begin{bmatrix} w_{N1} \\ w_{N2} \\ \cdot \\ \cdot \\ w_{Nn} \end{bmatrix}$$

YSA ağırlık vektörü N, n boyutlu oklid uzayında yayılır. YSA'nın enformasyon işleme performansı, ağın ağırlık vektörünün belirli bir değeri ile bulunacaktır.

Hata değişimini inceleyen iki çeşit kural vardır;

1- Hata düzeltme kuralları,

2- Gradyen kuralları.

Hata düzeltme kuralları; Her bir giriş örüntüsünde ağırlıkları yeniden ağırlayarak çıktı hatasını en aza indirmeye çalışırlar. Gradyen kurallarında ise, ağırlıklar yeniden ayarlanarak ortalama karesel hatayı (MSE) en aza indirilmeye çalışılır.

Ağırlık vektörü ile çalışan YSA'da, önemli noktalardan birisi, bir öğrenme kuralı geliştirip, enformasyon bölgesi kullanarak (eşik fonksiyonu ile) ağırlık vektörü

"w"yı, istenilen YSA performansını verecek noktaya yöneltmektir. Genellikle öğrenme kuralı için bir performans yada maliyet fonksiyonu tanımlanır. Minimizasyon veya maksimizasyon ile "w" vektörü bulunur. Bir performasyon çeşidi olarak bilinen, MSE (karesel ortalama hata) şu şekilde tanımlanır.

$$F(w) = \int_A |f(x) - G(x,w)|^2 \rho(x) dv(x)$$

Burada amaç F'i küçültmeye çalışmaktır.
y=G(w,x):sistemin giriş çıkış fonksiyonu.
y:çıkış işareti vektörü
x:giriş işareti vektörü
w:ağırlık vektörü
ρ(x):olasılık yoğunluk fonksiyonu

3.6. YSA'da Eğitme ("Training")

Eğitme algoritmaları YSA'nın ayrılmaz bir parçasıdır. Eğitme algoritması eldeki problemin özelliğine göre öğrenme kuralını YSA'na nasıl adapte edeceğimizi belirtir. Üç çeşit eğitme algoritması yaygın olarak kullanılmaktadır.

1- Öğreticili eğitme ("supervised traning").
2- Skor ile eğitme ("graded training").
3- Kendini düzenleme ile eğitme ("self-organization training")

Öğreticili eğitmede, elimizde doğru örnekler vardır. Yani $(x_1,x_2,.....x_n)$ şeklindeki giriş vektörünün, $(y_1,y_2,...,y_n)$ şeklindeki çıkış vektörü, tam ve doğru olarak bilinmektedir. Herbir (x_1,y_1), $(x_2,y_2),...,(x_n,y_n)$ çifti için ağ doğru sonuçları verecek şekilde seçilen bir öğrenme kuralıyla beraber eğitilir.

Skor ile eğitmede, giriş işaretlerine karşılık gelen çıkış işaretleri tam olarak bilinmemektedir. Çıkış işareti yerine skor verilir ve ağın değerlendirilmesi yapılır. özellikle kontrol uygulamaları için idealdir. Çeşitli maliyet (cost) fonksiyonları kullanılır.

Kendini düzenleyen ağ, giriş işaretine göre kendini düzenleyerek organize eder. Olasılık yoğunluk fonksiyonlarına, sınıflandırma ve şekil tanıma problemlerine uygulanabilir.

Ne tür eğitme yöntemi kullanılırsa kullanılsın, herhangi bir ağ için gerekli karekteristik özellik, ağırlıkların verilen eğitme örneğine nasıl ayarlanacağının belirtilerek öğrenme kuralının oluşturulmasıdır. Öğrenme kuralının oluşturulması için bir örneğin ağa defalarca tanıtılması gerekebilir. Öğrenme kuralı ile ilişkili parametreler, ağın zaman içinde gelişme kaydetmesiyle değişebilir. Hangi YSA algoritmasında ne tür bir eğitme kullanıldığı bu bölümün giriş işaretlerinin sınıflandırılması kısmında gösterilmiştir.

3.7. Bellek

YSA'nın önemli bir özelliği bilgiyi saklama şeklidir. YSA'da bellek dağıtılır. Bağlantı ağırlıkları YSA bellek biçimleridir. Ağırlıkların değerleri ağın o anki bilgi durumunu temsil eder. Mesela; bir (giriş)/(istenen çıkış) çiftinin belirtilen bilgi parçası, ağın içinde birçok bellek biçimine dağıtılmıştır. Bellek üniteleri ile diğer saklı bilgiler, bu bilgiyi paylaşırlar. Bazı YSA bellekleri ilişkilidir. Öyleki eğitilen ağa birkısmı uygulanırsa, ağ bu girişe belleğindeki en yakın çıkışı bu giriş için seçer ve tam girişe bağlı çıkış ortaya çıkar. Eğer YSA oto-ilişkili ise, kısmi giriş vektörlerinin ağa verilmesi, bu girişlerin tamamlanması ile sonuçlanır. YSA belleğinin yapısı; eksik, gürültülü ve tam seçilemeyen bir giriş uygulandığı zaman bile mantıklı çıkış üretmeye uygundur. Bu kurala, genelleme adı verilir. Bir genellemenin kalitesi ve anlamı, uygulama çeşidine, ağın tipine ve karmaşıklığına dayanır. Lineer olmayan çok katmanlı ağlar (özellikle geriye yayınım ağları) gizli katmandaki özelliklerden öğrenirler ve bunları çıkışlar üretmek için birleştirirler. Gizli katmandaki bilgi, yeni giriş örüntülerine akılcı çözümler oluşturmak için kullanılabilir.

3.8. Hata Toleransı

Klasik hesaplama sistemleri çok az bir zarardan bile etkilenir. YSA için durum farklıdır. Bu farklılık YSA'nın hata toleranslı olmasıdır. İşlem elemanlarının az da olsa zarar görmesi sistemin bütününü etkiler. YSA paralel dağılmış parametreli bir sistem olduğundan her bir işlem elemanı izole edilmiş bir ada olarak düşünülebilir. Daha çok işlem elemanın zarar görmesi ile sistemin davranışı biraz daha değişir. Performans düşer ama sistem hiç bir zaman durma noktasına gelmez. YSA sistemlerinin hata toleranslı olmasının nedeni bilginin tek bir yerde saklanmayıp, sisteme dağıtılmasıdır. Bu özellik sistemin durmasının önemli bir zarara neden olacağı uygulamalarda önem kazanır.

3.9. Öğrenme Kuralları

Bilginin kurallar şeklinde açıklandığı klasik uzman sistemlerin tersine, YSA gösterilen örnekten öğrenerek kendi kurallarını oluşturur. Öğrenme; giriş örneklerine veya (tercihen) bu girişlerin çıkışlarına bağlı olarak ağın bağlantı ağırlıklarını değiştiren veya ayarlayan öğrenme kuralı ile gerçekleştirilir. Günümüzde kullanılan birçok öğrenme kuralı vardır. Bilinen en çok kullanılan öğrenme kuralları şunlardır.

- Raslantısal (Hebb) öğrenme kuralı
- Performans (Widrow ve ADALİNE) öğrenme kuralı
- Kompetif (Kohonen) öğrenme
- Filtreleme (Grossberg)
- Spotitemporal öğrenme
- Genelleştirilmiş Delta Kuralı Öğrenme

Burada bütün öğrenme kuralları incelenmeyecektir. Sadece tezde kullanılan "Genelleştirilmiş Delta Kuralı" öğrenmesi ilk oluşumundan yani perceptron halinden başlayıp, tüm gelişimiyle son durumu anlatılacaktır.

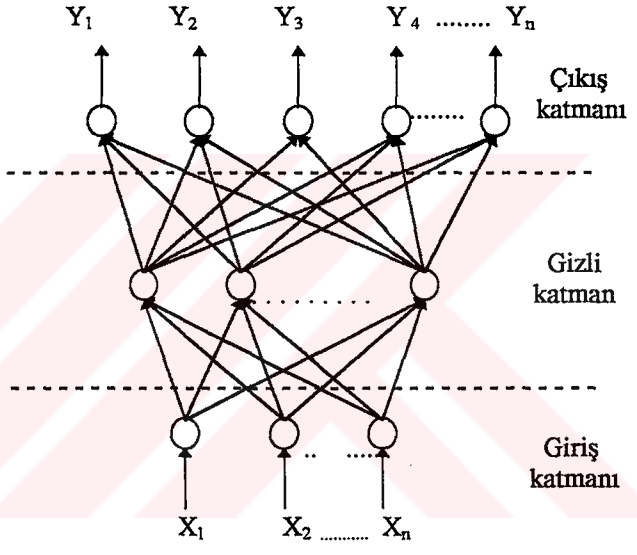
3.10. Perceptron (idrak,almaç)

Perceptron ağı, ilk 1943 yılında Mc Culloch ve Pitts tarafından saptandı. Bu ilk perceptron modeline göre, giriş bilgisinin mevcut iki sınıftan hangisine eşit olabileceğini bulacak şekilde eğitilen basit bir ağdır. Daha sonra 1960 yıllarında F.Rosenblatt yukarıdaki ağ tipini biraz daha geliştirdi. Ama Minsky ve Papert bu tek katmanlı perceptronun XOR (ayrıcalıklı veya) işlemini gerçekleştiremediğini ispatladılar. XOR gibi 3 veya daha fazla sınıfa ihtiyaç duyulan problemleri çözmek için yapılması gereken işlem; YSA yeni katmanlar eklemektir. Eşik bağlarıyla oluşturulan karar bölgesi şeklinin karmaşıklığı sadece eklenmiş olan katmanların sayısıyla sınırlıdır. Bilgi lineer yayılamıyorsa 2.katmanın çıkışı konveks bölgededir bunun neticesi olarak 3.katmandan gelecek olan çıkış bilgisinin şekli, herhangi bir bölgenin şeklinde olabilir. Bu sebeple ihtiyaç duyulan katman sayısı üç olmaktadır. Üç-katmanlı perceptronun 2. katmanında ihtiyaç olan düğümlerin sayısı, bir karar bölgesinin birleştirilmemiş hali veya bir ağ gözünün bir dış-bükey alandan meydana gelemediğinin birinden büyük olması lazımdır. 2. katmandaki düğüm sayısı en kötü durumda giriş bilgilerinin dağılımını yapan bölgenin bağlanmamış sayısına eşit olması gerekir. Birinci katmandakilerin sayısı her iki katmandaki değişim ile 3 yada 4 köşeli dışbükey bir alan oluşturmaya yeterli seviyede olmalıdır. Bunun tipik bir sonucu olarakda en az 1. katmandakinden üç kat fazla miktarda olması gerekmektedir. Bununla beraber Gutierrez ve arkadaşları değişik perceptron ağlarının ihtiyacı olan düğüm sayıları

hakkında çalışma yaptılar ve çok fazla düğümünde, çok az sayıda olduğu gibi zararlı etkisi olduğunu buldular. Tek katmanlı perceptron uygulanan her eğitimin seti modelinin en önemli özelliği, lineer biçimde dağılmak zorunda olmasıdır.

3.11. Çok Katmanlı İdrak (Multi-Layer Perceptron)

Çok katmanlı perceptron giriş ve çıkış katmanları arasında birden fazla katmanın kullanıldığı YSA sistemleridir. Gizli katman (hidden layer) olarak isimlendirilen bu katmanlarda, düğümleri aracısız giriş olmayan ve aracısız çıkış veremeyen üniteler vardır. Şekil 3.3'de çok katmanlı perceptronun genel yapısı verilmiştir.



Şekil 3.3. Çok katmanlı perceptron yapısı

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

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

3.12. Hatanın Geriye Yayılması Algoritması ve Genelleştirilmiş Delta Kuralı

Hatanın geriye yayılması algoritması, karesi alınmış hata fonksiyonunu minimize eden kodlu bir algoritma olup ve genelleştirilmiş delta kuralını eğitme için kullanılır. Algoritmaya göre her bir j biriminin çıkışı o_j şu şekilde tanımlanır;

$$o_j = f(net_j) = f(x) \quad ise \quad net_j = \sum_i^j w_{ji} o_i + \theta_j \qquad (3.1)$$

Burada o_i; i. biriminin çıkışı w_{ji}; j biriminden i birimine bağlantının ağırlığı, θ_j; j biriminin kutbu (bias), Σ_i; çıkışı j birimine akan her i biriminin toplamıdır. f(x) bir monoton artan ve türevi alınabilen fonksiyondur. Pratikte bir lojistik aktivasyon fonksiyonu olarak $f(x)=1/ 1 + e^{-x}$ (sigmoid) daha çok kullanılır.

m-boyutlu giriş örüntüleri set edildiğinde { $i_p = (i_{p1}, i_{p2} \ldots i_{pn})$; $p \in P$}'dir. Benzer şekilde istenilen n- boyutlu çıkış örüntüleri { $t_p = (t_{p1}, t_{p2} \ldots t_{pn})$ $p \in P$} belirtir. Burada; P:YSA uygulanan işaret, şekil vb gibi örüntülerini verir.

Bir görüntü için karesel hata (MSE) fonksiyonu E_p şu şekilde tanımlanır;

$$E_p = \frac{1}{2} \sum_{j \in \text{çıkışkatmanı}} (t_{pj} - o_{pj})^2 \qquad (3.2)$$

Amaç uygun w_{ji} ve θ_j seçimiyle toplam hatayı yeterince küçük yapmaktır. Bu amacı gerçekleştirmek için, giriş örüntüsü ard arda ve rasgele biçimde seçilir. Daha sonra w_{ji} ve θ_j şöyle değiştirilir;

i_{pi} : Giriş işaretinin i bileşeni;

t_{pj} : Çıkış vektörünün j bileşeni;

o_{pj} : YSA uygulanan P örüntü setinin ürettiği çıkış ise;

$$\delta_{pj} = (t_{pj} - o_{pj}) \tag{3.3}$$

$$\Delta_p w_{ji} = -\varepsilon\left(\frac{\partial E_p}{\partial w_{ji}}\right) \tag{3.4}$$

$$\Delta_p \theta_j = -\varepsilon\left(\frac{\partial E_p}{\partial \theta_j}\right) \tag{3.5}$$

Burada ε : öğrenme oranı adı verilen küçük bir pozitif sabit sayılır. Şayet gizli katman yok ise; (3.4) ve (3.5)'in sağ tarafi hesaplanır, o zaman;

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}} \tag{3.6}$$

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) = -\delta_{pj} \tag{3.7}$$

$$o_p = \sum_i w_{ji} . i_{pi} \qquad ise; \tag{3.8}$$

$$\frac{\partial o_{pj}}{\partial w_{ji}} = i_{pi} \tag{3.9}$$

elde edilir. (3.7) ve (3.9) ifadelerini (3.6)'da yerine koyarsak;

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} i_{pi} \tag{3.10}$$

olur. Öğrenmede en küçük minimuma ulaşılmak istenir. Bu durumda j. düğümün lineer olmayan çıkışı;

$$o_{pj} = f_j(netp_j) \quad \Rightarrow \quad netp_j = \sum_i w_{ji} o_{pj} \tag{3.11}$$

şeklindedir. Bu durumda;

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial netp_j} \frac{\partial netp_j}{\partial w_{ji}} \quad \Leftrightarrow \quad \frac{\partial netp_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k w_{jk} . o_{pk} = o_{pi} \tag{3.12}$$

$$\delta_{pi} = -\frac{\partial E_p}{\partial netp_j} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial netp_j} = -\frac{\partial E_p}{\partial o_{pj}} f_j'(netp_j) \tag{3.13}$$

Burada iki durum vardır:

1- o_{pj} YSA'nın çıkışı ise; (3.7) ifadesini (3.13)'de yerine koyarsak,

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(netp_j) \tag{3.14}$$

bulunur.

2- Eğer gizli katmanların çıkış işaretinden bahsediliyorsa;

$$\sum_k \frac{\partial E_p}{\partial \text{netp}_k} \frac{\partial \text{netp}_k}{\partial p_j} \tag{3.15}$$

şeklinde ise,

$$\sum_k \frac{\partial E_p}{\partial netp_k} \frac{\partial}{\partial o_{pj}} \sum_i w_{ki}\, o_{pj} = -\sum_k \delta_{pk} w_{kj} \tag{3.16}$$

olur. Bulduğumuz son işlemi (3.13)'de yerine koyarsak;

$$\delta_{pj} = f'(net_{pj}) \sum_k \delta_{pk} w_{kj} \tag{3.17}$$

elde edilir. (3.16) denklemindeki (-) işareti, ağırlıkların ters yönde değiştiğini belirtir. Bütün yaptığımız işlemleri kısaca özetleyecek olursak;

1. Genelleştirilmiş Δ (delta) kuralı:

$$\Delta_p w_{ji} = \varepsilon \delta_{pj} i_{pi}$$

2. Çıkış katmanı elemanları için;

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(\text{netp}_j)$$

3. Gizli katman elemanları için;

$$\delta_{pj} = f'_j(\text{netp}_j) \sum_k \delta_{pk} w_{kj}$$

olur. İşlem elemanında, transfer (eşik) fonksiyonu olarak "sigmoid" fonksiyonu kullanılarak türevi alınır ve gerekli kısaltmalar yapılırsa;

$$\frac{\partial o_{pj}}{\partial netp_j} = o_{pj}(1 - o_{pj}) \tag{3.18}$$

bulunur. Bunu (3.14) de yerine koyarsak, çıkış elemanı için;

$$\delta_{pj} = (t_{pj} - o_{pj}) o_{pj} (1 - o_{pj}) \tag{3.19}$$

elde edilir. (3.18)'u (3.17) de yerine koyarsak, gizli katman elemanı için;

$$\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_k \delta_{pk} w_{kj} \tag{3.20}$$

bulunur. Yukarıda toplam içerisinde gösterilen k'nın, j çıkış birimine akan her birim k olduğuna dikkat edilmelidir. Hesaplamayı hızlandırmak için momentum terimleri (α) eklenirse, en genel halde çıkış ve gizli katman ifadeleri şu şekilde olur:

$$\Delta_p w_{ji}(n+1) = \varepsilon \delta_{pj} o_{pi} + \alpha \Delta_p w_{ji}(n) \tag{3.21}$$

$$\Delta_p \theta_j(n+1) = \varepsilon \delta_{pj} + \alpha \Delta_p \theta_j(n) \tag{3.22}$$

Burada; n: öğrenme saykıllarının sayısını gösterir. (α) küçük pozitif bir sayıdır.

3.13. Öğrenme ve Momentum Katsayıları

YSA ile ilgili bir başka sorunda, düzgün bir öğrenme katsayısının (ε) ayarlanmasıdır. Ağırlıkları çok yüksek tutmak davranışın bozulmasına neden olabilir. O nedenle öğrenme katsayısını böyle bir davranışı önlemek için küçük tutmak gereklidir. Öğrenme katsayısı, $0.01 < \varepsilon < 10$ aralığında seçilen sabit bir sayıdır. Öte yandan çok küçük bir öğrenme oranıda, öğrenme işleminin yavaşlamasına yol açar. Momentum (α) fikri bu noktadan hareketle ortaya atılmıştır. Momentum mevcut delta ağırlığı üzerinden önceki delta ağırlığının belli bir kısmını besler. Böylece daha düşük öğrenme katsayısı ile daha hızlı öğrenme elde edilir. Momentum katsayısı genellikle $0 < \alpha < 1$ aralığında değişen sabit bir sayıdır.

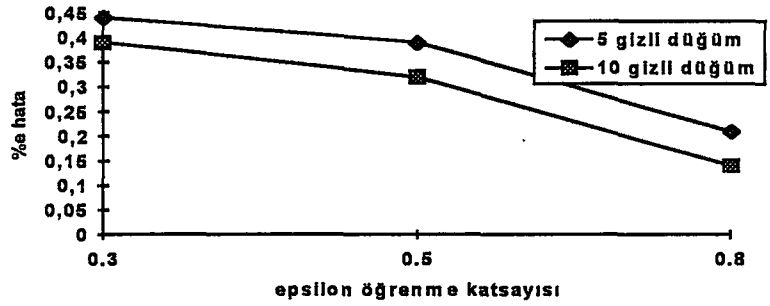
BÖLÜM 4

BULGULAR

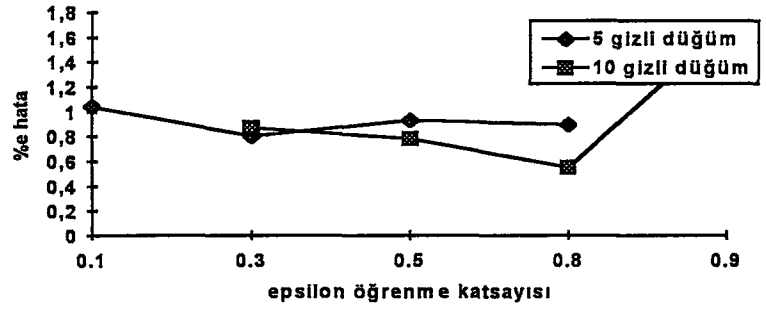
Bu tezde, yapay sinir ağları (YSA) kullanılarak çeşitli elektrokardiyogramların (EKG) sıkıştırma ve açma işlemi yapıldı. Eldeki EKG'ler II nolu derivasyona ait MIT/BIH 100,107,109 numaralı kayıtlarıdır. 100 nolu kayıt normal bir EKG'ye, 107 nolu kayıt kalp-pili ("pacemaker") takılı hastadan alınan EKG'ye, 109 nolu kayıt kalbin elektrik aktivitesinde sol dal bloğu olan hastadan alınan EKG'ye aittir. Orjinal veriler 360 Hz'de örneklenmiş olup, bu çalışmada bunu herbir peryodu 200 veri olacak şekilde ayarlayan Borland C programlama dilinde bir program yapıldı.

Pentium-75 mikroişlemcili bilgisayarda Borland C programlama diliyle yapılan programda (yazılımda) sıkıştırma işlemi Bölüm 3'te anlatıldığı gibi 200 giriş, 10 ve 5'er gizli düğümle gerçekleştirildi. Bu değerler aynı zamanda sıkıştırma oranını da yani 20:1 ve 40:1 verir. Buna göre kullanılan YSA ağ mimarisi 200:10:200 ve 200:5:200 şeklindedir. Eğitme işlemlerinde genellikle 20 periyot kullanıldı. Bölüm 3'te de belirtildiği gibi öğrenme katsayısı (ε) ve momentum katsayısı(α) eğitme işleminde hatayı minimum yapacak şekilde seçilmelidir. Bunun için 500 iterasyonluk denemeler sonucu hatayı etkileyen ε ve α değerleri Şekil 4.1'de verilmiştir.

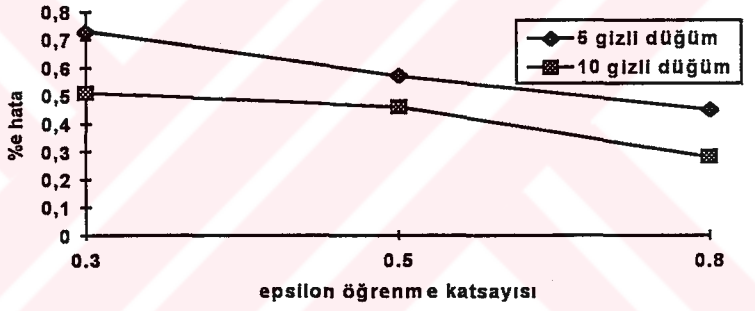
Sıkıştırılan EKG'leri açma işlemi 10 giriş (sıkıştırmadaki 10 gizli düğüm nedeniyle) veya 5 giriş (sıkıştırmadaki 5 gizli düğüm nedeniyle) 200 çıkış düğümü ile gerçekleştirildi. Açma işleminde gizli düğüm sayıları eğitme esnasında ε ve α ile birlikte en az hatayı verecek şekilde seçildi. Eğitme işlemleri sonucu elde edilen ağırlık parametreleriyle yeniden yapılandırılmış EKG grafikleri, orjinalleriyle birlikte karşılaştırmalı olarak Şekil 4.2.a,b,c,d' de gösterilmiştir.



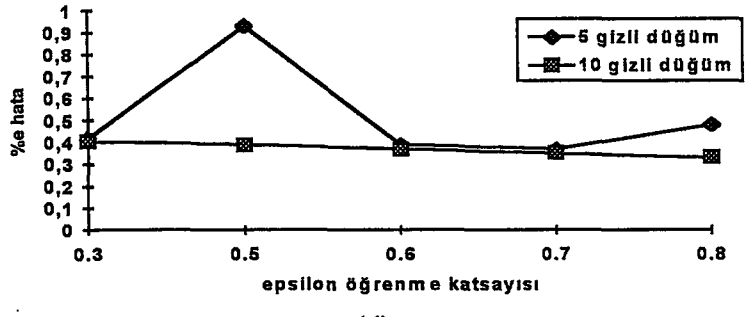
(a)



(b)

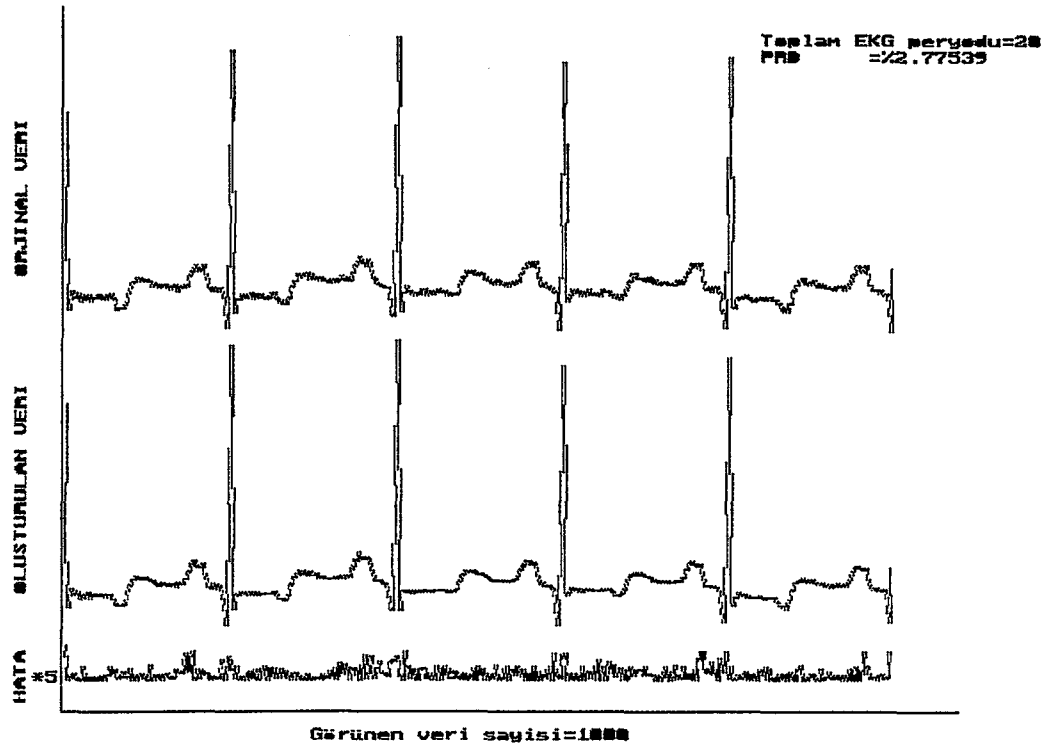


(c)

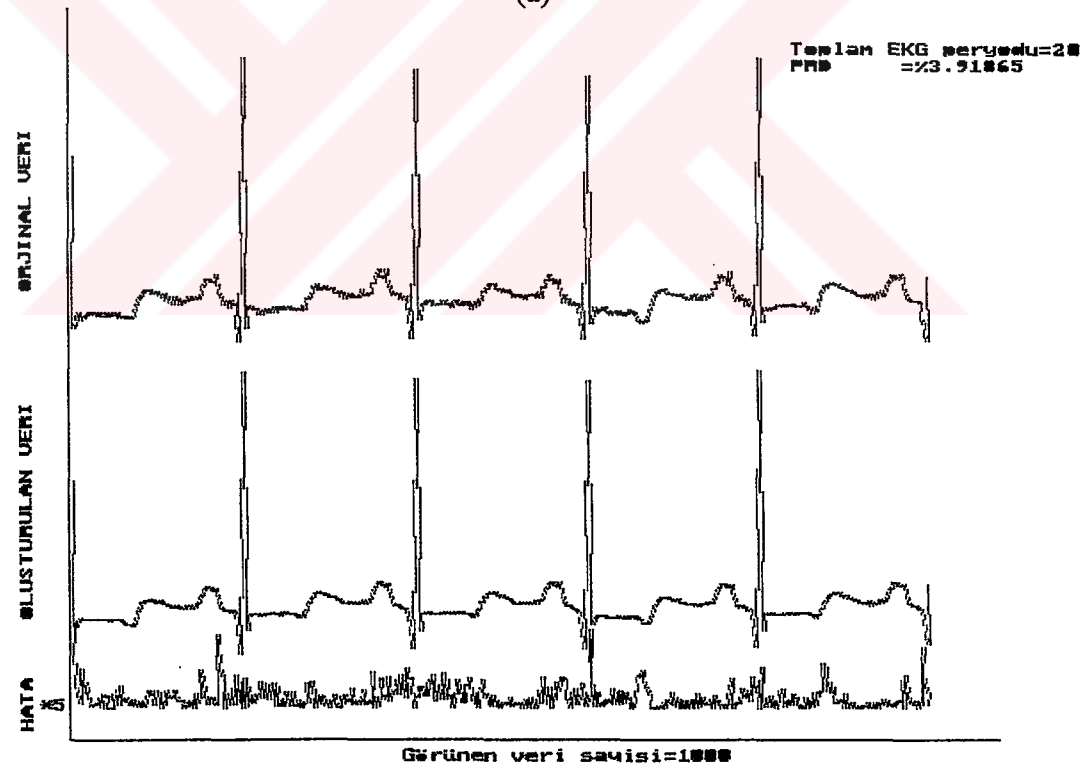


(d)

Şekil 4.1 Şıkıştırma işlemi için aşağıdaki EKG tiplerinde momentum katsayısı (α)=0.1 değerine karşılık ε öğrenme katsayısının hataya etkisi:
a) MIT/BIH 100 nolu EKG verisi (normal),
b) MIT/BIH 107 nolu EKG verisi (hastalıklı),
c) MIT/BIH 109 nolu EKG verisi(hastalıklı,
d) Üçünün karışımı ile elde edilen EKG verisi.

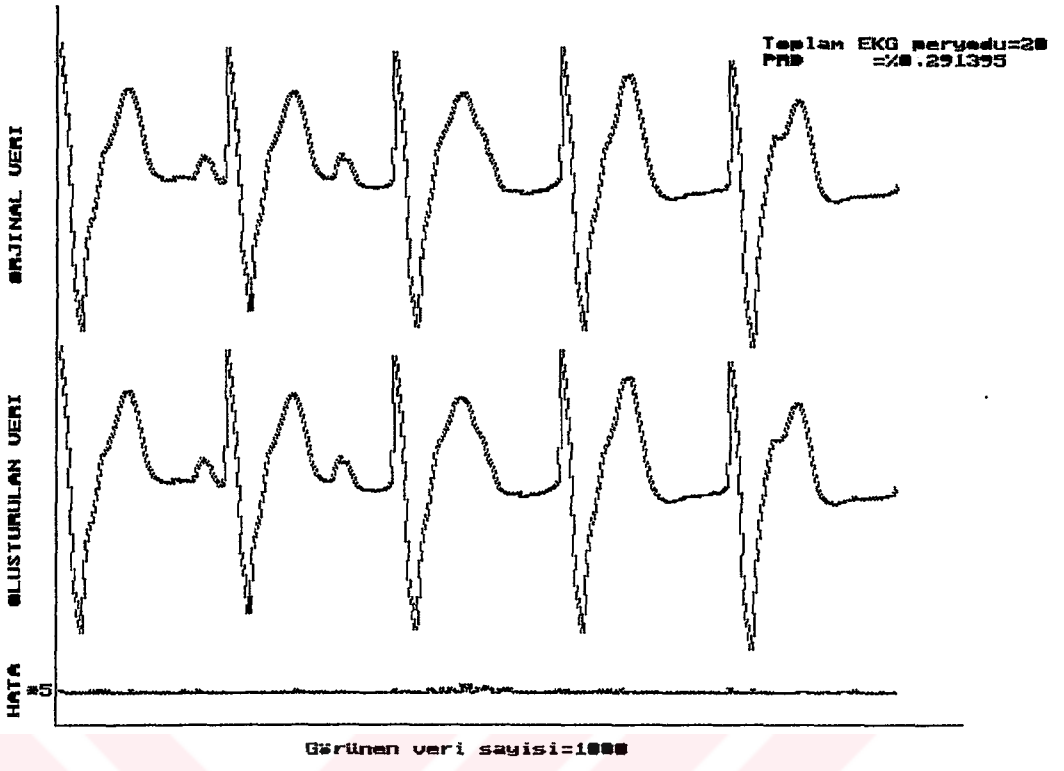


(a)

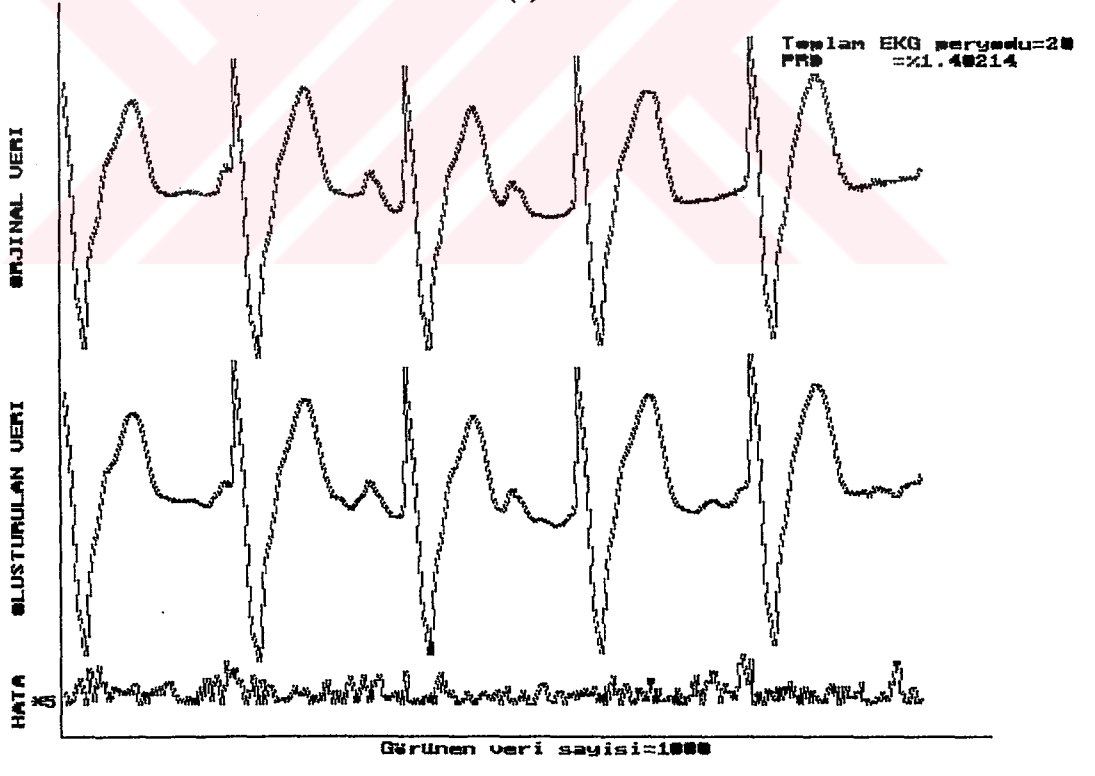


(b)

Şekil 4.2.a MIT/BIH 100 nolu EKG'nin orjinal ve (a) 10 gizli düğüm, (b) 5 gizli düğüm için yeniden yapılanmış grafikleri ve aralarındaki fark (5 kat) grafikleri.

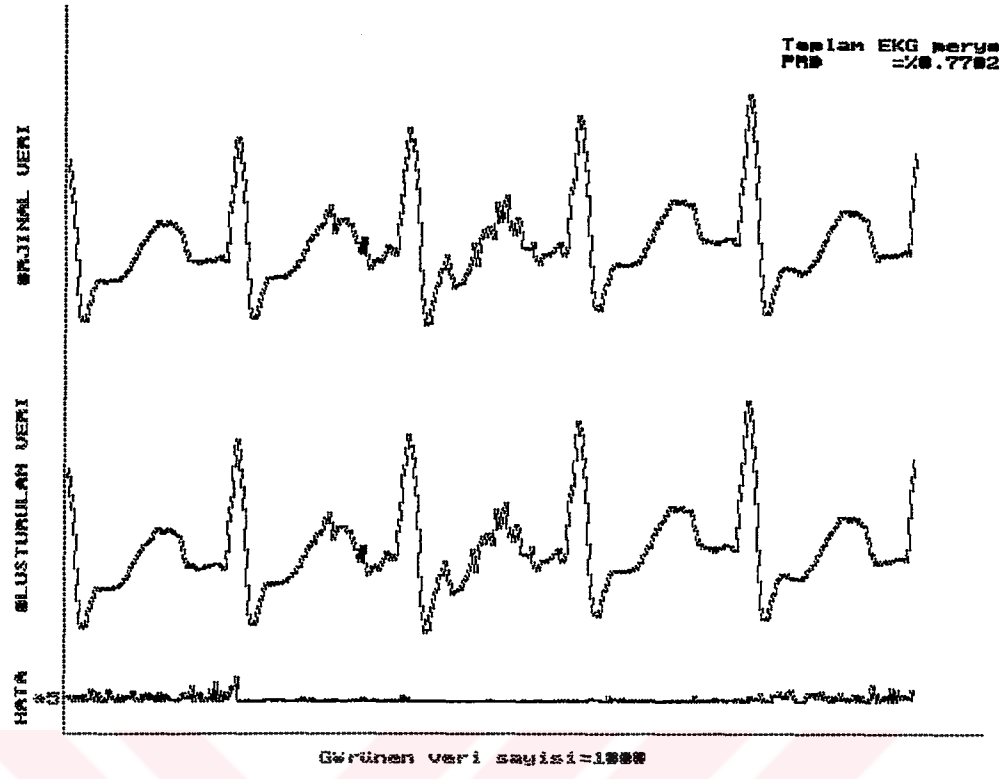


(a)

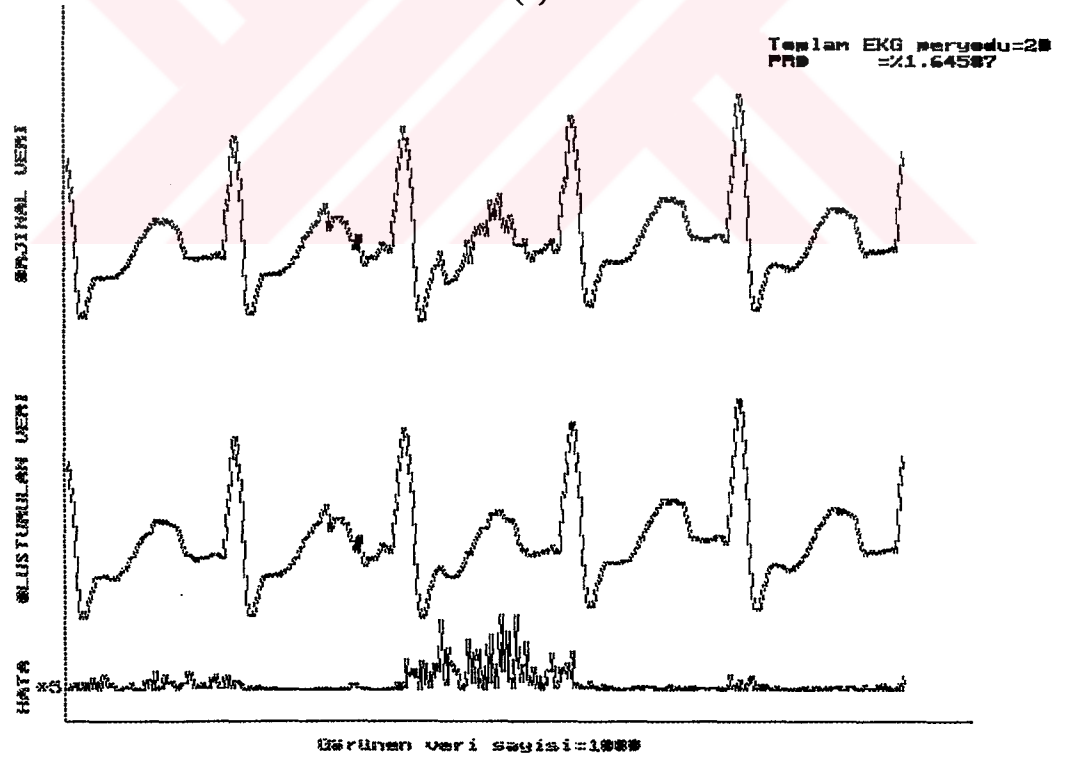


(b)

Şekil 4.2.b MIT/BIH 107 nolu EKG'nin orjinal ve (a) 10 gizli düğüm, (b) 5 gizli düğüm için yeniden yapılanmış grafikleri ve aralarındaki fark (5 kat) grafikleri.

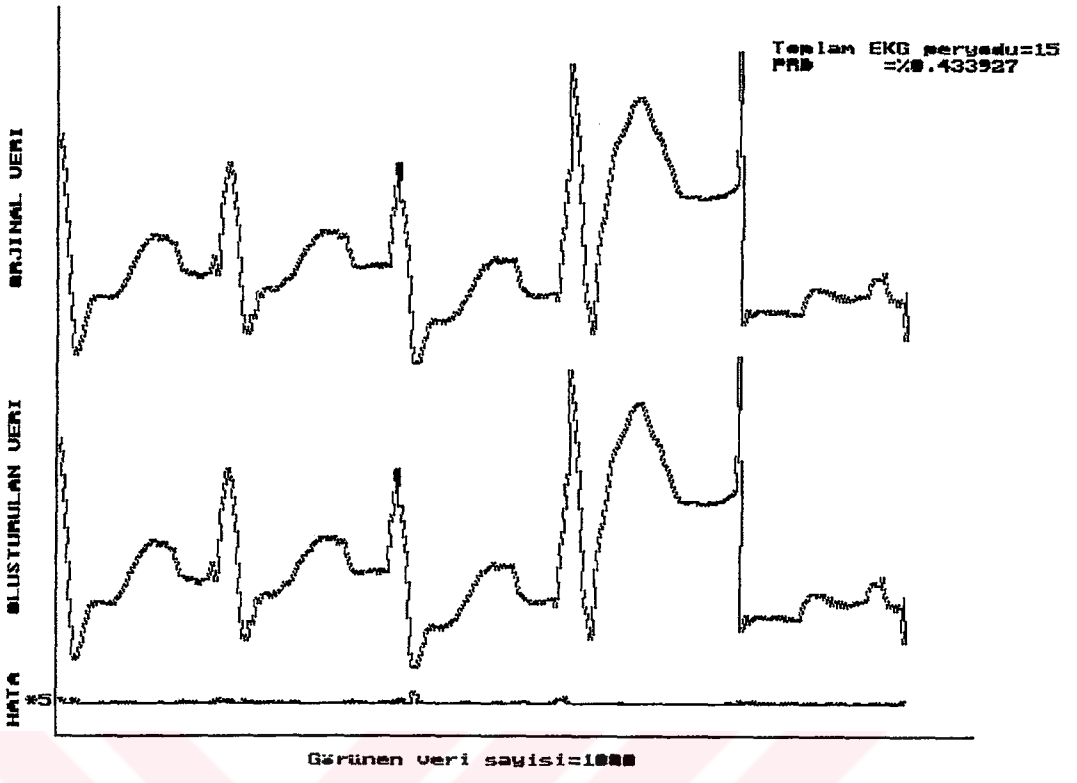


(a)

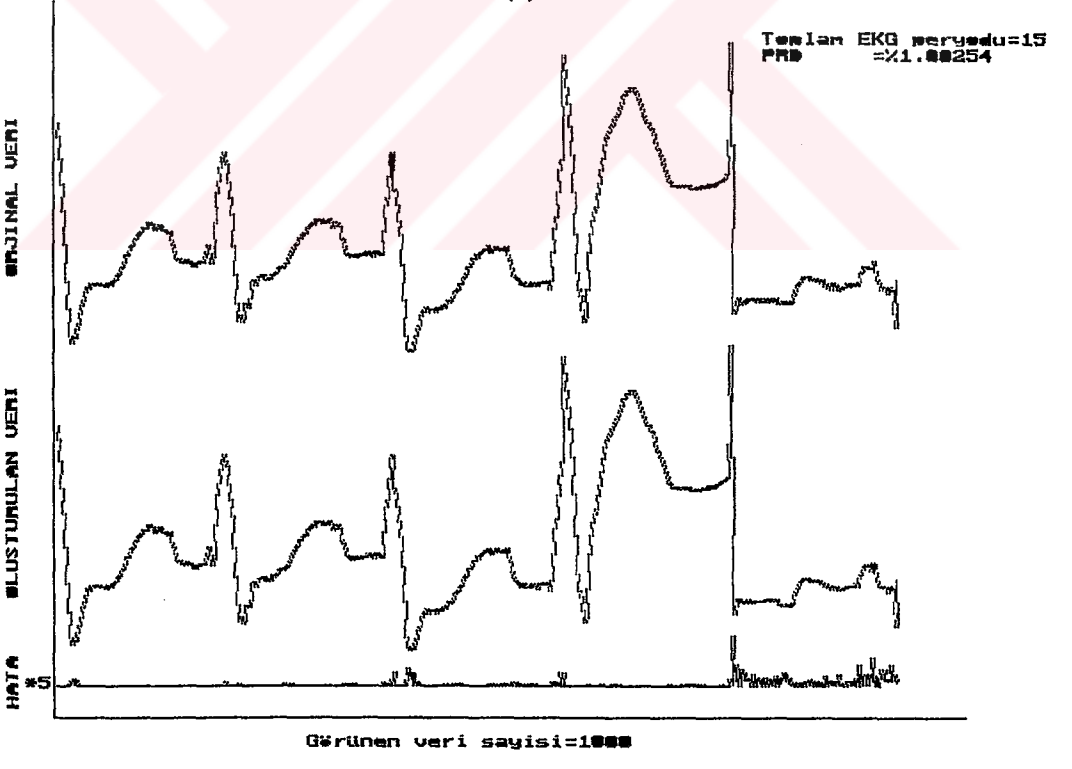


(b)

Şekil 4.2.c MIT/BIH 109 nolu EKG'nin orjinal ve (a) 10 gizli düğüm, (b) 5 gizli düğüm için yeniden yapılanmış grafikleri ve aralarındaki farkın (5 kat) grafikleri.



(a)



(b)

Şekil 4.2.d MIT/BIH 100,107,109 nolu EKG'lerin karışımına ait orjinal ve (a) 10 gizli düğüm, (b) 5 gizli düğüm için yeniden yapılanmış grafikleri ve aralarındaki fark (5 kat) grafikleri.

BÖLÜM 5

SONUÇLAR, ÖNERİLER VE VERİ SIKIŞTIRMA PROGRAMININ İŞLEYİŞİ

Tezde yeni sayılabilecek bir teknik ile EKG verilerinin sıkıştırılması yapıldı. Çalışma diğer tekniklerle karşılaştırıldığında gerçek zamanda çalışmadığı için genel olarak transformasyon teknikleri grubuna dahildir. Ayrıca eğitme işlemi sonucu parametre seti elde edilmektedir. Veriler, bu parametre setine göre sıkıştırılıp yeniden yapılandırılmaktadır. Bu tekniğin üstünlüğü, çok sayıda veri sıkıştırmada ortaya çıkmaktadır.

Bulgular bölümündeki grafiklerde EKG'lerin performans indeksleri (PRD) karşılaştırıldığında,5 gizli düğüm kullanılan yapılardaki PRD'lerin, 10 gizli düğüm kullanılan yapılardaki PRD'lerden daha fazla çıkmaktadır. PRD'lerin düşük olması Bölüm 2'de belirtildiği gibi algoritmanın başarısını göstermektedir. 20 periyotluk EKG'ler için elde edilen PRD'ler Tablo 5.1.'de verilmiştir. Aynı tip bir EKG için farklı verilerle sıkıştırılıp ve yeniden yapılandırıldığında hatalar artmaktadır ve dolayısıyla PRD'si büyümektedir.

Tablo 5.1. 20 periyotluk EKG'ler için elde edilen PRD'ler.

	Gizli D.	100 nolu kayıt	107 nolu kayıt	109 nolu kayıt	Karma
PRD	10 G.D.	2.77	0.29	0.77	0.43
(%)	5 G.D:	3.91	1.40	1.64	1.0

Diğer veri sıkıştırma tekniklerinin sıkıştırma oranlarını ve performans indeksleri, 60 vurum/dak.'lık EKG verisi ve 500 Hz ve 12 bit örnekleme şartları için Tablo 5.2.'de verilmiştir. Bu tekniklerin bazıları örnekleme oranına, kuantalama adım miktarına ve işaretteki gürültüye duyarlıdır. Tam karşılaştırma yapmak için genel bir

Tablo 5.2. Bazı sıkıştırma teknikleri için CR ve PRD'ler.

Teknik	CR	PRD (%)
TP	2.0	4.8
AZTEC	10.0	8.7
SAPA-1	9.8	8.2
SAPA-2	9.3	7.6

veri tabanından EKG'lere ait büyük bir set, tüm yöntemler tarafından işlenmesi gereklidir ve performansları genel bir "iyilik" ölçüsüyle değerlendirilmesi gereklidir.

Yapılan çalışmanın eksiklikleri mevcuttur. Öncelikle eğitmelerin çoğu, aynı tip EKG'ler için yapıldı.. Yani aynı hastalığa ait veya normal EKG'ler için sözkonusudur. Ayrıca program kapasitesi yüzünden eğitme için sınırlı sayıda EKG peryodu alınabilmektedir.

Algoritma basit bir algoritma olup daha kompleks bir çalışma, sözkonusu hataları da azaltacak şekilde yapılabilir. Nitekim Japonya'da böyle bir çalışma yapılmıştır.

Çalışma, Borland C programlama dilinde yapılan iki ayrı programla gerçekleşmektedir. Birinci program sıkışturma ve yeniden yapılanma için ağırlık parametre setlerini oluşturan eğitme programıdır. İkinci program ise elde edilen ağırlık parametre setlerine göre yeni EKG'ler için sıkıştırma ve yeniden yapılanmayı sağlayan programdır. Programlar EK 1 ve EK 2 'de verilmiştir.

Birinci programın akış diyagramı şu şekildedir;
Eğitme işlemi için giriş değerleri girilir: Giriş düğüm sayısı (200;5 veya 10), çıkış düğüm sayısı (200;200)[1], gizli düğüm sayısı (5 veya 10; hatayı minimum yapacak sayıda), giriş veri dosyası (averi.dat;aver.dat), EKG periyot sayısı (genellikle 20;20), momentum katsayısı ($0<\alpha<1$), öğrenme katsayısı ($0<\varepsilon<1$) ve iterasyon sayısı.

[1] Parantez içindeki ilk sayı sıkıştırma ve ikinci sayı yeniden yapılanma için geçerlidir.

İkinci programın akış diyagramı ise;

Sıkıştırma veya yeniden yapılanma (açma) işlemi seçilir. Sıkıştırma için şu değerler girilir: Sıkıştırılacak giriş veri dosyası (averi.dat), ağırlık parametreleri dosyası (averi), sıkıştırılacak örnek sayısı. Yeniden yapılanma için şu değerler girilir: Sıkıştırılmış giriş veri dosyası (sdeg.dat), ağırlık parametreleri dosyası (aver), yeniden yapılanacak örnek sayısı.

KAYNAKLAR:

[1] REDDY, B.R.S. and MURTHY, I.S.N., ECG Data Compression Using Fourier Descriptors, IEEE Trans. Biomed. Eng., Vol. BME-33, pp. 428-434, April 1986.

[2] AHMED, N., et al., Electrocardiographic Data Compression via Orthogonal Transforms, IEEE Trans. Biomed. Eng., Vol. BME-22, pp. 484-492, November 1975.

[3] KORÜREK, M.,Tıp Elektroniğinde Tasarım İlkeleri, Ders Notları, İ.T.Ü. Elektrik-Elektronik Fakültesi, 1988.

[4] COX, J.R., et al., AZTEC: A Preprocessing Program Real-Time ECG Rhythm Analysis, IEEE Trans. Biomed. Eng., Vol. BME-15, pp. 128-129, 1968.

[5] FURHT, B. and PEREZ, A., An Adaptive Real-Time ECG Compression Algorithm with Variable Treshold, IEEE Trans. Biomed. Eng., Vol.BME-35, 1988.

[6] MUELLER, W.C., Arrhythmia Detection Program for an Ambulatory ECG Monitor, Biomed. Sci. Instrument., Vol. 14, pp. 81-85, 1978.

[7] ISHIJIMA., M., et al., Scan Along Polygonal Approximation of Electrocardiograms, IEEE Trans. Biomed. Eng., Vol.BME-30, pp. 723-729, November 1983.

[8] RUTTIMAN, U.E. and PIPBERGER, H.V.,Compression of the ECG by Prediction or Interpolation and Entropy Encoding , IEEE Trans. Biomed. Eng. Vol. BME-26, pp. 613-623, November 1979.

[9] HUFFMAN, D.A., A Method for the Construction of Minimum-Redundancy Codes, Proc. IRE, Vol. 40, pp. 1098-1101, September 1952.

[10] AYDIN, M.C., Multilead ECG Data Compression By Multirate Signal Processing and Transform Domain Coding Techniques, Yüksek Lisans Tezi, Bilkent Üniversitesi, 1991.

[11] YOUNG, T.Y., and HUGGINS, W.H., On the Represantation of Electrocardiograms, IEEE Trans. Biomed. Electr., Vol. BME-9, pp. 214-221, Octeber 1962.

[12] SHANKARA, B.R., and MURTHY, I.S.N., ECG Data Compression Using Fourier Descriptors, IEEE Trans. Biomed. Eng., Vol. BME-33, pp. 428-434, April 1986.

[13] DYER,S.A., et al, Vectorcardiographic Data Compression via Walsh and Cosine Transforms, IEEE Trans. on Electromagnetic Compatibility, Vol. EMC-27, pp. 24-34, February 1985.

[14] JALALEDDINE, S.T.S., et al., ECG Data Compression Techniques- A Unified Approach, IEEE Trans. Biomed. Eng., Vol. BME-37, pp. 329-343, April 1990.

[15] KARLIK, B., Çok Fonksiyonlu Protezler İçin Yapay Sinir Ağları Kullanarak Miyoelektrik Kontrol , Doktora Tezi, Y.T.Ü., 1994.

EK 1

EĞİTME İŞLEMLERİNİ YAPAN PROGRAM

```
#include <stdio.h>
#include <graphics.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
#include <alloc.h>
#include<ctype.h>
#define NMXUNIT   10
#define NMXHLR   10
#define NMXOATTR  240
#define NMXINP  60
#define NMXIATTR  240
#define SEXIT     3
#define RESTRT   2
#define FEXIT   1
#define CONTNE  0
float far alfa, eps, err_curr, maxe, maxep;
float far *wtptr[NMXHLR+1];
float far *outptr[NMXHLR+2];
float far *errptr[NMXHLR+2];
float far *delw[NMXHLR+1];
float far target[NMXINP][NMXOATTR];
float far  input[NMXINP][NMXIATTR];
float far  ep[NMXINP];
float huge outpt[NMXINP][NMXOATTR];
float PRD,cn,cn1[6000],cn2[6000];
int nunit[NMXHLR+2],nhlayer,ninput,ninattr,noutattr,sd;
int result,i,j,k,c,i78,i79,gdriver = DETECT,gmode,errorcode;
long int cnt_num,cnt;
int nsnew,pih,pih1;
int hih,gos,fft;
char task_name[20];
char *err_file="criter.dat";
char def[30],def1[20];
char *sec[3]={"1-EGITIM ISLEMI ","2-CIKIS GRAFIGI","3-CIKIS"};
char atama1[10], fnam[20],atama2[10],atama3[10];
double fe,fe1,de,se,cnt1;
FILE *fp1,*fp2,*fp3,*fopen(),*fr;
void *p;
long randseed=568731L;

//*************************************************

void git()
{ clrscr();printf("YAPILAN  İTERASYON = %ld",cnt);
  printf("\nŞU ANKİ HATA    = %f\n",err_curr);
```

```
printf("ÇIKMAK İSTİYOR MUSUNUZ ? 'd(dur)-HERHANGİ BİR TUŞA BASINIZ");
 getch();i=getch();if(i=='d')cnt=cnt_num-1;clrscr();printf("Bilgisayar hesap yapıyor");
 }
//*****************************************
int random()
 {
 randseed=15625L*randseed+22221L;
 return((randseed>>16)&0x7FFF);
 }
//*****************************
init()
{
 int len1,len2,k;
 float *p1,*p2,*p3,*p4;
 len1=len2=0;
 nunit[nhlayer+2]=0;
 for(i=0;i<(nhlayer+2);i++){
 len1+=(nunit[i]+1)*nunit[i+1];
 len2+=nunit[i]+1;
 }
 p1=(float *) calloc(len1+1,sizeof(float));
 p2=(float *) calloc(len2+1,sizeof(float));
 p3=(float *) calloc(len2+1,sizeof(float));
 p4=(float *) calloc(len1+1,sizeof(float));
 wtptr[0]=p1;
 outptr[0]=p2;
 errptr[0]=p3;
 delw[0]=p4;
 for (i=1;i<(nhlayer+1);i++){
 wtptr[i]=wtptr[i-1]+nunit[i]*(nunit[i-1]+1);
 delw[i]=delw[i-1]+nunit[i]*(nunit[i-1]+1);
 }
  for (i=1;i<(nhlayer+2);i++){
 outptr[i]=outptr[i-1]+nunit[i-1]+1;
 errptr[i]=errptr[i-1]+nunit[i-1]+1;
 }
 for (i=0;i<nhlayer+1;i++){
 *(outptr[i]+nunit[i])=1.0;
 }
 return(0);
 }
//*******************************************
initwt()
 {
 for (j=0;j<nhlayer+1;j++)
 for (i=0;i<(nunit[j]+1)*nunit[j+1];i++){
   *(wtptr[j]+i)=random()/pow(2.0,15.0)-0.5;
   *(delw[j]+i)=0.0;
  }
  return(0);
  }
//*****************************************
set_up()
{
alfa=0.9;
printf("\n alfa momentum katsayısı (default=0.9)?: ");
scanf("%f",&alfa);
eps=0.7;
printf("\n epsilon "Öğrenme oranı(default=0.7)?: ");
```

```
scanf("%f",&eps);
maxe=0.0000001;maxep=0.0000001;
// printf("\nMaksimum toplam hata (default=0.00001)?: ");
// scanf("%f",&maxe);
//  printf("\nMaksimum başlangıç hatası(default=0.00001)?: ");
// scanf("%f",&maxep);
cnt_num=1000;
printf("\nMaksimum iterasyon sayısı (default=1000)?: ");
scanf("%ld",&cnt_num);
printf("\nGizli katman sayısı?: ");
scanf("%d",&nhlayer);
for(i=0;i<nhlayer;i++) {
printf("\n\t %d. gizli katmandaki düğüm sayısı ?:",i+1);
scanf("%d",&nunit[i+1]); }
clrscr();
printf("\n Bilgisayar hesap yapıyor");
nunit[nhlayer+1]=noutattr;
nunit[0]=ninattr;
return(0);
 }
//*************************************************
dwrite(char *taskname)
          {
        char var_file_name[20];
          strcpy(var_file_name,taskname);
          strcat(var_file_name,"_v.dat");
          if ((fp1=fopen(var_file_name,"w+"))==NULL)
          {
          perror("\n Veri dosyası açılamıyor");
          exit(0);
          }
          fprintf(fp1,"%u %u %u %f %f %u
%lu\n",ninput,noutattr,ninattr,alfa,eps,nhlayer,cnt_num);

          for (i=0; i<nhlayer+2;i++){fprintf(fp1,"%d ",nunit[i]);}
          fprintf(fp1,"\n");
          for (i=0; i<ninput;i++){
            for (j=0; j<noutattr;j++)
              fprintf (fp1,"%f",outpt[i][j]);
              fprintf (fp1,"\n");
             }
          if ((c=fclose(fp1))!=0)
            printf("\n %d dosya kapat□lam□yor",c);
          return(0);
          }
//*************************************************
wtwrite( char *taskname)
          {
          char wt_file_name[20];
          strcpy(wt_file_name,taskname);
          strcat(wt_file_name,"_w.dat");
          if ((fp2=fopen(wt_file_name,"w+"))==NULL)
          {
          perror("\n Veri dosyası açılamıyor");
          exit(0);
          }
           k=0;
          for (i=0; i<nhlayer+1;i++)
             for (j=0; j<(nunit[i]+1)*nunit[i+1];j++){
```

```
            if(k==8) {
            k=0;
            fprintf (fp2,"\n");
            }
            fprintf (fp2,"%f ",*(wtptr[i]+j));
            k++;
          }
          if ((c=fclose(fp2))!=0)
           printf("\n%d Dosya kapatılamıyor",c);
          return(0);
          }
//*********************************************************
void forward(i)
          {
           int m,n,p,offset;
           float net;
          for (m=0; m<ninattr;m++)
            *(outptr[0]+m)=input[i][m];
            for (m=1; m<nhlayer+2;m++){
            for (n=0; n<nunit[m];n++){
         net=0.0;
             for (p=0; p<nunit[m-1]+1;p++){
             offset=(nunit[m-1]+1)*n+p;
             net += *(wtptr[m-1]+offset)
                  *(*(outptr[m-1]+p));
                  }
                  *(outptr[m]+n)= 1/(1+exp(-net));
             }
             }
           for(n=0; n<nunit[nhlayer+1];n++)
            outpt[i][n]=*(outptr[nhlayer+1]+n);
            }

//*************************************************************
int introspective (int nfrom,int nto)
    {
    int flag;
    if (cnt>=cnt_num) return(FEXIT);
    nsnew=0;
    flag=1;
    for (i=nfrom;(i<nto)&&(flag==1);i++){
    if (ep[i]<=maxep)nsnew++;
    else flag=0;
    }
    if (flag==1) return(SEXIT);
    if (err_curr<=maxe) return(SEXIT);
    return(CONTNE);
    }
//*************************************************************

int rumelhart(int from_snum,int to_snum)
    {
    int m,n,p,offset,index;
    float out;
    char *err_file="criter.dat";
    cnt=0;
    result=CONTNE;
    if ((fp3=fopen(err_file,"w"))==NULL)
    {
```

```
    perror("Veri dosyası açılamıyor");
    exit(0);
    }
    do {
    err_curr=0.0;
    for (i=from_snum;i<to_snum;i++){
    forward(i);
    for(m=0;m<nunit[nhlayer+1];m++){
     out=*(outptr[nhlayer+1]+m);
    *(errptr[nhlayer+1]+m)=(target[i][m]-out)
                          *(1-out)*out;
   }
  for (m=nhlayer+1;m>=1;m--){
  for (n=0; n<nunit[m-1]+1;n++){
   *(errptr[m-1]+n)=0.0;
   for (p=0; p<nunit[m];p++){
    offset=(nunit[m-1]+1)*p+n;
   *(delw[m-1]+offset)=alfa*(*(errptr[m]+p))
   *(*(outptr[m-1]+n))
    +eps*(*(delw[m-1]+offset));
   *(errptr[m-1]+n)+=*(errptr[m]+p)
          *(*(wtptr[m-1]+offset));
   }
   *(errptr[m-1]+n)=*(errptr[m-1]+n)*
        (1-*(outptr[m-1]+n))*(*(outptr[m-1]+n));
        }
        }
    for (m=1;m<nhlayer+2;m++){
        for(n=0;n<nunit[m];n++){
        for(p=0;p<nunit[m-1]+1;p++){

            offset=(nunit[m-1]+1)*n+p;
     *(wtptr[m-1]+offset)+=*(delw[m-1]+offset);
     }
     }
     }
    ep[i]=0.0;
    for(m=0;m<nunit[nhlayer+1];m++){
     ep[i]+=(fabs((target[i][m]-*(outptr[nhlayer+1]+m))))*(fabs((target[i][m]-
*(outptr[nhlayer+1]+m))));
                }
     err_curr+=0.5*ep[i];
     }
     err_curr=err_curr/nunit[nhlayer+1];

     if(kbhit())git();
     fprintf(fp3,"%ld%f\n",cnt,err_curr);
     cnt++;
     result=introspective(from_snum,to_snum);
     }
     while (result==CONTNE);


    if ((fr=fopen("sonuc.dat","w"))==NULL)
    {
    perror("Veri dosyası açılamıyor");
    exit(0);
     }
    for (i=0; i<ninput; i++)
```

```
        for (j=0;j<noutattr;j++){
  //    printf("\n\n sample %d output %d=%f target %d=%f",
   //       i,j,outpt[i][j],j,target[i][j]);
            fprintf(fr,"%f\n",outpt[i][j]);}
            printf("\n\n Toplam iterasyon sayısı %ld.",cnt);
            printf("\n Normalize edilmiş sistem hatası %f\n\n\n",err_curr);
    fclose(fr);
    return(result);  }
//*********************************************************
user_session()
    { int showdata;
        char fnam[20],dtype[20];
        FILE *fp;
        printf("\n Eğitme safhasının başı ");
        printf("\n\n\n Veri dosyasının adını giriniz (taskname): ");
        scanf("%s",task_name);
        printf("\n Kaç tane giriş  düğümü(veri sayısı?:");
        scanf("%d",&ninattr);
        printf("\n Kaç tane çıkış düğümü?: ");
        scanf("%d",&noutattr);
        printf("\n Toplam kaç tane EKG peryodu?: ");
        scanf("%d",&ninput);
        pih1=ninput*200;
        gos=ninput;
        strcpy(fnam,task_name);
        strcat(fnam,".dat");
        printf("\n Giriş dosyasının adı %s",fnam);
        if ((fp=fopen(fnam,"r"))==NULL)
        {
        printf("\n %s dosyası yok", fnam);
        exit(0);
        }
        printf("\n Verilere bakmak ister misiniz ?");
        printf("\n   evet veya hayır :");
        scanf("%s",dtype);
        showdata=((dtype[0]=='e')||(dtype[0]=='E'));
         for (i=0;i<ninput;i++){
           for (j=0;j<ninattr;j++) {
           fscanf(fp,"%d %f",&pih,&input[i][j]);
           if (showdata)printf("%f  ",input[i][j]);
           }}
          fclose(fp);

          printf("\n\t Çıkış dosyası adı giriniz  : ");
          scanf("%s",task_name);
          strcpy(fnam,task_name);
          strcat(fnam,".dat");

          if ((fp=fopen(fnam,"r"))==NULL)
        {
         printf("\n %s dosyası yok", fnam);
         exit(0);
         }

          for (i=0;i<ninput;i++){
          for (j=0;j<noutattr;j++) {

           fscanf(fp,"%d %f",&pih,&target[i][j]);
           if (showdata)printf("%f  ",target[i][j]);
```

```
            }
            }
            if ((i=fclose(fp))!=0)
           {
            printf("\n %d  Dosya kapatılamıyor ",i);
            exit(0);
            }

             return(0);
     }
//*****************************************************
training()
     {

     int result;
     user_session();
     set_up();
     init();
     do {
          initwt();
          result=rumelhart(0,ninput);
          } while (result==RESTRT);
          dwrite(task_name);
          wtwrite(task_name);
          return(0);
          }
//*****************************************************
void der()
{
 FILE *f;
 fft=i78;
 se=550.0;
 de=fft/se;
 hih=floor(de);

 if ((f=fopen(fnam,"r"))==NULL)
{
 perror("Dosya açılamaz");
 exit(0);
 }
for(i=0;i<i79;i++)fscanf(f," %d %f\n",&cnt1,&cn);
sd=0;
for(i=1;i<fft;i++)
{ fscanf(f," %d %f\n",&cnt1,&cn);
if(i/(hih+1.0)==floor(i/(hih+1.0))){ sd=sd+1;cn1[sd]=cn*10;}

 }
   if ((i=fclose(f))!=0)
   printf("\n%d dosya kapatılamıyor",i);
 }
//*****************************************************
void  *cevre(int x1,int y1,int x2,int y2,int pattip,int icrenk,int cevrerenk)
 {
   unsigned int d;
   void *p  ;
   setfillstyle(pattip,icrenk);
   setcolor(cevrerenk);
   rectangle(x1,y1,x2,y2);
   floodfill(x1+1,y1+1,cevrerenk);
```

```
   d=imagesize(x1,y1,x2,y2);
   p=malloc(d);
   getimage(x1,y1,x2,y2,p);
   cleardevice();
   return(p);
    }

//*************************************************
int  main1()
{  char select[20],cont[20];
    strcpy(task_name,"**********");
    do {
    printf("\n ******** EĞİTME ********\n");
    training();

            printf("\nDevam etmek istiyor musunuz?   ");
            scanf("%s",cont);
            }

         while ((cont[0]=='e')||(cont[0]=='E'));
         return(0);
}
//*****************************************************************
int main(void)
{    int gdriver = DETECT, gmode,fr,ht,cg,ii,jj, errorcode,i=90,j=210,k,ch;
   FILE *fk,*fp7;
   float r;
   initgraph(&gdriver, &gmode, "");
   errorcode = graphresult();
   if (errorcode != grOk)
  {
    printf("Grafik hatası: %s\n", grapherrormsg(errorcode));
    printf("Herhangi bir tuşa basınız:");
    getch();
    exit(1);
  }
  p=cevre(0,0,220,20,5,1,2);
settextstyle(4,0,9);
 rer:   i=90;ii=90;jj=j;
   goto der;

  while(!0){
  tek:ht=20; ii=i;jj=j;ch=getch();
  if(ch==80) j=j+ht; else if(ch==72) j=j-ht; else
  if(ch==27) { printf("esc basıldı");
  goto cik;}
  else if((ch==13)&&(j==210))
{clrscr();restorecrtmode();main1();setgraphmode(getgraphmode());cleardevice();i=90;ii=90;jj=j;}else
          if((ch==13)&&(j==230))goto dfd; else
           if((ch==13)&&(j==250)) exit(1); else  goto tek;
 der:  if(j<210)j=250; else if(j>250)j=210;
  moveto(100,220);outtext(sec[0]);
  moveto(100,240);outtext(sec[1]);
  moveto(100,260);outtext(sec[2]);
   putimage(ii,jj,p,1);
   putimage(i,j,p,2);
 }
 cik: free(p);
 closegraph();
```

```
getch();
return 0;
dfd:cleardevice();
strcpy(fnam,"averi.dat");
printf("\n Kaçıncı Veriden ?: ");
scanf("%d",&i79);
printf("\n Kaç Veri Çizilecek ?: ");
scanf("%d",&i78);

der();
cleardevice();
int x1=0,x2,y1,y11,y2 ;
line(50, getmaxy()-50,50 , 0);
line(50,getmaxy()-50,getmaxx()-50,getmaxy()-50);
strcpy(def,"Görünen veri sayısı= ");itoa(fft,def1,10);strcat(def,def1);
moveto(200,440);outtext(def);
settextstyle(4,1,9);
moveto(30,230);outtext("OLUSTURULAN VERI");
settextstyle(4,0,9);
settextstyle(4,1,9);
moveto(30,70);outtext("ORJINAL VERI");
settextstyle(4,0,9);
settextstyle(4,1,9);
moveto(30,392);outtext("HATA");
settextstyle(4,0,9);
moveto(33,405);outtext("*5");
 for(i=2;i<sd;i++)
 {
 fe=cn1[i]*22;
 y1=floor(fe);
 fe1=cn1[i-1]*22;
 y11=floor(fe1);
 setcolor(14);
 line(i+50,getmaxy()-49-y1-220,i+51,getmaxy()-49-y11-220);
  }
//*********************************************************
  if(pih1!=0)
 {
  if ((fp7=fopen("averi.dat","r"))==NULL)
 {
 perror("Dosya açılamıyor");
 exit(0);
 }
  if ((fp1=fopen("sonuc.dat","r"))==NULL)
 {
 perror("Dosya açılamıyor");
 exit(0);
 }
  if((fk=fopen("fark.dat","w"))==NULL)
 {
 perror("Dosya açılamıyor");
 exit(0);
 }
  if((fp2=fopen("ark.dat","w"))==NULL)
 {
 perror("Dosya açılamıyor");
 exit(0);
 }
 for(i=0;i<pih1;i++)
```

```
  {
  fscanf(fp1,"%d %f",&pih,&cn2[i]);
  fscanf(fp7,"%d %f",&pih,&cn1[i]);
  cn1[i]-=cn2[i];
  fprintf(fk,"%d %f",pih,cn1[i]);
  }
  r=0.0; PRD=0.0;
     for(i=0;i<pih1;i++) {
            PRD+=pow(cn1[i],2);
            r +=pow(cn2[i],2);
           }
            PRD=sqrt(PRD/r)*100;

   fprintf(fp2,"%f %d",PRD,gos);

  fclose(fp7);fclose(fp1);fclose(fk);fclose(fp2);}
//*****************************************
  if((fp2=fopen("ark.dat","r"))==NULL)
 {
  perror("Dosya a‡□lam□yor");
  exit(0);
  }
  fscanf(fp2,"%f %d",&PRD,&gos);
  fclose(fp2);
  strcpy(def,"PRD     =%");gcvt(PRD,6,def1);strcat(def,def1);
  moveto(470,30);outtext(def);
  strcpy(def,"Toplam EKG peryodu=");itoa(gos,def1,10);strcat(def,def1);
  moveto(470,20);outtext(def);
  strcpy(fnam,"sonuc.dat");
  der();
  for(i=2;i<sd;i++)
  {
   fe=cn1[i]*22;
   y1=floor(fe);
   fe1=cn1[i-1]*22;
   y11=floor(fe1);
   setcolor(9);
   line(i+50,getmaxy()-49-y1-40,i+51,getmaxy()-49-y11-40);
   }
//************************************************************
   strcpy(fnam,"fark.dat");
   der();
   for(i=2;i<sd;i++)
   {
   fe=cn1[i]*110;
   y1=floor(fe);
   fe1=cn1[i-1]*110;
   y11=floor(fe1);
   setcolor(10); //7 beyaz renk
   line(i+50,getmaxy()-49-y1-20,i+51,getmaxy()-49-y11-20);
   }
//*****************************************************
   getch();
   cleardevice();
   goto rer;
}
```

EK 2

SIKIŞTIRMA VE YENİDEN YAPILANMA İŞLEMLERİNİ YAPAN PROGRAM

```
#include <stdio.h>
#include <graphics.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
#include <alloc.h>
#include<ctype.h>
#define NMXUNIT   10
#define NMXHLR   10
#define NMXOATTR  240
#define NMXINP  50
#define NMXIATTR  240
#define SEXIT     3
#define RESTRT   2
#define FEXIT   1
#define CONTNE  0
float far alfa, eps;
float far *wtptr[NMXHLR+1];
float far *outptr[NMXHLR+2];
float far  input[NMXINP][NMXIATTR];
float huge outpt[NMXINP][NMXOATTR];
int nunit[NMXHLR+2],nhlayer,ninput,ninattr,noutattr,sd;
int result,i,j,i78,i79,gdriver = DETECT,gmode,errorcode;
long int cnt_num;
char task_name[20];
FILE *fp1,*fp2,*fp3,*fopen(),*fr;
int pih,pih1;
char def[30],def1[20];
char *sec[3]={"1-SIKISTIRMA VE ACMA ISLEMI ","2-CIKIS GRAFIGI","3-CIKIS"};
void *p;
char atama1[10], fnam[20],atama2[10],atama3[10];
int hih,gos,fft,j9=0;
float PRD,cn,cn1[6000],cn2[6000];
double fe,fe1,de,se,cnt1;

//*************************************************************
init()
 {
  int len1,len2;
  float *p1,*p2;
  len1=len2=0;
  nunit[nhlayer+2]=0;
  for(i=0;i<(nhlayer+2);i++){
  len1+=(nunit[i]+1)*nunit[i+1];
  len2+=nunit[i]+1;
  }
```

```
  p1=(float *) calloc(len1+1,sizeof(float));
  p2=(float *) calloc(len2+1,sizeof(float));
  wtptr[0]=p1;
  outptr[0]=p2;
  for (i=1;i<(nhlayer+1);i++){
  wtptr[i]=wtptr[i-1]+nunit[i]*(nunit[i-1]+1);
  }
   for (i=1;i<(nhlayer+2);i++){
  outptr[i]=outptr[i-1]+nunit[i-1]+1;
   }
  for (i=0;i<nhlayer+1;i++){
  *(outptr[i]+nunit[i])=1.0;
  }
  return(0);
  }
//*****************************************************************
dread(char *taskname)
   {
    int c;
    char var_file_name[20];
    strcpy(var_file_name,taskname);
    strcat(var_file_name,"_v.dat");
    if (( fp1=fopen(var_file_name,"r"))==NULL)
    {
     perror("\n Veri dosyası açılamıyor ");
     exit(0);
     }
fscanf(fp1,"%u%u%u%f%f%u%lu\n",&ninput,&noutattr,&ninattr,&alfa,&eps,&nhlayer,&cnt_num)
;
    for(i=0; i<nhlayer+2; i++)
      fscanf(fp1,"%d",&nunit[i]);
    if ((c=fclose(fp1))!=0)
    printf("\n%d dosya kapatılamıyor",c);
    return(0);
     }

//*************************************************
wtread(char *taskname)
   {
        int c;
          char wt_file_name[20];
          strcpy(wt_file_name,taskname);
          strcat(wt_file_name,"_w.dat");
          if ((fp2=fopen(wt_file_name,"r"))==NULL)
          {
           perror("\n Veri dosyası açılamıyor");
           exit(0);
           }
           for (i=0; i<nhlayer+1;i++){
             for (j=0; j<(nunit[i]+1)*nunit[i+1];j++)
           { fscanf (fp2,"%f",(wtptr[i]+j)); }
            }
           if ((c=fclose(fp2))!=0)
            printf("\n%d dosya kapatılamıyor",c);
           return(0);
           }
```

```
//************************************************
void forward(i)
            {
             int m,n,p,offset;
             float net;
             for (m=0; m<ninattr;m++)
            *(outptr[0]+m)=input[i][m];
             for (m=1; m<nhlayer+2;m++){
             for (n=0; n<nunit[m];n++){
             net=0.0;
             for (p=0; p<nunit[m-1]+1;p++){
             offset=(nunit[m-1]+1)*n+p;
             net += *(wtptr[m-1]+offset)
            *(*(outptr[m-1]+p));
                     }
            *(outptr[m]+n)= 1/(1+exp(-net));
                 }
                 }
            for(n=0; n<nunit[nhlayer+1];n++)
            outpt[i][n]=*(outptr[nhlayer+1]+n);
                }
//*****************************************************************
reconstruction()
    {  FILE *fp7;
            int m,nsample;
            char ans[10];
            char dfile[20];
            printf("\n Yeni bir veri girdisi için çıkışın üretilmesi");
            printf("\n\n Şimdiki ağırlık değerleri dosyası (task name) *** %s *** dir", task_name);
            printf("\n Farklı bir ağırlık dosyası (task) olsun mu?");
            printf("\n evet veya hayır :");
            scanf("%s",ans);
            if ((ans[0]=='e')||(ans[0]=='E'))
           {
            printf("\n Ağırlık değerleri dosyası (task) adını giriniz : ");
            scanf("%s",task_name);
            dread(task_name);
            init();
            wtread(task_name);
            }
  printf("\n Açılacak veri dosyası adı : ");
  scanf("%s",dfile);
  if ((fp1=fopen(dfile,"r"))==NULL)
  {
   perror("Dosya açılamıyor");
   exit(0);
    }
  if ((fp7=fopen("sonuc.dat","w"))==NULL)
  {
  perror("Dosya açılamıyor");
  exit(0);
  }
  printf("\n Açılacak Örnek sayısı :  ");
  scanf("%d",&nsample);
  pih1=nsample*200;
  gos=nsample;
  for (i=0;i<nsample;i++)
  for (m=0; m<ninattr;m++)
  { //printf("\n%d veri=",m+1);
```

```
  fscanf(fp1,"%d %f",&pih,&input[i][m]); }
  for (i=0; i<nsample;i++)
  {
   forward(i);
   for(m=0;m<noutattr;m++)
  { j9++;
    fprintf(fp7," %d %f\n",j9,*(outptr[nhlayer+1]+m));
    printf("\n Örnek %d çıkış %d=%f",
    i,m,*(outptr[nhlayer]+m));}
    }
   printf("\nÇıkış üretildi ");
   fclose(fp7);fclose(fp1);
   return(0);
        }
//*********************************************
compression()
        {
         FILE *fp7;
         int m,nsample;
         char ans[10];
         char dfile[20];
        printf("\n Yeni bir veri girdisi için çıkışın üretilmesi");
        printf("\n\n Şimdiki ağırlık değerleri dosyası(task name) *** %s *** dir", task_name);
        printf("\n Farklı bir ağırlık dosyası(task) olsun mu?");
        printf("\n evet veya hayır :");
        scanf("%s",ans);
         if ((ans[0]=='e')||(ans[0]=='E'))
        {
         printf("\n Ağırlık değerleri dosyası (task) adını giriniz : ");
         scanf("%s",task_name);
         dread(task_name);
         init();
         wtread(task_name);
          }
 printf("\n Sıkıştırılacak dosya adı : ");
 scanf("%s",dfile);
 if ((fp1=fopen(dfile,"r"))==NULL)
 {
 perror("Dosya açılamıyor");
 exit(0);
 }
 if ((fp7=fopen("sdeg.dat","w"))==NULL)
 {
 perror("Dosya açılamıyor");
 exit(0);
 }
 printf("\n Sıkıştırılacak örnek sayısı :       ");
 scanf("%d",&nsample);
 for (i=0;i<nsample;i++)
 for (m=0; m<ninattr;m++)
{ //printf("\n%d veri=",m+1);
 fscanf(fp1,"%d %f",&pih,&input[i][m]); }
 for (i=0; i<nsample;i++)
 {
  forward(i);
  for(m=0;m<nunit[1];m++)
   {j9++;
    fprintf(fp7," %d %f\n",j9,*(outptr[nhlayer]+m));
    printf("\n Örnek %d çıkış %d=%f",
```

```
      i,m,*(outptr[nhlayer]+m));}
     }
      printf("\nÇıkış üretildi ");
     fclose(fp7);fclose(fp1);
     return(0);
      }
//*************************************************
void der()
 {
  FILE *f;
   fft=i78;
   se=550.0;
   de=fft/se;
   hih=floor(de);
   if ((f=fopen(fnam,"r"))==NULL)
   {
   perror("Dosya açılamıyor");
   exit(0);
   }
  for(i=0;i<i79;i++)fscanf(f," %d %f\n",&cnt1,&cn);
  sd=0;
  for(i=1;i<fft;i++)
  {
   fscanf(f," %d %f\n",&cnt1,&cn);
   if(i/(hih+1.0)==floor(i/(hih+1.0))){ sd=sd+1;cn1[sd]=cn*9;}
  }
   if ((i=fclose(f))!=0)
   printf("\n%d dosya kapatılamıyor",i);
 }
//************************************************************
void  *cevre(int x1,int y1,int x2,int y2,int pattip,int icrenk,int cevrerenk)
 {
   unsigned int d;
   void *p  ;
   setfillstyle(pattip,icrenk);
   setcolor(cevrerenk);
   rectangle(x1,y1,x2,y2);
   floodfill(x1+1,y1+1,cevrerenk);
   d=imagesize(x1,y1,x2,y2);
   p=malloc(d);
   getimage(x1,y1,x2,y2,p);
   cleardevice();
   return(p);
    }
//*****************************************************
int  main1()
{    char select[20],cont[20];
     strcpy(task_name,"**********");
      do {
      printf("\n ***** S(SIKIŞTIRMA) veya A(AÇMA) seçiniz ******\n");
           do {
             scanf ("%s",select);
             switch(select[0]){
             case 's':
             case 'S':
             compression();
             break;
               case 'a':
               case 'A':
```

```
            reconstruction();
            break;
            default:
            printf("\n sıkıştırma veya açmayı seçiniz");
    break;
    goto d12;
    }
        }
          while ((select[0]!='a')&&(select[0]!='A')
          &&(select[0]!='s')&&(select[0]!='S'));
     d12:          printf("\nDevam etmek istiyor musunuz?");
          scanf("%s",cont);
          }
         while ((cont[0]=='e')||(cont[0]=='E'));
         return(0);
}
//*************************************************************
int main(void)
{  int gdriver = DETECT, gmode,fr,ht,cg,ii,jj, errorcode,i=90,j=210,k,ch;
   FILE *fp7,*fk;
   float r;
  initgraph(&gdriver, &gmode, "");
  errorcode = graphresult();
  if (errorcode != grOk)
 {
   printf("Grafik hata: %s\n", grapherrormsg(errorcode));
   printf("Herhangi bir tuşa basın:");
   getch();
   exit(1);
 }
 p=cevre(0,0,220,20,5,1,2);
 settextstyle(4,0,9);
rer:  i=90;ii=90;jj=j;
  goto der;
  while(!0){
tek:ht=20; ii=i;jj=j;ch=getch();
  if(ch==80) j=j+ht; else if(ch==72) j=j-ht; else
   if(ch==27) { printf("esc basildi");
    goto cik;}
    else if((ch==13)&&(j==210))
{clrscr();restorecrtmode();main1();setgraphmode(getgraphmode());cleardevice();i=90;ii=90;jj=j;}else
          if((ch==13)&&(j==230))goto dfd; else
          if((ch==13)&&(j==250)) exit(1); else  goto tek;
der:  if(j<210)j=250; else if(j>250)j=210;
 moveto(100,220);outtext(sec[0]);
 moveto(100,240);outtext(sec[1]);
 moveto(100,260);outtext(sec[2]);
  putimage(ii,jj,p,1);
  putimage(i,j,p,2);
          }
 cik: free(p);
 closegraph();
 getch();
 return 0;
dfd:cleardevice();
 strcpy(fnam,"averi.dat");
 printf("\n Kaçıncı Veriden ?:");
 scanf("%d",&i79);
 printf("\n Kaç Veri Çizilecek ?:");
```

```
scanf("%d",&i78);
der();
cleardevice();
int x1=0,x2,y1,y11,y2 ;
 line(50, getmaxy()-50,50 , 0);
 line(50,getmaxy()-50,getmaxx()-50,getmaxy()-50);
 strcpy(def,"Görünen veri sayisi=");itoa(fft,def1,10);strcat(def,def1);
 moveto(200,440);outtext(def);
 settextstyle(4,1,9);
 moveto(30,230);outtext("OLUSTURULAN VERI");
 settextstyle(4,0,9);
 settextstyle(4,1,9);
 moveto(30,70);outtext("ORJINAL VERI");
 settextstyle(4,0,9);
 settextstyle(4,1,9);
 moveto(30,392);outtext("HATA");
 settextstyle(4,0,9);
 moveto(33,405);outtext("*5");

 for(i=2;i<sd;i++)
{
 fe=cn1[i]*22;
 y1=floor(fe);
 fe1=cn1[i-1]*22;
 y11=floor(fe1);
 setcolor(14);  //14 sari   (7)
 line(i+50,getmaxy()-49-y1-220,i+51,getmaxy()-49-y11-220);
 }

  if(pih1!=0)
{
 if ((fp7=fopen("averi.dat","r"))==NULL)
{
perror("Dosya açılamıyor");
exit(0);
}
 if ((fp1=fopen("sonuc.dat","r"))==NULL)
{
perror("Dosya açılamıyor");
exit(0);
}
 if((fk=fopen("fark.dat","w"))==NULL)
{
perror("Dosya açılamıyor");
exit(0);
}
 if((fp2=fopen("ark.dat","w"))==NULL)
{
perror("Dosya açılamıyor");
exit(0);
}
for(i=0;i<pih1;i++)
{
fscanf(fp1,"%d %f",&pih,&cn2[i]);
fscanf(fp7,"%d %f",&pih,&cn1[i]);
cn1[i]-=cn2[i];
fprintf(fk,"%d %f",pih,cn1[i]);
}
r=0.0; PRD=0.0;
```

```
                for(i=0;i<pih1;i++) {
                        PRD+=pow(cn1[i],2);
                        r +=pow(cn2[i],2);
                       }
                PRD=sqrt(PRD/r)*100;

  fprintf(fp2,"%f %d",PRD,gos);
  fclose(fp7);fclose(fp1);fclose(fk);fclose(fp2);}

if((fp2=fopen("ark.dat","r"))==NULL)
  {
  perror("Dosya açılamıyor");
  exit(0);
  }
   fscanf(fp2,"%f %d",&PRD,&gos);
   fclose(fp2);
   strcpy(def,"PRD     =%");gcvt(PRD,6,def1);strcat(def,def1);
   moveto(470,30);outtext(def);
   strcpy(def,"Toplam EKG peryodu=");itoa(gos,def1,10);strcat(def,def1);
   moveto(470,20);outtext(def);
   strcpy(fnam,"sonuc.dat");
   der();
   for(i=2;i<sd;i++)
   {
   fe=cn1[i]*22;
   y1=floor(fe);
   fe1=cn1[i-1]*22;
   y11=floor(fe1);
   setcolor(12);  //12 kirmizi renk  (15 gri)
   line(i+50,getmaxy()-49-y1-40,i+51,getmaxy()-49-y11-40);
   }
//*****************************************************
   strcpy(fnam,"fark.dat");
   der();
   for(i=2;i<sd;i++)
   {
   fe=cn1[i]*110;
   y1=floor(fe);
   fe1=cn1[i-1]*110;
   y11=floor(fe1);
   setcolor(7); //7 beyaz renk
   line(i+50,getmaxy()-49-y1-20,i+51,getmaxy()-49-y11-20);
   }
//*****************************************************
   getch();
   cleardevice();
   goto rer;
}
```

ÖZGEÇMİŞ

Yücel Koçyiğit 1971 yılında Trabzon'da doğdu. 1988 yılında İstanbul Pendik Lisesinden mezun olarak, aynı yıl İstanbul Teknik Üniversitesi, Elektrik-Elektronik Fakültesi, Elektronik ve Haberleşme Bölümü'ne girdi. 1992 yılında bu bölümden mezun olarak, aynı yıl İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik ve Haberleşme Anabilim Dalı Biyomedikal Mühendislik lisansüstü programına başladı.

Yücel Koçyiğit halen, Manisa Celal Bayar Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü'nde araştırma görevlisi olarak çalışmaktadır.