BİR YAPAY SİNİR AĞI BİLEŞENİ GERÇEKLEMESİ

Mehmet ERDİ
Yüksek Lisans Tezi

Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Ağustos – 2004

JÜRİ VE ENSTİTÜ ONAYI

Mehmet ERDİ' nin **Bir Yapay Sinir Ağı Bileşeni Gerçeklemesi** başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki, Yüksek Lisans tezi 17.08.2004..tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	İmza
Üye (Tez Danışmanı)	**: Yard.Doç.Dr.Yusuf OYSAL**	
Üye	**: Prof.Dr.Ali GÜNEŞ**	
Üye	**: Yard.Doç.Dr.Serkan TAPKIN**	

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 08.09.2004. **tarih ve** .29/21.. **sayılı kararıyla onaylanmıştır.**

Enstitü Müdürü

Prof. Dr. Altuğ İFTAR
Fen Bilimleri Enstitüsü
Müdürü

ÖZET

Yüksek Lisans Tezi

BİR YAPAY SİNİR AĞI BİLEŞENİ GERÇEKLEMESİ

MEHMET ERDİ

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yard.Doç.Dr.Yusuf OYSAL
2004, 70 sayfa

Bu tezde bilgisayar uygulamaları içerisinde kullanılabilecek bir yapay sinir ağı bileşeni gerçeklenmesi amaçlanmıştır. COM (Component Object Model) yapısı kullanılarak değişik uygulama geliştirme platformlarında kullanılabilecek bir yapay sinir ağı bileşeni oluşturulmuştur. Bu bileşenin farklı ileri beslemeli yapay sinir ağı modelleri ve eğitim yöntemleri kullanılarak değişik uygulamalar için kullanılabilmesi amaçlanmıştır.

Anahtar Kelimeler: Yapay Sinir Ağları, COM, ActiveX, Bileşen Nesne Modeli, Çok Katmanlı Perseptronlar

ABSTRACT

Master of Science Thesis

IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK COMPONENT

MEHMET ERDİ

Anadolu University
Graduate School of Natural and Applied Sciences
Computer Engineering Program

Supervisor: Asst.Prof.Dr.Yusuf OYSAL
2004, 70 pages

In this thesis it is aimed to implement a neural network component that could be used for windows applications. Using the COM (Component Object Model) structure a neural network component is implemented that could be used in a variety of application development platforms. The component is aimed to be used in a variety of applications by implementing different neural network structures and training algorithms.

Keywords: Artificial Neural Networks, COM, ActiveX, Component Object Model, Multi Layer Perceptrons

TEŞEKKÜR

Çalışmalarım sırasında benden desteklerini esirgemeyen değerli hocam Yard.Doç.Dr.Yusuf OYSAL' a, hayatımın her anında yanımda olan ve bana destek olan anne ve babama teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

SİMGELER ve KISALTMALAR DİZİNİ

BFGS	: Broyden-Fletcher-Goldfarb-Shanno
COM	: Component Object Model
GUID	: Global Unique Identifier
İBYSA	: İleri Beslemeli Yapay Sinir Ağı
MFC	: Microsoft Foundation Classes
RTFSA	: Radyal Tabanlı Fonksiyon Sinir Ağı
UML	: Unified Modeling Language
YSA	: Yapay Sinir Ağı
w	: Bağlantı ağırlığı
φ	: Aktivasyon fonksiyonu
ε	: Ağ çıkış hatası
Δw	: Ağırlık değişimi
Δg	: Gradyan değişimi
θ	: Eşik değeri
η	: Öğrenme sabiti

Alt ve üst indisler

i : Sinir hücresi indisi

j : Önceki katman sinir hücresi indisi

k : Sinir katmanı indisi

l : İterasyon sırası indisi

K : Sinir katmanı sayısı

N : Girdi vektörü sayısı

1. GİRİŞ

İnsan beyninin çalışma yapısı yüzyıllar boyunca insanın ilgisini çekmiş ve özellikle 20.yy ortalarından itibaren insan beyninin modellenerek benzer işlevleri yerine getirebilecek yapay sistemler kurma çalışmaları hızlanmıştır. İnsan beyninin yapısı matematiksel olarak modellenmeye çalışılmış ve kurulan matematiksel modellerden yola çıkılarak yapay sistemler geliştirilmeye çalışılmıştır.

İnsan beyninin yapay olarak modellenmesinde büyük aşamalar kaydedilmiş olmasına rağmen halen beynin karmaşık yapısını tam olarak modelleyebilecek genel bir yöntem bulunmamaktadır[1,2]. Ancak kullanılacak uygulamanın özelliklerine bağlı olarak geliştirilen bir takım modeller yaygın bir şekilde kullanılmakta ve mevcut modellerin geliştirilmesine yönelik çalışmalar sürdürülmektedir.

Geliştirilen yapay sinir ağ modellerinin, insan beyninin özelliklerine benzer şekilde paralel yapılı olma, doğrusal olmama, genelleme yapabilme, öğrenme ve hata toleransı özelliklerine sahip olması, bu ağların geniş uygulama alanı bulmasını sağlamıştır[1,3-5].

Günümüze kadar yapay sinir ağları çözümü güç ve karmaşık olan veya ekonomik olmayan bir çok farklı problemin çözümünde kullanılmış ve genelde başarılı sonuçlar alınmıştır. Bir çok farklı alana uygulanabilen yapay sinir ağlarının kullanımı, kullanım alanlarına bağlı olarak, tahmin, sınıflandırma, veri ilişkilendirme, veri yorumlama ve veri filtreleme olarak sınıflandırılabilir[1,4].

Bu çalışmada, halen yaygın olarak kullanılmakta olan bu modellerden, ileri beslemeli yapay sinir ağları ele alınmıştır. İleri beslemeli ağlardan çok katmanlı algılayıcılar ve Radyal Tabanlı Fonksiyon Sinir ağları modellenmiş, bu ağ tiplerinin eğitimlerinde kullanılan öğreticili öğrenme metotlardan hata geri yayılım algoritması, konjüge gradyan yöntemler ve BFGS yöntemi kullanılarak yapay sinir ağının eğitilmesini sağlayacak yöntemler gerçeklenmiştir.

Görsel programlama araçlarında kullanılabilecek bir Yapay Sinir Ağı bileşeni gerçeklenip, görsel programlama araçlarında geliştirilecek uygulamalarda

Yapay Sinir Ağları kullanımını kolaylaştırılması ve teorik detaylarına girmeden YSA' nın uygulamalarda kullanılmasının basitleştirilmesi amaçlanmıştır.

Söz konusu çalışma yapılırken Microsoft (c)' un görsel programlama araçlarından Visual C++ kullanılmış, Component Object Modeling(COM) teknolojisi kullanılarak geliştirilen bileşenin çok farklı ortamlarda kullanılabilen bir ActiveX bileşeni gerçeklenmiştir.

Yapay sinir ağları ile ilgili birçok yazılım bulunmaktadır. Bu yazılımlar genel olarak yapay sinir ağı simülatörleri, yapay sinir ağı yazılım kütüphaneleri ve yapay sinir ağı bileşenleri şeklindedir[6-9]. Yapay sinir ağı simülatörleri kendine ait çalışma ortamı bulunan ve verilen parametrelere ve verilere göre yapay sinir ağının çalışmasını modelleyen ve çıktılarını sunan yapıdaki yazılımlardır. Bu yazılımların kendi başına çalışan uygulama programlarında kullanılması genelde mümkün olmamaktadır. Yazılım kütüphaneleri yapay sinir ağları ile ilgili işlevleri, yordamları sağlayan genelde ikili formda sunulan kütüphanelerdir. Bu kütüphaneler kendi başına çalışan uygulama programları içerisinde kullanılabilmektedir. Ancak kütüphanelerin yazılımla birlikte derlenme gereksinimleri ve kullanımı için geniş bir dokümantasyon ve işlevlerin iyi anlaşılması gereksinimi bu kütüphanelerin kullanımını zorlaştırılmaktadır. Yapay sinir ağı bileşenlerinin kullanımı kolay ve esnek olmasına karşın genelde ticari amaçlı olarak sunulması ve beklenen işlevler artıkça maliyetinin yükselmesi uygulama programcılarının geliştirdikleri uygulama içerisinde yapay sinir ağı modüllerini yeniden yazmaya yönlendirmektedir.

Bu çalışma ile yapay sinir ağlarının uygulama programları içerisinde kullanımın kolaylaştırılması, uygulama geliştirme dilinden bağımsız olarak, yapay sinir ağlarının teorik ve gerçekleme detaylarına girilmeden kullanılması amaçlanmaktadır. Bu nedenle çalışmada gerçeklenen bileşenin geliştirilmesinde Microsoft tarafından sunulan COM modeli kullanılmıştır [10].

Gerçeklenen yapay sinir ağı bileşeni içerisinde eğitim yöntemi olarak, konjüge gradyan yöntemler ve ikinci dereceden gradyan bilgisini kullanan eğitim algoritmaları kullanılarak eğitim süresinin azaltılması amaçlanmıştır.

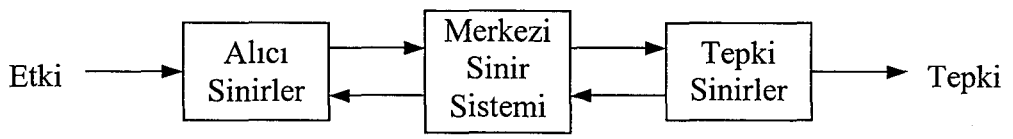
Çalışmanın uygulama bölümünde yapay sinir ağı bileşeninin doğrusal olamayan fonksiyon tahmininde ve bir uzman sistem içerisinde kullanımı

gösterilmiştir. Bu amaçla Visual Basic 6.0 programlama dili kullanılarak geliştirilen bir uygulama yazılımı içerisinde, gerçeklenen yapay sinir ağı bileşeni kullanılmıştır. Kullanılan yapay sinir ağı bileşeni ile, daha önce uzman bilgisi ile elde edilmiş hastalık teşhis sonuçlarından faydalanılarak kullanıcı tarafından girilen belirtilerle kanser hastalığının teşhisi amaçlanmıştır.

2. SİNİR AĞLARI

2.1 Biyolojik Sinir Ağları

Biyolojik sinir sistemi, organizmanın içinden veya dışından alınan bilgiyi, sinir hücreleri aracılığı ile merkezi bir sinir sistemine ve burada oluşturulan tepkileri de organizmanın ilgili yerlerine ileten üç katmanlı bir yapı olarak tanımlanabilir[11]. Merkezi sinir ağında bilgiler, etki ve tepki sinirlerinin ileri ve geri beslemesi yönünde değerlendirilerek işlenir(Şekil 2-1). Bu yönüyle biyolojik sinir sistemi kapalı döngü bir kontrol sistemi olarak düşünülebilir.



Şekil 2-1 Biyolojik sinir sistemin temel gösterimi

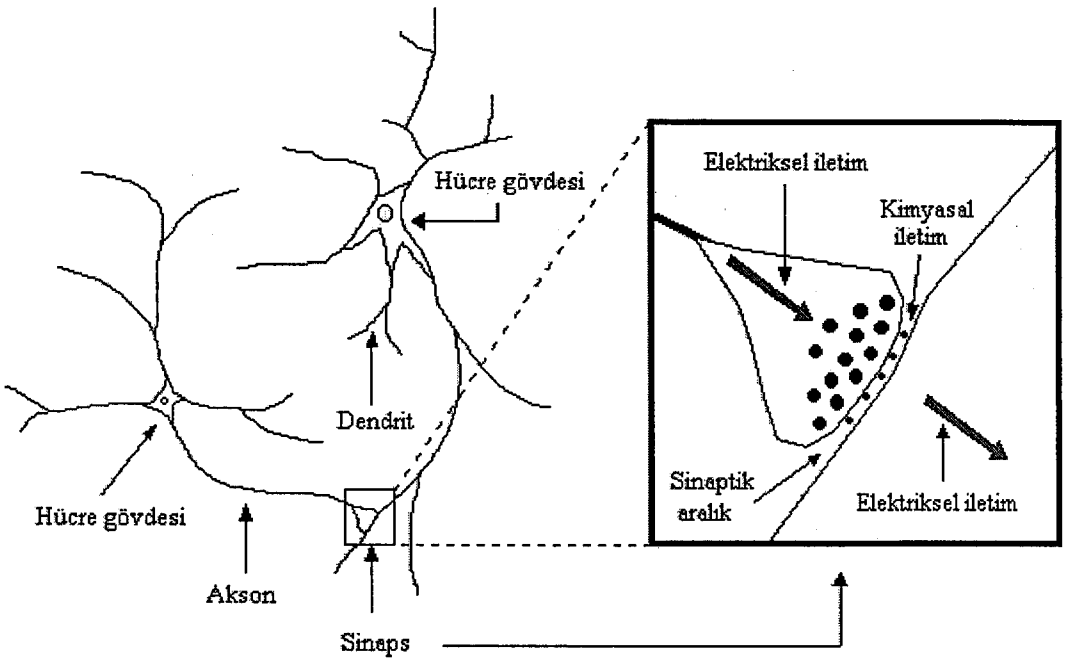
Biyolojik sinir sistemini oluşturan temel yapı taşları birbirine benzer yapıda olan sinir hücreleridir. Sinir hücreleri arasında bilgi iletimi elektriksel ve kimyasal sinyaller aracılığı ile yapılmaktadır. Bir sinir hücresi, dendrit, hücre gövdesi, akson ve sinapsis olmak üzere dört temel bileşenden oluşmaktadır(Şekil 2-2). Bu bileşenlerin karmaşık bir çalışma yapısı vardır ancak her bir bileşenin görevleri basit olarak su şekilde özetlenebilir:

a. Dendrit : Dentritler bir sinir hücresine dışarıdan gelen elektriksel sinyalleri alan yapılardır. Dentritler bir ağaç yapısında olup hücre gövdesine bağlanırlar.

b. Hücre Gövdesi : Hücreye giren sinyalleri toplayıp hücre çıkışına yönlendiren yapıdır.

c. Akson : Elektriksel sinyaller olarak gelen verileri hücre dışına taşıyan yapılardır.

d. Sinapsis : Hücreler arasında bağlantıların kurulduğu yapılardır.



Şekil 2-2 Biyolojik sinir ağı Yapısı

Bir sinir hücresinde bilgi iletimi şu şekilde olur: Diğer hücrelerden gelen sinyaller dendritler tarafından hücre gövdesine gönderilir. Hücre gövdesine gelen bu sinyaller toplanır ve belirli bir eşik seviyesinin üzerine çıktığında hücre tarafından bir elektriksel darbe oluşturulur. Oluşturulan bu elektriksel darbe akson tarafından diğer hücrelere iletilmek üzere aksonun sonunda bulunan ve aksonu diğer hücrelerin dendritlerine bağlayan sinapsislere kadar iletilir. Aksonun bitiminde bulunan ve aksonu diğer hücrelerin dendritlerine bağlayan sinapsisler gelen bu elektriksel sinyalleri kimyasal tepkimeler yoluyla diğer hücrelere iletirler. Bu iletim sırasında elektriksel sinyallerin gücünü değiştirerek sinyalin diğer hücreleri ne oranda etkileyeceğini belirlerler. Böylece hücrenin çıkışları diğer hücreler için uyarıcı veya önleyici girişler olabilir. Özet olarak bir sinir hücresi kendisine bağlantılı olan hücrelerden gelen sinyalleri toplar, bu sinyaller belirli bir seviyenin üzerinde ise tepki olarak bir elektriksel sinyal üretir ve bu sinyali kendisine bağlı diğer hücrelere iletir. Hücreler arasındaki bağlantılara göre bu iletilen sinyal hücrelere farklı seviyede iletilir. Bu iletim her bir hücre için aynı şekilde devam eder.

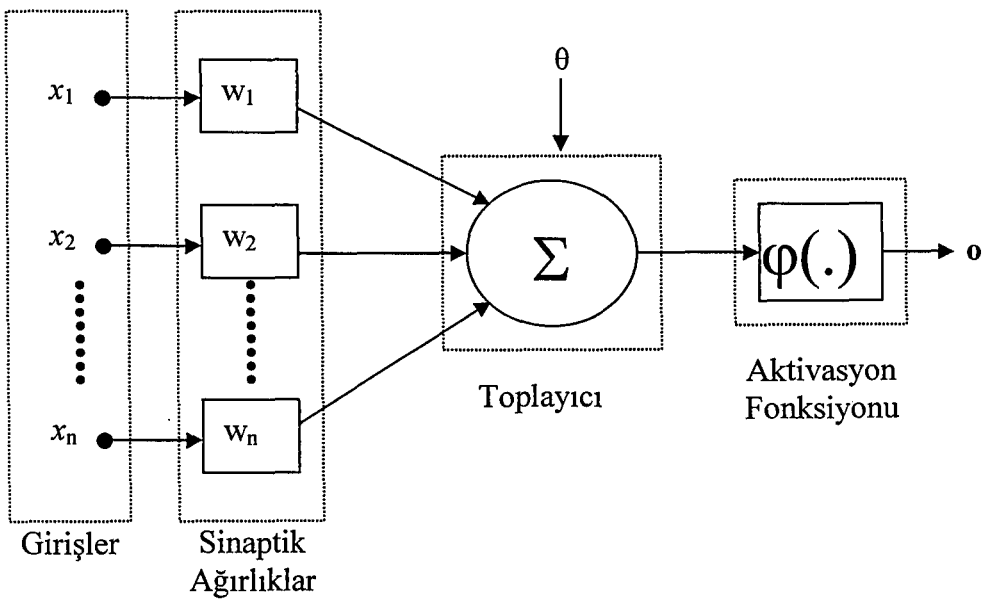
Canlılarda bulunan sinir sistemi, yapısı birbirine benzer yaklaşık 10 milyar sinir hücresinden oluşur ve her bir sinir hücresinin diğer hücrelerle sayısı 100 ila 10000 arasında değişen bağlantısı bulunabilir[12]. Bir sinyalin hücre tarafından iletilmesi milisaniyeler düzeyinde bir zaman alır. Bu zaman elektronik devreler ile karşılaştırıldığında çok uzun bir zamandır. Ancak hücreler arasında kurulmuş olan bağlantıların çok sayıda olması ve hücrelerin paralel bir yapıda çalışması işlemleri çok fazla hızlandırmaktadır.

2.2 Yapay Sinir Ağları

Canlılarda bulunan sinir sisteminin özelliklerinden esinlenilerek sinir hücrelerinin matematiksel bir modeli çıkarılmaya çalışılmıştır. Sinir sisteminin davranışını tam olarak modelleyebilmek için fiziksel bileşenlerin doğru olarak modellenmesi gerekliliği düşüncesinden yola çıkarak sinir sisteminin temel bileşenleri olan sinir hücreleri modellenmiştir[13].

2.2.1 Yapay sinir hücresi modeli

Sinir hücrelerinin, sinir sisteminin temel yapı taşlarını oluşturması gibi yapay sinir hücreleri de yapay sinir ağlarının temelini oluşturmaktadır. Yapay sinir hücrelerinin temel modeli Şekil 2-3' de verildiği gibidir[1].



Şekil 2-3 Yapay sinir hücresi modeli.

Yapay sinir Hücresi yapısında üç yapı bulunmaktadır:

a. Sinapsisler : Kendilerine ait ağırlıkları veya güçleri olan yapılardır. Girişlerden gelen sinyalleri ağırlıkları ölçüsünde etkileyerek toplayıcıya göndeririler.
b. Toplayıcı : Hücrede doğrusal birleştirici olarak görev yapan bölümdür. Girişlerden gelen, sinapsisler tarafından ağırlıklandırılan sinyalleri toplama görevini yapar.
c. Aktivasyon Fonksiyonu : Toplayıcıdan gelen sinyalin genliğini sınırlandırarak hücre çıkışını oluşturur.

Yapay Sinir Hücresi modelinde hücreye gelen sinyaller, girişte sinapsislerde ağırlıklarla çarpılarak, toplayıcıda girişlerin ağırlıklandırımış değerleri toplanır. Elde edilen bu toplam aktivasyon fonksiyonundan geçirilerek hücrenin çıkış sinyalini oluşturur. Hücre girişindeki her bir giriş sinyali ve sinapsislerdeki ağırlıklar hücre çıkışını etkilemektedir. Bu etkinin miktarı giriş sinyalinin gücüne ve sinaptik ağırlıkların miktarına bağlıdır. Hücrenin çıkış sinyalini etkileyen diğer bir faktör de aktivasyon fonksiyonudur. Aktivasyon fonksiyonu hücrenin çıkış sinyalinin değerini belirli bir aralıkta tutmak için kullanılır.

Şekil 2-3' de verilen yapay hücre modeli matematiksel olarak şu şekilde ifade edilebilir:

$$S = w_1 x_1 + w_2 x_2 + ... + w_n x_n - \theta = \sum_{i=1}^{n} w_i x_i - \theta \qquad (2\text{-}1)$$

$$o = \varphi(S)$$

(2-1)'nolu eşitlikte verilen $x_1, x_2 ... x_n$ değerleri yapay sinir hücresine gelen sinyalleri ifade etmektedir. $w_1, w_2 .. w_n$ değerleri sinaptik ağırlıkları ifade eder. Her bir giriş sinyaline karşı gelen sinaptik ağırlıklarla giriş sinyallerinin çarpımının toplamı hücreye giren sinyal toplamını vermektedir. Hücrenin çıkış sinyalinin değeri olan o değeri ise hücreye giren toplam sinyalin aktivasyon fonksiyonundan geçmiş halidir.

2.2.2 Aktivasyon fonksiyonları

Aktivasyon fonksiyonları hücrenin çıkış değerini sınırlandırmak için kullanılan fonksiyonlardır. Aktivasyon fonksiyonu yapay sinir hücresinin davranışını belirleyen önemli etmenlerden biridir. Aktivasyon fonksiyonu kullanılması sinir hücresine doğrusal olmama özelliğini katmaktadır[2].

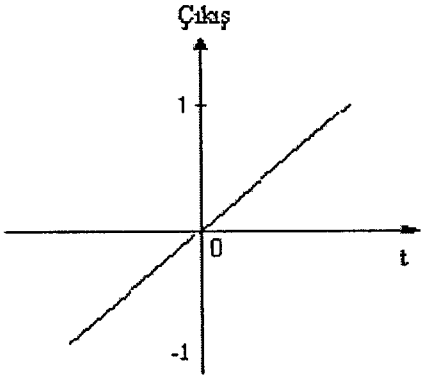
Yapay sinir hücrelerinde kullanılan başlıca aktivasyon fonksiyonları şunlardır:

a. Doğrusal Aktivasyon Fonksiyonu,
b. Sert geçişli Aktivasyon fonksiyonu,
c. Sigmoid tipi aktivasyon fonksiyonu,
d. Hiperbolik tanjant tipi aktivasyon fonksiyonu.

2.2.2.1 Doğrusal aktivasyon fonksiyonu

Doğrusal aktivasyon fonksiyonu sürekli değerli ve giriş değerini doğrusal olarak çıkış değerine çeviren fonksiyondur. Fonksiyonun giriş değeri çıkış değerine eşittir. Doğrusal aktivasyon fonksiyonunun formülü 2-2' de verilmiştir. Fonksiyonun grafiği Şekil 2-4' de verilmiştir.

$$y = \varphi(x) \tag{2-2}$$

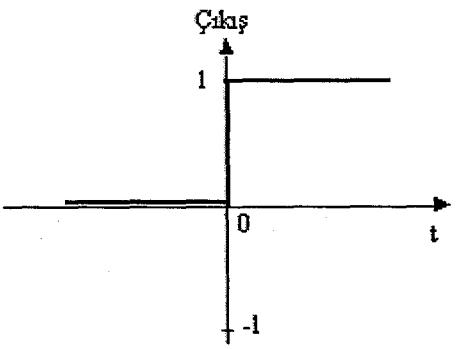


Şekil 2-4 Doğrusal fonksiyon grafiği

2.2.2.2 Sert geçişli aktivasyon fonksiyonu

Sert geçişli aktivasyon fonksiyonu mantıksal çıkış değeri üreten bir fonksiyon olup sınıflandırma amacıyla kullanılabilmektedir. Fonksiyonun girdi değeri sıfırdan küçükse veya sıfıra eşitse sıfır, sıfırdan büyük ise 1 değerini vermektedir. Sert geçişli aktivasyon fonksiyonunun matematiksel formülü 2-3 de verilmiştir. Fonksiyonun grafiği Şekil 2-5' de verildiği gibidir.

$$y = \varphi(x)\begin{cases} 0 & if \quad x \le 0 \\ 1 & if \quad x > 0 \end{cases} \tag{2-3}$$

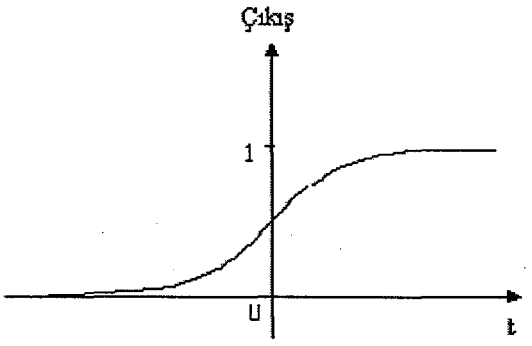


Şekil 2-5 Sert geçişli fonksiyon grafiği

2.2.2.3 Sigmoid tipi aktivasyon fonksiyonu

Sigmoid aktivasyon fonksiyonu türevlenebilir, sürekli ve doğrusal olmayan bir fonksiyondur. Sigmoid fonksiyonu tek yönlü bir fonksiyon olup 0 ile 1 aralığında değer üretir. Sigmoid aktivasyon fonksiyonunun matematiksel formülü 2-4' de verilmiştir. Fonksiyonun grafiği Şekil 2-6' da verildiği gibidir.

$$y = \varphi(x) = \frac{1}{1+e^{-\alpha x}} \tag{2-4}$$

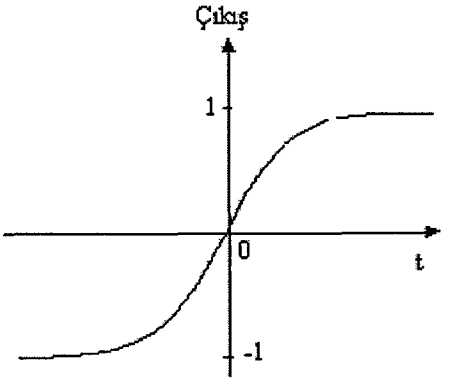


Şekil 2-6 Sigmoid fonksiyon grafiği

2.2.2.4 Hiperbolik tanjant tipi aktivasyon fonksiyonu

Hiperbolik tanjant tipi aktivasyon fonksiyonu sigmoid fonksiyonu ile benzer özelliklere sahip olup çıkış değerler aralığı –1 ile 1 arasındadır. Fonksiyonun matematiksek formülü 2-5' te verilmiştir. Fonksiyonun grafiği ise Şekil 2-7' de verildiği gibidir.

$$y = \varphi(x) = \frac{1 - e^{-\alpha x}}{1 + e^{+\alpha x}} \tag{2-5}$$



Şekil 2-7 Hiperbolik tanjant fonksiyon grafiği

2.2.3. Yapay sinir ağı

Yapay Sinir Ağları, yapay sinir hücrelerinin belirli bir düzenle bir araya getirilmesiyle oluşmuş yapılardır[1,2]. Hücreler genel olarak 3 katman halinde, her katmanda birbirine paralel olacak şekilde bir araya gelirler. Yapay sinir ağındaki her katman bir önceki ve bir sonraki katmanla bağlantılıdır. Her katmanın çıktısı bir sonraki katmanın girdisini oluşturmaktadır.

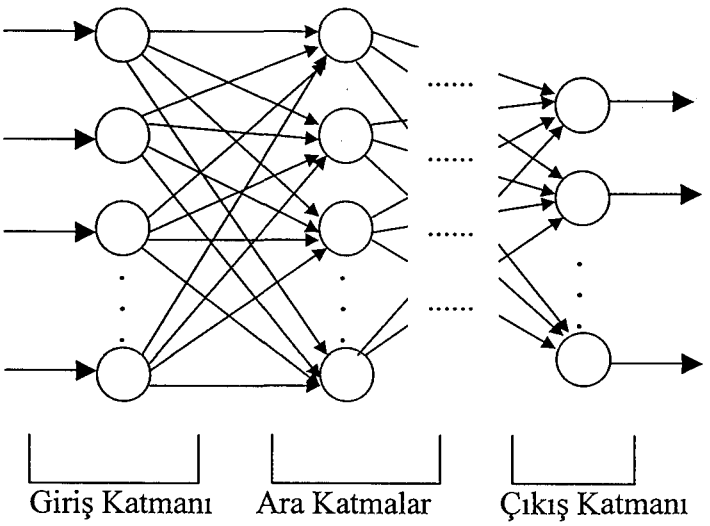
Yapay sinir ağ yapısını oluşturan katmanlar ve görevleri şu şekildedir:

a. Giriş Katmanı: Dış dünyadan gelen girdileri alarak ara katmana gönderir. Bu katmanda herhangi bir bilgi işleme olmaz, gelen bilgi olduğu gibi bir sonraki katmana gönderilir. Bu katmanda bir veya birden fazla süreç elemanı bulunabilir ve her süreç elemanının bir girdisi ve bir çıktısı vardır.

b. Ara Katmanlar: Ara katmanlar gizli katmanlar olarak da adlandırılabilir. Ara katmanlar girdi ve çıktı katmanları arasında yer alır. Bir yapay sinir ağında bir yada birden fazla gizli katman bulunabilir ve her gizli katmanda bir veya birden fazla sinir hücresi bulunabilir. Ara katmanlar giriş katmanından gelen bilgiyi işleyerek bir sonraki katmana gönderir. Ara katmanlarda bulunan sinir hücrelerinin bir önceki katmanda bulunan sinir hücresi kadar giriş ve tek çıkışı vardır. Ara katmanlarda bulunan her bir hücre bir sonraki katmanda bulunan tüm hücrelere bağlıdır.

c. Çıktı Katmanı: Ara katmanlardan gelen bilgileri işleyerek ağın çıkışını oluşturan katmandır. Çıktı katmanında bulunan sinir hücreleri bir önceki katmanda bulunan tüm sinir hücrelerine bağlıdır. Her bir sinir hücresinin bir yada birden fazla girişi ve sadece bir çıkışı vardır.

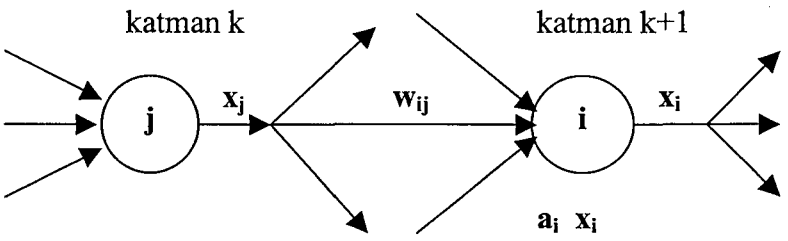
Tipik bir yapay sinir ağı yapısı Şekil 2-8' de gösterilmiştir.



Şekil 2-8 Yapay sinir ağı yapısı

Yapay Sinir ağı, ağı oluşturan yapay sinir hücrelerinden ve bu hücreleri biri birine bağlayan ağırlıklardan oluşmaktadır. Her bir katmanda bulunan hücreler bir önceki ve bir sonraki katmanlarda bulunan hücrelere bağlıdır. Giriş katmanlardaki hücreler sadece bir sonraki katmana, çıkış katmanındaki hücreler ise sadece bir önceki katmana bağlıdır.

Ağ içerisindeki iki hücrenin bağlanması Şekil 2-9' de verilmiştir.



Şekil 2-9 Yapay sinir hücrelerinin bağlanması

j. hücre ile i. hücre arasında kurulan bağlantının ağırlığı w_{ij} şeklinde gösterilmektedir. i. Hücrenin çıkışı x_i şeklinde gösterilmektedir. Hücreye giren bilgilerin ağırlıklı toplamı a_i şeklinde gösterilmektedir.

Bir hücre için çıkış fonksiyonu

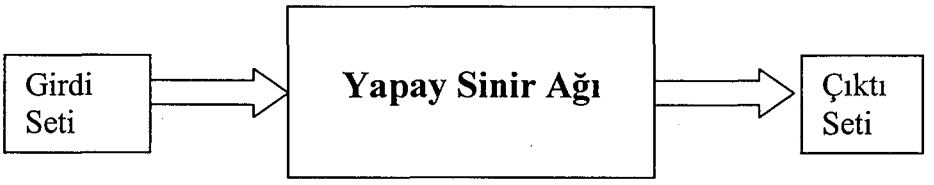
$$net_i^k = \sum_j^N w_{ji} out_j^{k-1} \quad (2\text{-}6)$$

$$out_i^k = f(net_i^k) \quad (2\text{-}7)$$

şeklinde verilir.

2.2.4. Yapay sinir ağı çalışması ve eğitimi

Yapay Sinir Ağlarının genel çalışma prensibi, bir girdi setini alarak onları çıktı setine çevirmek olarak özetlenebilir[2,4]. Yapay sinir Ağı çeşitli öğretme yöntemleri ile eğitilerek belirli girdiler için istenen çıktıları oluşturacak şekle getirilir, ancak yapay sinir ağı girdi setini çıktı setine nasıl çevirdiği konusunda bir bilgi vermez. Bu açıdan bakıldığında yapay sinir ağları kara kutuya benzetilebilir(Şekil 2-10).



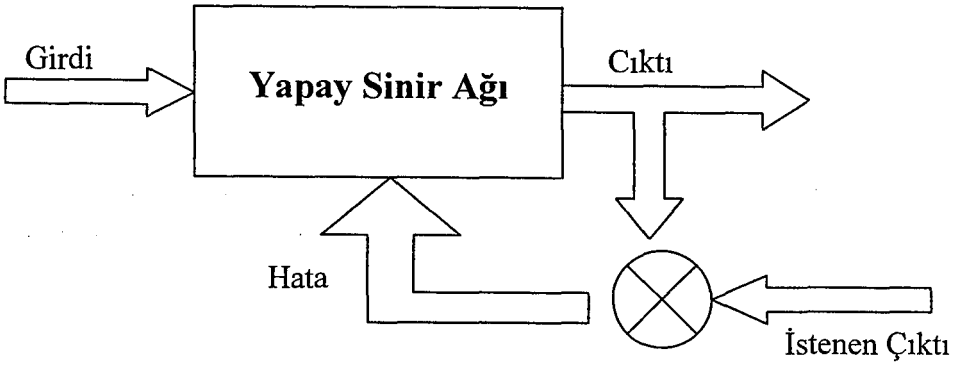
Şekil 2-10 Yapay sinir ağı blok gösterimi

Yapay sinir ağının girdileri çıktılara dönüştürme formülü

$$y = f(x, w) \quad (2\text{-}8)$$

şeklinde verilebilir. Burada y ağın çıkış değeri x ağın giriş değeri ve w ağın hücreleri arasındaki ağırlıklardır.

Yapay Sinir ağı girdiler ile çıktılar arasında kuracağı ilişkiyi eğitim ile öğrenir. Eğitim aşamasında ağın öğrenmesi istenen örnek giriş ve çıkışlar ağa sunulur. Her bir girdi çıktı bileşeninde ağın yaptığı hata miktarına göre, hatayı azaltacak şekilde ağ parametrelerinde değişiklikler yapılır(Şekil 2-11).



Şekil 2-11 Yapay sinir ağı öğreticili eğitimi

Yapay sinir ağında ağın çıkış değeri ağ girişlerinin ve ağ içerisindeki ağırlıkların fonksiyonu olarak ifade edilmektedir. Ağ eğitiminde kullanılan n. giriş vektörü için ağın yaptığı hata miktarı

$$\varepsilon(n) = y(n) - t(o) \tag{2-9}$$

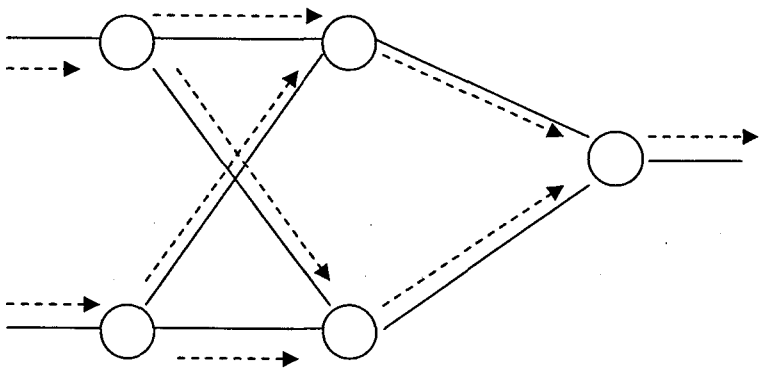
ile verilir. Ağ çıkışında hesaplanan hata miktarının azaltılması için formül parametrelerinde güncelleme yapılmalıdır. Ağın istenen çıkışı üretmesini sağlamak için ağırlık değerlerinde iteratif olarak değişiklik yapılır. Ağırlıkların genel güncelleme formülü

$$\Delta_l w_{ji} = \lambda.h \tag{2-10}$$

şeklinde verilir. Formüldeki λ değeri ağırlık güncellemesinin yönünü h değeri de belirlenen yönde ne kadar büyüklükte bir güncelleme yapılacağını belirtir. Formülde kullanılan λ ve h değerlerinin bulunma biçimi eğitim algoritmasına göre değişmektedir.

2.3. İleri Beslemeli Ağlar

İleri beslemeli Yapay Sinir Ağ yapılarında hücreler katmanlar olarak düzenlenmiştir. Ağ yapısı bir adet giriş katmanı bir adet çıkış katmanı ve bir veya birden fazla gizli ara katmandan oluşur. Her katmanda istenen sayıda sinir hücresi kullanılabilir. Her bir katmandaki tüm hücreler bir üst katmandaki tüm hücrelere bağlıdır. Katmandaki hücrelerin çıkışları bir sonraki katmanda bulunan hücrelere girdi olarak verilir. Tipik bir ileri beslemeli ağ yapısı Şekil 2-12' de gösterilmiştir.



Şekil 2-12 İleri beslemeli ağda ileri veri akışı modeli

İleri beslemeli ağlarda bilgi akışı ileri doğru olup herhangi bir geri dönüş yoktur. Ağa verilen girdiler giriş katmanı, gizli katmanlar, çıkış katmanı şeklinde bir yol takip ederler ve ağ çıkışını oluştururlar. Ağın çıkış değeri, ağın giriş değerinin bir fonksiyonu olarak ifade edilebilir. Ağın belirli bir andaki çıktısını ağın o andaki girdisi belirler.

İleri beslemeli ağlarda, çıkış fonksiyonu x giriş vektörü ve y çıkış vektörü olmak üzere aşağıdaki matematiksel fonksiyon ile ifade edilebilir.

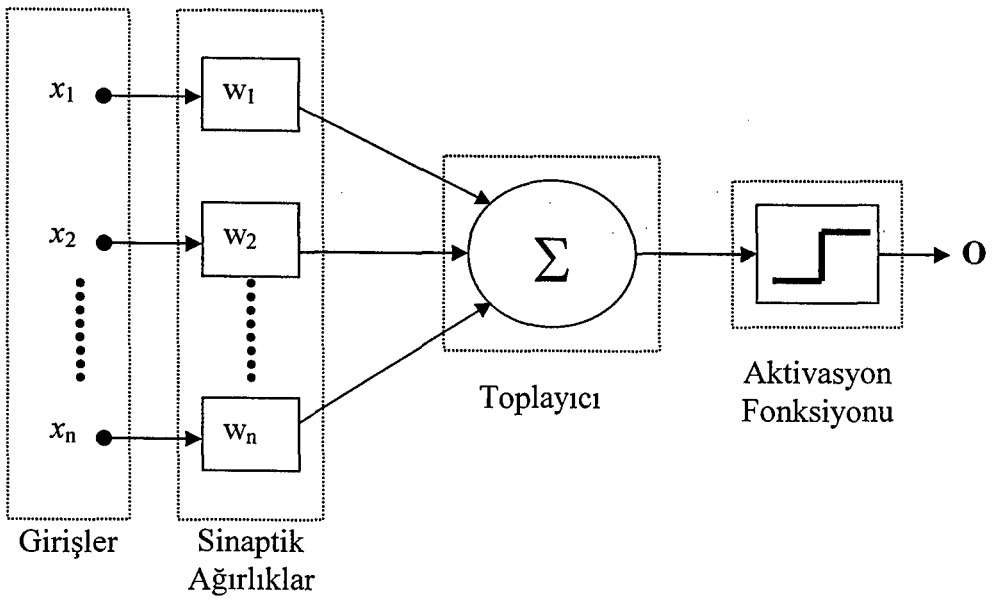
$$y = f(x, w) \tag{2-11}$$

2.3.1. Perseptronlar

Yapay sinir ağlarının en temel biçimi tek katmanlı perseptron olarak adlandırılmaktadır[14-16]. Perseptronun çalışma ilkesi birden fazla sürekli girdiyi alıp bir çıkış üreten doğrusal yapay sinir hücresine dayanır. Ağın çıktısı ikili bir değerden oluşur. Ağın çıktı değerinin hesaplanmasında eşik değer fonksiyonu kullanılır. Ağın eğitilmesi sırasında istenen çıktıların elde edilmesi için ağın ağırlık değerleri ve eşik değer ünitesinin ağırlık değerleri değiştirilir.

Perseptron modelinde kullanılan yapay sinir hücrelerinde aktivasyon fonksiyonu olarak sert geçişli fonksiyonlar kullanılmaktadır. Bu ağ yapısı doğrusal olarak ayrılabilen problemlerin çözümünde kullanılabilmekte, fakat doğrusal olmayan sınıflandırma problemlerinin çözümünde yetersiz kalmaktadır.

Perseptron modeli Şekil 2-13' de verilmiştir.



Şekil 2-13 Perseptron modeli

Perseptron çıkış formülü

$$f(a) = \begin{cases} -1, & a \le 0 \\ +1, & 0 < a \end{cases} \tag{2-12}$$

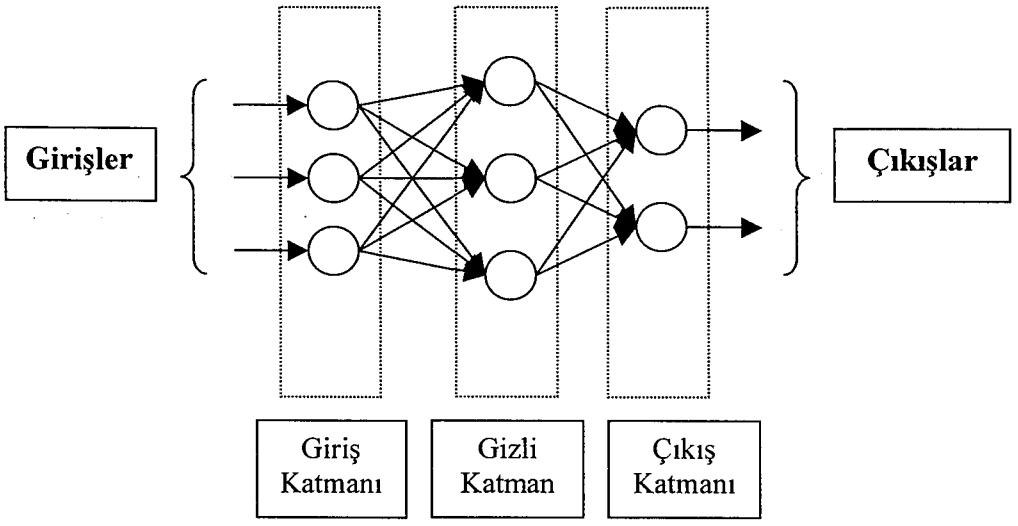
$$a_i = \left(\sum_{j=1}^{N} w_j u_j \right) + \theta \tag{2-13}$$

şeklinde yazılabilir.

2.3.2 Çok katmanlı perseptronlar

Çok katmanlı perseptronların en önemli özellikleri ağı oluşturmada kullanılan aktivasyon fonksiyonları ve kullanılan ara katmanların sayısıdır. Aktivasyon fonksiyonu olarak doğrusal olmayan türevlenir fonksiyonlar ile bir veya birden fazla ara katman kullanılmaktadır(Şekil 2-14).

Çok katmanlı perseptron ağ yapıları doğrusal olmayan problemlerin çözümünde kullanılır[1-2].



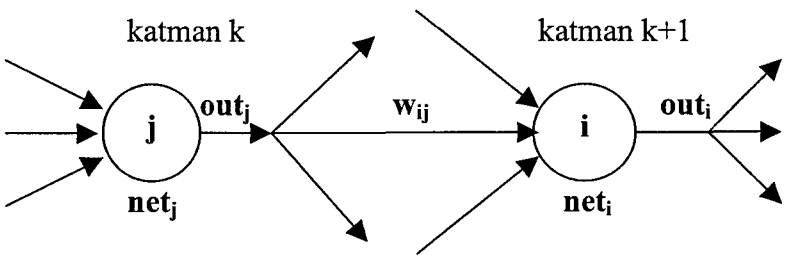
Şekil 2-14 Çok katmanlı perseptron

Çok katmanlı perseptronlarda kullanılan her bir yapay sinir hücresinin yapısı Şekil 2-3' de verildiği gibidir. Bu hücrelerin aktivasyon fonksiyonları doğrusal olmayan sürekli ve türevlenebilir fonksiyonlardır.

Her bir sinir hücresi bir önceki katmandan gelen girdileri bağlantılar arasında bulunan ağırlıklarla çarparak hücreye olan ağırlıklı girdiyi hesaplar(Şekil 2-15). Ağırlıklı girdilerin toplamı hücreye olan toplam girişi verir. Katmanında bulunan hücrelerden k +1 katmanında bulunan i. hücreye giriş, i. hücrenin toplam girdisi

$$net_i^{k+1} = \sum_j^K w_{ij}^{k+1} out_j^k \qquad (2\text{-}14)$$

ile hesaplanır. Bu formülde neti i. hücrenin ağırlıklandırılmış toplam girdisidir. Hücrenin çıkışında hesaplanan değer ise hücrenin ağırlıklandırılmış toplam



Şekil 2-15 Çok katmanlı perseptronda sinir hücrelerinin bağlantıları

ile hesaplanır. Bu formülde neti i. hücrenin ağırlıklandırılmış toplam girdisidir. Hücrenin çıkışında hesaplanan değer ise hücrenin ağırlıklandırılmış toplam girdisinin aktivasyon fonksiyonundan geçirilmiş şeklidir. k+1. katmandaki i. hücrenin çıkış değeri

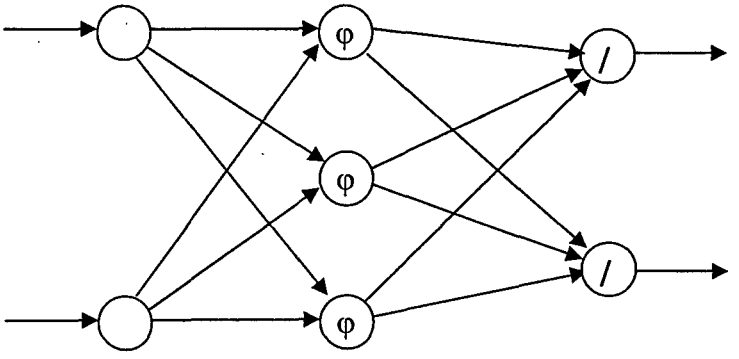
$$\text{out}_i^{k+1} = f(net_i^{k+1}) \tag{2-15}$$

şeklinde verilebilir.

2.3.3 Radyal tabanlı fonksiyon sinir ağları

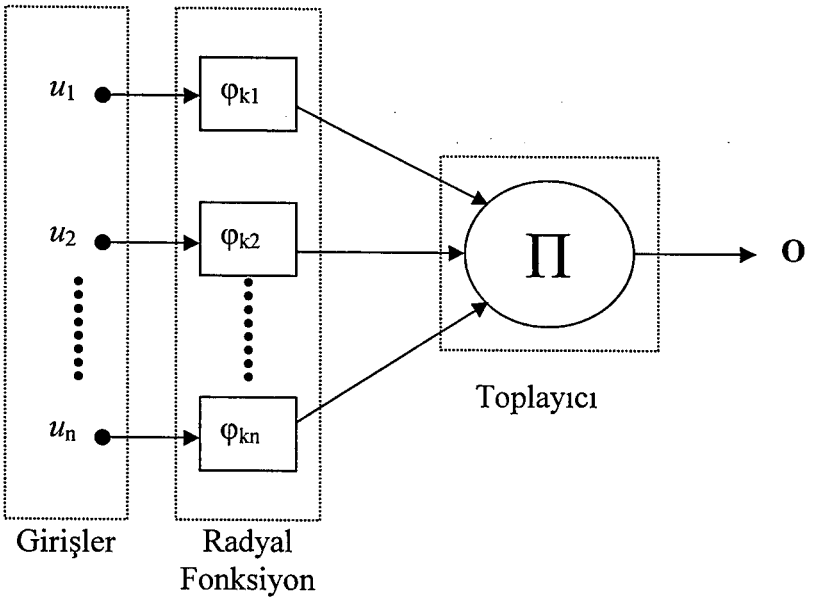
Radyal tabanlı fonksiyon sinir ağları(RTFSA) yapısal olarak çok katmanlı perseptronlara benzerler(Şekil 2-16)[17]. Bu ağlar da benzer şekilde giriş, orta ve çıkış katmanından oluşur ancak, giriş katmanından orta katmana dönüşüm, radyal tabanlı aktivasyon fonksiyonları ile doğrusal olmayan sabit bir dönüşümdür. Orta katmandan çıkış katmanına ise uyarlamalı ve doğrusal bir dönüşüm gerçekleştirilir.

RTFSA' larda giriş katmanından ara katmana olan girdiler ağırlıklandırılmamış girdilerdir.



Şekil 2-16 Radyal tabanlı sinir ağı modeli

Radyal tabanlı fonksiyon sinir ağlarında bir tane ara katman bulunur ve bu ara katmandaki sinir hücreleri özel bir doğrusal olmayan aktivasyon fonksiyonuna sahiptir. Sinir ağında ara katmanlarda bulunan sinir hücrelerinin yapısı Şekil 2-17' de verildiği gibidir.



Şekil 2-17 Radyal tabanlı fonksiyon sinir ağı hücre modeli

Radyal Tabanlı Fonksiyon sinir ağında ara katmanda bulunan bir hücrenin çıkışı:

$$o_k = \prod_i^m \varphi(u, c, \sigma) \tag{2-16}$$

şeklinde verilir. Formülde u giriş değeri, c merkez ve σ varyans değeridir. Örneğin formülde kullanılan radyal tabanlı fonksiyon olarak Gauss merkezcil taban fonksiyonu kullanılabilir:

$$\varphi(u, c, \sigma) = \exp\left[-\left(\frac{u-c}{\sigma}\right)^2\right] \tag{2-17}$$

Ara katmanlara olan girdiler giriş katmanından doğrudan gelen bilgileridir. Ara katmanda bulunan "Radyal Tabanlı Fonksiyonlar"ın merkez ve varyans olmak üzere iki parametresi bulunur. Radyal tabanlı fonksiyonun merkezi boyu süreç elemanına giriş sayısına eşit olan bir merkez vektörüdür. Ağda bulunan her bir eleman için değişik bir merkez değeri vardır. RTFSA'da orta katman

aktivasyon fonksiyonu genellikle standart öklit uzaklıklarını üstsel fonksiyondan geçiren Gaussian fonksiyonudur.

Giriş değerleri merkeze yaklaştıkça fonksiyonun çıkış değeri büyür, merkezden uzaklaştıkça çıkış değeri küçülür.

Çıkış katmanında ara katmandaki her bir sinir hücresinin ağırlıklı etkisi toplanarak ağ çıkışı bulunur. Ağın çıkış formülü şu şekilde verilir:

$$y = \sum_i w_i o_i = \sum_i \left(w_i \prod_j^m \varphi(u, c, \sigma) \right) \tag{2-18}$$

2.4. Yapay Sinir Ağları Öğrenme Algoritmaları

İleri beslemeli yapay sinir ağlarında, ağın eğitilmesi için öğreticili eğitim metotları kullanılmaktadır. Öğreticili eğitim metotlarında ağın öğrenmesi istenen giriş ve çıkış değerleri kullanılmaktadır.

2.4.1. Hata geri yayılım algoritması

Hata geri yayılım algoritması YSA uygulamalarında en yaygın kullanılan algoritmadır. Bu algoritma, hataları çıkış katmanından giriş katmanına doğru azaltmaya çalışan bir algoritmadır[19].

Öğrenme işlemi sırasında ağa hem girdi değerleri hem de bu girdi değerlerine karşılık beklenen çıktı değerleri sunulur. Öğrenme işleminin sonunda ağın her bir girdi değeri için gerekli çıktıyı üretmesi beklenir. Ağın öğrenme işlemi iki safhada ele alınmaktadır.

a. İleri Doğru hesaplama
b. Geriye doğru hesaplama veya hatanın geri yayılımı

2.4.1.1. İleri doğru hesaplama

İleri doğru hesaplama işlemi girdinin ağa sunulması ile başlar. Girdi katmanından gelen bilgiler herhangi bir hesaplama yapılmaksızın ara katmanlara gönderilir.

Ara katmanlardaki elemanlar giriş katmanındaki her bir elemandan gelen bilgiyi bağlantı ağırlıklarının etkisi ile alır. Ara katmandaki bir elemana gelen net girdi şu formül ile verilir:

$$net_i = \sum_j w_{ij} out_j \tag{2-19}$$

Formülde w_{ij} ile verilen değer girdi elemanı ile ara katmanda yer alan eleman arasındaki ağırlık değeridir. Ara katmandaki her bir elemanın çıktısı aktivasyon fonksiyonundan geçirilerek o elemana ait ağırlıklı çıktı değeri hesaplanır.

$$out_i = f(net_i) \tag{2-20}$$

Ara katman çıktılarından elde edilen değerler çıktı katmanındaki elemanlara girdi olarak verilir. Çıkış katmanındaki elemanlara gelen girdiler ağırlıklandırılarak toplanır. Çıkış katmanında bulunan her bir eleman için çıktı değeri ağ çıktısını ifade eder ve şu şekilde ifade edilir.

$$out_i = f(\sum_j w_{ij} out_j) \tag{2-21}$$

Çıkış elemanlarındaki ağırlıklı toplam girdi çıkış aktivasyon fonksiyonundan geçirilerek ağ çıktısı elde edilir.

2.4.1.2. Geriye doğru hesaplama

Ağ girişlerine sunulan bilgileri sonucunda ağ çıkışında elde edilen değerler ağ çıktısı olarak kabul edilir. Ağın çıkışında gözlenen değerler ile beklenen değerler arasındaki fark ağın hatası olarak adlandırılır. Ağın eğitilmesindeki amaç bu hatayı azaltmaktır. Bu nedenle geriye doğru yapılan hesaplama da ağ çıkışında gözlenen bu hata değeri kullanılır ve bu hata ağın ağırlık değerlerine yansıtılır. Eğitim setinde bulunan n. eğitim vektörünün çıktı katmanında gözlenen hatası

$$\varepsilon_n = \frac{1}{2}\sum_j (t_{nj} - out_{nj})^2 \tag{2-22}$$

şeklinde ifade edilir.

Her bir süreç elemanı için oluşan hata miktarının kareleri toplamı ağın toplam hatasını verir. Kareli toplam hatanın matematiksel işaretinden etkilenmesini engellemek için kullanılır.

Tüm eğitim vektörleri için ortalama ağ hatası

$$E = \frac{1}{N}\sum_n^N (\varepsilon_n) \tag{2-23}$$

şeklinde ifade edilir. Bu hata tüm eğitim seti için bulunan ağın ortalama hatasıdır.

Bulunan bu toplam hatayı en aza indirmek için hataya neden olan süreç elemanlarına bu hatanın dağıtılması gerekmektedir. Hatanın azaltılması için ağda bulunan proses elemanları arasındaki ağırlıkların hatayı azaltacak şekilde düzeltilmesi gereklidir. Ağın toplam hatasını azaltmak için yapılan işlem maliyet fonksiyonunu en küçüklemek şeklinde ele alınabilir[1]. Bu işlem sırasında en aza indirilmeye çalışılan fonksiyon hata sinyalleri cinsinden (2-23) eşitliği ile ifade edilir.

2-22 ve 2-23' nolu eşitlikte görüldüğü gibi yaptığı hata miktarı ağın çıkış değerine bağlıdır. En küçüklemeye çalışılan hata fonksiyonunda yer alan ağ çıkış değeri ağ parametreleri cinsinden

$$out_p = f(W^T.x) \tag{2-24}$$

şeklinde ifade edilebilir bu denklemde W^T ağda bulunan tüm ağırlıkların vektör cinsinden ifadesidir. x değeri ise ağın giriş değerlerini ifade eder. 2-24' nolu eşitlikte ağın çıkış değerlerini etkileyen serbest değişkenler ağın ağırlıklarıdır. Maliyet fonksiyonunun değerini etkileyen değişkenler ağın ağırlık değişkenleridir. Ağ parametrelerinde bulunan her bir ağırlık değişkeni maliyet N boyutlu maliyet fonksiyonun hata yüzeyindeki bir boyut olarak düşünülebilir. Hata yüzeyinde hatanın en küçük olduğu dolayısıyla fonksiyonun en küçüklendiği noktaya ağırlık değişkenlerinin uygun değerlerinin bulunmasıyla ulaşılabilir. Hatanın en küçük olduğu noktaya ulaşabilmek için ağırlıklarda yapılacak değişiklikler her bir ağırlığın hata fonksiyonuna göre kısmi türevlerini hesaplanması ve ağırlıkların eğimi azaltacak şekilde değiştirilmesiyle ulaşılabilir. Ağırlıklarda yapılacak değişiklikler

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}} \tag{2-25}$$

şeklinde ifade edilebilir. Hatayı azaltmak için ağırlıklarda yapılacak değişiklikler hata fonksiyonun ağırlıklara göre kısmi türevi ile orantılıdır.

Kısmi türevlerin ağ hatası cinsinden kısmi türevi zincir kuralı uygulanarak

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_{ij}} \frac{\partial net_{ij}}{\partial w_{ij}} \tag{2-26}$$

şeklinde verilebilir burada yer alan net_j değeri 2-19' nolu eşitlik ile verilen değerdir. Bu değer sinir hücresine bir önceki katmandan gelen tek bir hücreye ait çıkış değeridir. Sinir hücresine giren tüm değerleri bulmak için önceki katmanda bulunan sinir hücresi çıkışları toplanırsa eşitlikte

$$\frac{\partial net_{ij}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_l w_{jl} out_l \tag{2-27}$$

$$\frac{\partial net_{ij}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{l \neq i} w_{jl} out_l + w_{ji} out_i \right) = o_i \tag{2-28}$$

şeklinde ifade edilir. Bulunan değerler 2-26' nolu eşitlikte yerine konursa

$$\frac{\partial E}{\partial w_{ij}} = O_i \frac{\partial E}{\partial net_j} \tag{2-29}$$

eşitliği elde edilir.

Geriye yayılacak hata sinyali δ_{pj} olarak tanımlanırsa bu sinyal

$$\delta_j = -\frac{\partial E}{\partial net_j} \tag{2-30}$$

ile ifade edilir.

2-19 ve 2-30 numaralı eşitlikler birleştirilir ise hata değeri

$$-\frac{\partial E}{\partial w_{ji}} = \delta_j o_i \tag{2-31}$$

ile ifade edilir.

2-15 numaralı eşitlikte değerler yerine konursa ağırlıklardaki değişim miktarı

$$\Delta w_{ji} = \eta \delta_j o_i \tag{2-32}$$

şeklinde bulunur.

Burada hata yüzeyinde yer değiştirilecek adım büyüklüğü η sabiti ile ifade edilir. η değeri öğrenme katsayısı olarak nitelendirilmektedir.

δ_j değerinin ağ parametreleri cinsinden ifadesi için

$$\delta_j = -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} \tag{2-33}$$

formülü ile türev zincir kuralı uygulanır. 2-33 numaralı eşitlikte kullanılan ifadelerden net_j nin fonksiyonu olan çıkış değeri yerine konursa

$$o_j = f(net_j) \tag{2-34}$$

$$\frac{\partial o_j}{\partial net_j} = f'(net_j) \tag{2-35}$$

burada kullanılan $f()$ fonksiyonu katmanlarda kullanılan aktivasyon fonksiyonudur.

Maliyet fonksiyonunun çıkış değerine göre kısmi türevi hesaplanırken iki durum göz önüne alınmalıdır.

a. Ara katmanlar için kısmi türev hesaplaması
b. Çıkış katmanı için türev hesaplaması

Çıkış katmanları için ağın yaptığı hata miktarı doğrudan hesaplanabilir.

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j}\left(\frac{1}{2}\sum_j (t_j - o_l)^2\right) = -(t_j - o_j) \tag{2-36}$$

2-35 ve 2-36 numaralı eşitliklerde bulunan değerler 2-33 numaralı eşitlikte yerine konursa çıkış değeri için hata miktarı

$$\delta_j = (t_j - o_j) f'(net_j) \tag{2-37}$$

ile hesaplanır.

Ara katmanlardaki sinir hücrelerinin hata miktarları doğrudan hesaplanamamaktadır. Bu sebeple

$$\frac{\partial E}{\partial o_j} = \sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial o_j} \tag{2-38}$$

eşitliği kullanılarak ara katmanlardaki sinir hücrelerinin çıkışlarının ağ hatasına göre kısmi türevlerin hesaplanması gerekmektedir.

2-38 numaralı eşitlikte, 2-19 numaralı eşitlikte verilen net_j tanımı yerine konursa

$$\frac{\partial net_k}{\partial o_j} = \frac{\partial}{\partial o_j}\left(\sum_l w_{kl} o_{pl}\right) = \frac{\partial}{\partial o_j}\left(\sum_l w_{kl} o_{pl} + w_{kj} o_{pj}\right) = w_{kj} \tag{2-39}$$

eşitliği elde edilir.

$$\frac{\partial E}{\partial o_j} = \sum_k \frac{\partial E}{\partial net_k} w_{kj} \tag{2-40}$$

eşitliği elde edilir. Daha önce bulunan hata değeri

$$\delta_k = p \frac{\partial E}{\partial net_k} \tag{2-41}$$

2-23 numaralı eşitlikte kullanılarak

$$\frac{\partial E}{\partial o_j} = \sum_k \delta_k w_{kj} \tag{2-42}$$

eşitliği elde edilir. Sonuç olarak 2-33, 2-35 ve 2-42 numaralı eşitlikler birleştirilirse hata değeri

$$\delta_j = f'(net_j) \sum_k \delta_k w_{kj} \tag{2-43}$$

şeklinde elde edilir. 2-43 numaralı eşitlikte yer alan *f()* fonksiyonu ara katmanda kullanılan aktivasyon fonksiyonudur.

2-43 numaralı eşitlikte bulunan δ_{pj} değerleri ağ içerisinde bulunan ağırlıklar için gradyan değerleridir.

Hata geri yayılım algoritmasında maliyet fonksiyonun en aza indirmek için ağırlıklarda yapılacak değişiklikler

$$\Delta w_{ji} = \eta \delta_j o_i \tag{2-44}$$

eşitliği ile ifade edilir.

2.4.2. Hata geri yayılım algoritması iyileştirme yöntemleri

Hata geri yayılım algoritmasının öğrenme hızının artırılması için algoritmada bir takım iyileştirmelere gidilebilir[19]. Bu iyileştirmeler ağırlık değişimlerinde momentum katsayısı kullanılması, değişken öğrenme hızı kullanımı ve doğru araması yöntemleridir.

Momentum katsayısı kullanımı ile ağırlık değişimlerinin (0,+1) aralığında reel bir katsayı ile çarpılıp bir sonraki ağırlık değişimine eklenmesidir. Bu sayede ağırlık değişiminin fazla olduğu yönde daha fazla ilerleme sağlanır.

Değişken öğrenme katsayısı kullanımı eğitim süresince adım büyüklüğünün değiştirilmesini sağlar. Eğitim sırasında bir iterasyon sırasında ağın hata miktarı azalmış ise öğrenme katsayısı artırılarak hatanın azaldığı yönde daha büyük adım büyüklüğü ile ilerleme sağlanmaya çalışılır. Eğer ağın hata miktarı

artmış ise öğrenme katsayısı azaltılarak hata yapılan yönde ilerleme miktarı azaltılmaya çalışılır.

Doğru araması yönteminde ise eğimin azaldığı yönde ilerlenecek en uygun adım büyüklüğü tespit edilmeye çalışılır. Bunun için ilerlenecek yönde hatanın en küçük olacağı noktaya doğru araması ile bulunur. Doğru araması yöntemiyle tespit edilen adım büyüklüğü kullanılarak eğimin tersi yönünde ilerlenir.

2.4.3. Konjüge gradyan yöntemler

Hata geri yayılım algoritmasında maliyet fonksiyonun azaltılması için ağırlık değerleri eğimin tersi yönünde değiştirilmektedir. Ağırlıkların bu yönde değiştirilmesi performans fonksiyonun hızla düşmesini sağlar. Performans fonksiyonun en küçük olacağı noktaya doğru arama yapılırken her bir iterasyonda sabit bir adım büyüklüğü kullanılır ve bu adım büyüklüğü ile eğimin tersi yönünde ilerlenir. Adım büyüklüğünün sabit olması ve arama yönünün eğimin tersi yönünde olması her zaman doğrudan en küçük noktaya gidilmesini garanti etmeyebilir.[1,4]

Konjüge gradyan yöntemler ile maliyet fonksiyonun en küçük olduğu noktanın aranması esnasında gidilecek adım büyüklüğü ve gidilecek yön her adımda yeniden belirlenir. Bu yönde ve adım büyüklüğünde en optimum nokta aranır[20].

2-10 numaralı eşitlikte verilen ağırlık güncelleme formülünde belirtilen ağırlıkların değiştirilme yönü ve değiştirilme miktarları her bir adımda hesaplanır ve bu yönde ağırlık güncellemesi yapılır.

Konjüge gradyan yöntemlerde kullanılan ağılık güncelleme formülü

$$\Delta w_k = \alpha_{k-1} d_{k-1} \tag{2-45}$$

şeklinde verilir. Eşitlikte verilen α değeri ağırlık güncellemesindeki adım büyüklüğünü d değeri ise ağırlık değişim yönünü verir.

Konjüge gradyan yöntemlerde ilk iterasyondaki arama doğrultusu eğimin tersi olacak şekilde seçilir.

$$d_0 = -g_0 \tag{2-46}$$

Daha sonraki adımlarda arama doğrultusu

$$d_k = -g_k + \beta_{k-1} d_{k-1} \tag{2-47}$$

olacak şekilde hesaplanır. Yeni arama doğrultusunun belirlenmesinde şimdiki iterasyondaki gradyan bilgisi ve önceki iterasyonda kullanılan arama doğrultusu kullanılır. Ancak arama doğrultusu hesaplanırken skalar bir değer olan β değeri kullanılır. Burada kullanılan β değeri mevcut iterasyon ve bir önceki iterasyonda kullanılan gradyan bilgisinden elde edilen bir değerdir. β değerinin hesaplanma yöntemine göre konjüge gradyan yöntemler biri birinden ayrılırlar.

Ağırlık güncellemesinde kullanılan diğer parametre olan adım büyüklüğü, hesaplanan doğrultuda bir doğru araması yaparak bulunur. Doğru arama algoritmasıyla belirtilen doğrultudaki minimum noktaya ulaşmak için gerekli olan adım büyüklüğü belirlenir.

2.4.3.1. Fletcher –Reeves konjuge gradyan yöntemi

Fletcher-Reeves konjüge gradyan yönteminde β değerinin hesaplanmasında

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \tag{2-48}$$

eşitliği kullanılmaktadır[22]. Eşitlikte şimdiki iterasyonda hesaplanan ağırlıklara ait gradyan değerlerinin bulunduğu vektör g_k ile bir önceki iterasyona ait gradyan vektörü g_{k-1} ile gösterilmektedir.

2.4.3.2. Polak –Ribiere konjuge gradyan yöntemi

Polak-Ribiere konjüge gradyan yönteminde β değerinin hesaplanmasında

$$\beta_k = \frac{g_k^T (g_k - g_{k-1})}{g_{k-1}^T g_{k-1}} \tag{2-49}$$

eşitliği kullanılmaktadır[22]. Eşitlikte şimdiki iterasyonda hesaplanan ağırlıklara ait gradyan değerlerinin bulunduğu vektör g_k ile bir önceki iterasyona ait gradyan vektörü g_{k-1} ile gösterilmektedir.

2.4.3.3 Doğru Arama Algoritması

Konjüge gradyan yöntemlerde ağırlık güncelleme sırasında kullanılan adım büyüklüğü belirlenen doğrultuda yapılan doğru arama algoritması ile bulunur[22]. Doğru arama algoritmasının amacı belirlenen doğrultuda minimum noktaya ulaşabilmek için gidilmesi gereken adım büyüklüğünü belirlemektir.

Doğru arama algoritmasında, adım büyüklüğünün hesaplanması için öncelikle aranan doğrultuda minimum noktayı bulundurduğu düşünülen bir çerçeve aranır. Bu amaçla

$$\varepsilon(\eta_1) \geq \varepsilon(\eta_2) \leq \varepsilon(\eta_3) \tag{2-50}$$

koşulunu sağlayan 3 adım büyüklüğü bulunur. Bu 3 noktaya oturan parabolün en küçük noktası bulunur ve bu nokta yeni adım büyüklüğü olarak belirlenir.

2.4.4. Yaklaşık Newton yöntemler

Yaklaşık Newton yöntemleri maliyet fonksiyonun en küçüklenmesi için sadece birinci dereceden gradyan bilgisini değil aynı zamanda ikinci dereceden gradyan bilgisini kullanan yöntemlerdir[23]. İkinci dereceden gradyan bilgisinin kullanılması fonksiyona ait Hessian matrisinin hesaplanmasını gerektirmektedir. Ancak bu yöntemlerde ikinci dereceden gradyan bilgisi doğrudan hesaplanmaz. İkinci dereceden gradyan bilgisinin hesaplanması amacıyla maliyet fonksiyonuna ait önceki gradyan ve ağırlık değişimleri kullanılır. Bu yöntemle Hessian matrisi yaklaşık olarak hesaplanmaktadır.

Yaklaşık Newton yöntemlerinde arama doğrultusunun belirlenmesi yaklaşık Hessian matrisi kullanılarak yapılır. Adım büyüklüğün belirlenmesinde konjüge gradyan yöntemlerde olduğu gibi doğru arama yöntemleri kullanılır.

Yaklaşık Newton yöntemleri yaklaşık Hessian matrisinin bulunma yöntemine göre birbirinden ayrılmaktadır.

2.4.4.1. BFGS yaklaşık newton yöntemi

BFGS yaklaşık yöntemi Broyden-Fletcher-Goldfarb-Shanno tarafından önerilmiş ikinci dereceden gradyan bilgisine dayanan bir yaklaşım Newton yöntemidir[24]. BFGS yönteminde Hessian matrisinin yaklaşık tersinin bulunması için

$$B_{k+1} = B_k + \frac{\Delta w\, \Delta w^T}{\Delta w^T \Delta g}\left(1 + \frac{\Delta g^T B_k \Delta g}{\Delta w^T \Delta g}\right) - \frac{B_k \Delta g \Delta w^T + \Delta w \Delta g^T B_k}{\Delta g^T \Delta w} \tag{2-51}$$

eşitliği kullanılır. Eşitlikte kullanılan B_{k+1} değeri Hessian matrisinin hesaplanmış yaklaşık tersidir. g ve w değerleri sırası ile gradyan bilgisi ve ağın ağırlıklarıdır.

Yöntemin ilk iterasyonunda hesaplanan Hessian matrisinin yaklaşık tersi birim matris olarak kullanılır.

$$B_0 = I \tag{2-52}$$

2.4.4. Radyal tabanlı fonksiyon sinir ağlarının eğitimi

Çok katmanlı perseptronların eğimi bir optimizasyon problemi olarak ele alınmaktadır. Radyal tabanlı fonksiyon ağı eğitimi ise çok boyutlu uzayda eğri uydurma yaklaşımıdır ve bu nedenle RTFSA' nın eğitimi, çok boyutlu uzayda eğitim verilerine en uygun bir yüzeyi bulma problemine dönüşür[17].

RTFSA'da uyarlanabilecek serbest parametreler, 2-6 numaralı eşitlikte verilen; merkez vektörleri, radyal fonksiyonların genişliği ve çıkış katman ağırlıklarıdır. Çıkış katmanı doğrusal olduğundan ağırlıklar, çok katmanlı perseptron eğitimine benzer şekilde bulunabilir. Merkezler, başlangıç değeri olarak girişler arasından rasgele ve sabit olarak seçilebilmektedir.

3. BİLEŞEN NESNE MODELİ(COM)

Component Object Model (COM) teknolojisi farklı programlama dillerinde veya yazılım geliştirme ortamlarında kullanılabilecek ortak modellerin ve bileşenlerin geliştirilmesine olanak sağlayan bir teknolojidir. COM, nesneye dayalı bileşen geliştirimine izin veren servis ve uygulamaları içeren bir alt yapı önerir ve basit anlamda tekrar kullanılabilir teknolojiler kümesi olarak tanımlanabilir[10,25]

COM teknolojisinin ortaya çıkışındaki temel fikir uygulama geliştiricilere, kendi fonksiyonlarını yazılım bileşenlerinin içine, donanım yapısındaki tümleşik devreler gibi, paketleme imkanı sunmaktır. Bir donanım geliştirilmek istendiğinde gerekli tümleşik devreler temin edilir ancak bunun için temin devrenin yapısıyla veya kaynak maddesi asıl ilgi alanının dışındadır. Donanım üreticileri gibi günümüzde, uygulama geliştiriciler de yazılımı oluşturacak bileşenleri hazır paketler halinde satın alarak, yazılımları oluşturma eğilimine girmişlerdir. Bu durum yazılım geliştirme anlayışını tamamen geliştirip, farklı bir yön vermiştir[25].

Yazılım geliştirmede bileşen kullanım kavramı bir çok kolaylığın yanında problemleri de beraberinde getirmektedir. Bu problemlerden başlıcası değişik platformlarda, farklı programlama dilleri ile geliştirilen bileşenlerin bir araya getirilmesi sorunudur. Farklı yazılım bileşenlerinin bir arada sorunsuz olarak çalışabilmesi bileşenler arasında iletişim kurmasını, bileşenlerin programlama dilinden bağımsız olmasını ve bileşenlerin yeniden kullanılabilir özellikte olmasını gerektirmektedir.

COM teknolojisi, yazılım geliştirmede bileşen kullanma sorunlarına çözüm olarak Microsoft firması tarafından sunulan, tümleşik bileşen tabanlı bir iskelet yapıdır. Microsoft tarafından sunulan bu yapı ActiveX olarak da bilinir[26]. Bu iskelet yapı, bileşenlerin birbiriyle etkileşimini, bilgi alış-verişini sağlayan bir alt yapı kurar ve bu amaçla oluşturulmuş nesnelerin yeniden kullanılabilmesine olanak verir. Yeniden kullanılabilirlik, uygulama geliştiricilerin sistem oluşturmasını kolaylaştıran bir özellik olmasıyla birlikte, bu şekilde oluşturulmuş farklı kuruluşların sistem bileşenlerinin standart bir iskelet yapıyı kullanarak

iletişim kurabilecekleri ortamı da yaratır. Temelde aynı yapıya sahip, COM ortamını kullanabilen nesnelerin iskelet yapıya uygun özelliklerinin fazla esnek yapıda olmaması, sistemlerin bakımını, ayrıca nesnelerin farklı COM iskelet yapısını barındıran ortamlara uyum sağlamasını kolaylaştırır.

COM teknolojisinde, uygulama geliştiricilere, standart bir uygulama geliştirme arayüzü tanımlanmış, bu arayüz sayesinde uygulama geliştiricilerin, COM iskelet yapısını kullanabilen nesnelerini yaratması ve bu nesneleri özel uygulamalar içerisinde kullanarak, farklı uygulamalardaki nesnelerle etkileşebilme yeteneği kazanması amaçlanmıştır. Bu nesnelerin en önemli özelliği Microsoft tarafından tanımlanan standart ikili(binary) yapıya uygunluğudur. Dolayısıyla nesneler ikili yapı bozulmadan özelleştirilebilir, değişebilir yapı kazanmış olur. Bu özellikleri ile COM teknolojisi, programlamaya evrensellik yaklaşımı kazandırır.

COM teknolojisinin getirdiği temel kavramlar şu şekilde sıralanabilir:

a) Uygulamaların nesnelere erişmesini ve onlar üzerinde işlem yapmasını sağlayan ortak bir yol sağlamak,
b) Bir nesnenin kullanımda olup olmadığını anlayacak bir mekanizma sağlamak ve kullanımda değilse nesneyi silmek,
c) Standart hata bildirim mekanizması, hata kodları ve değerleri sağlamak,
d) Uygulamaların nesnelerini karşılıklı olarak kullanabilmelerine olanak tanımak,
e) Nesnelerin tanımlanabilmeleri için bir yol sağlamak ve uygulamaların, bu nesnelerin gerçekleştirimlerini anlayabilmelerine olanak tanımak.

COM teknolojisinde bu temel kavramları sağlamak için bazı kıstaslar göz önüne alınmıştır. Bu kıstaslar:

a) Yeni bir bileşen yaratarak veya var olan bileşenleri kullanarak, Bileşen Nesnelerini oluşturma ve bunların yeniden kullanılabilirliğini sağlama
b) İkil bir alt yapı sağlayarak, etkileşimi sağlama
c) Etkili bir Nesne Modeli yaklaşımı kullanma
d) Dağılım yetenekleri gösterebilmedir.

3.1 COM Nesneleri

COM, nesne tabanlı programlama modeli olarak tasarlanmıştır. Oluşturulan nesnelere bazı özellikler kazandırılarak, yazılımların etkileşimi sağlanmıştır. Böylece, geliştirildiği programlama dili, çalıştığı işletim sistemi her ne olursa olsun, iki veya daha fazla bileşenin kolaylıkla iletişim kurup birlikte çalışmalarına olanak veren standart bir mimari yaratılmıştır. Bu tür bir özelliği desteklemek amacıyla COM teknolojisinde yazılım nesnelerinin birbirleriyle bağlantı kurmasını sağlayan bir mekanizma tanımlanıp, gerçekleştirilmiştir. Buna göre bir yazılım nesnesi, yapısı içinde, genel fonksiyonları ve bu fonksiyonların paylaştığı standart durumları barındırır. COM bileşenleri standartlaşmış belirli bir arabirimi ve metodolojileri destekleyen nesneler ile veri iletimini gerçekleştirirler.

Her bir COM nesnesi şu temel bileşenlerden oluşur.

a) Sınıflar: Sınıf belirli bir amaç için bir arada bulunan veriler ve yöntemler kümesidir. Her bir COM nesnesi içerisinde bir veya daha fazla sınıfı bulundurabilir.

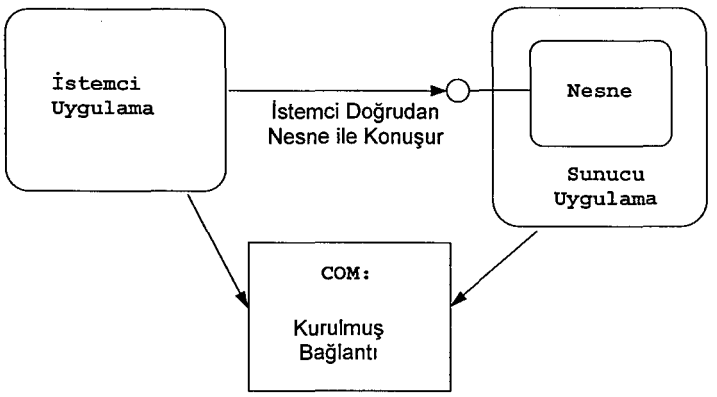
b) Nesne: Nesneler çalışma esnasında yaratılırlar ve birer sınıf olgusudur. Yapabileceği işlevleri(yöntemleri) ve bu işlevler için bulunduğu durum(veri üyesi) bilgisini içeren yazılım parçasıdır. COM nesnesinde herhangi bir anda birden fazla nesne bulunabilir ancak bu nesnelere dışarıdan doğrudan erişime izin verilmez.

c) Arayüzler: Arayüzler bir sınıfın parçası olan işlevler bütünüdür ve sınıfın dış ortam ile olan bağlantısını tanımlarlar. Bir COM nesnesinin birden fazla arayüzü bulunabilir ve nesnenin nasıl kullanılacağını ve davranışının nasıl olacağını belirler.

d) Evrensel Biricik Tanımlayıcı (Global Unique Identifier): Tanımlayıcılar COM nesnelerini ayırt etmek için kullanan 128 bitlik sayılardır. Her bir COM nesnesin kendisine ait ve o nesneyi tanımlayan bir tanımlama numarası vardır.

COM, bilindik yapıdaki sunucu modeline göre istemcilere bazı işlemler yapma olanağı sunar. COM' dan hizmet alabilecek şekilde yapılandırılmış bir istemci, birçok servis sağlayıcıyla değişik şekillerde bağlantı kurabilme yetisini de kazanmış olur. Ancak COM' un önemli bir farkı, bağlantı kurulduğu andan itibaren, yazılım ile nesneler arası bağlantının kurulabilmesidir. Başka bir deyişle

COM, istemci yazılım ile sunucu nesne arasındaki bağlantıyı herhangi bir merkezi Uygulama Geliştirme Arayüzünün zorunlu kıldığı kodların işletim yükü olmaksızın dolaysız bir biçimde kurabilecek şekilde tasarlanmıştır(Şekil 3-1).

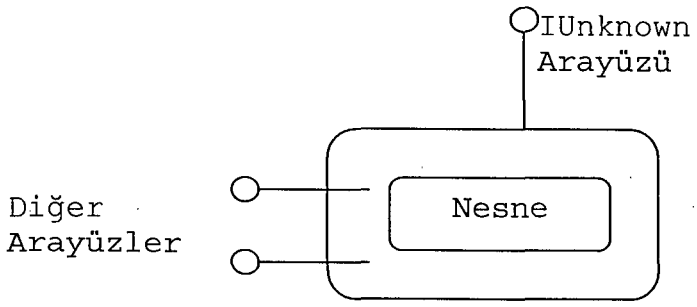


Şekil 3-1 İstemci nesne iletişimi COM aracılığı ile, istemci uygulama ile nesne arasında bağlantı bir kere kurulduktan sonra ilave yük getirmeksizin doğrudan iletişim kurulur

3.2 COM Arayüzleri

COM yapısında, arayüz nesne ile istemci arasında bir anlaşma noktasıdır. Nesnenin verebileceği servisleri ve işlevleri tanımlar. İstemci ile nesne arasında yerleştirilmiş olan ve istemcinin taleplerini nesneye iletip cevap alan bir veri yapılarından oluşur. Pratikte arayüz tanımlamaları nesnenin bileşenlerine erişmeyi sağlayan nesnenin üye işlevleridir. COM nesnesini kullanmak isteyen bir uygulama yazılımı nesneye tanımlı arayüzler yardımı ile erişir. Nesnenin arayüz tanımlamaları dış ortama nesnenin neler yapabileceğini ve hangi özellikleri olduğunu bildirir. Nesneyi kullanacak uygulama nesne içerisinde olup bitenle ilgilenmeyip sadece arayüzler aracılığı ile nesneden hizmet alır.

COM yapısında, tüm bileşen nesneleri IUnknown adı verilen temel bir arabirimi desteklemek zorundadır. IUnknown dışındaki ara birimleri de destekleyen bir nesne, IUnknown' da tanımlı yöntemleri de gerçekleştirmiş olmalıdır. COM nesnesine ait arabirim gösterimi Şekil 3-2' de verilmiştir.



Şekil 3-2 COM nesnesi arayüzleri

Tüm COM nesneleri tarafından tanımlanması gereken IUnknown nesnesinin yapısı Şekil 3-2' de verilmiştir.

IUnknown arayüzü tanımında bulunan yöntemler istemci uygulamanın nesnenin diğer arayüzlerini sorgulamasına izin verecek yapıdadır. IUnknown arayüzünde bulunan QueryInterface metodu sayesinde istemi uygulama nesnenin istenen arayüzlere sahip olup olmadığını sorgulayabilmektedir. Nesneye eklenecek diğer arayüzler için belirli kurallara uyulması gerekmektedir. Eklenecek arayüzler kalıtım yoluyla IUnknown arayüzünden türetilmeli, arayüz tanımlamaları "I" ön eki ile başlamalıdır. Örnek bir nesneye "Face1" ve "Face2" adında 2 arayüzün eklenmesi Şekil 3-3' de verilmiştir.

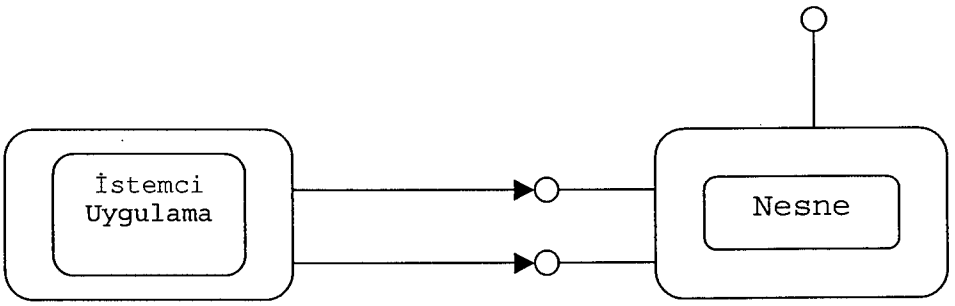
```
interface IUnknown
    {
    virtual HRESULT   QueryInterface(IID& iid, void** ppv
    virtual ULONG     AddRef(void) =0;
    virtual ULONG     Release(void) =0;
    };
```

Şekil 3-3 COM Nesnesi IUnknown arayüzü tanımlaması

```
//example.idl
//
import "mydefs.h","unknwn.idl";
[
object,
uuid(a03d1420-b1ec-11d0-8c3a-00c04fc31d2f),
] interface IFace1 : IUnknown
{
HRESULT MethodA([in] short Bread,
                [out] BKFST * pBToast);
HRESULT MethodB([in, out] BKFST * pBPoptart);
};

[
object,
uuid(a03d1421-b1ec-11d0-8c3a-00c04fc31d2f),
pointer_default(unique)
] interface IFace2 : IUnknown
{HRESULT MethodC([in] long Max,
                 [in, max_is(Max)] BkfstStuff[ ],
                 [out] long * pSize,
                     [out, size_is( , *pSize)] BKFST **
ppBKFST);
}; //end IFace2 def
//end example.idl
```

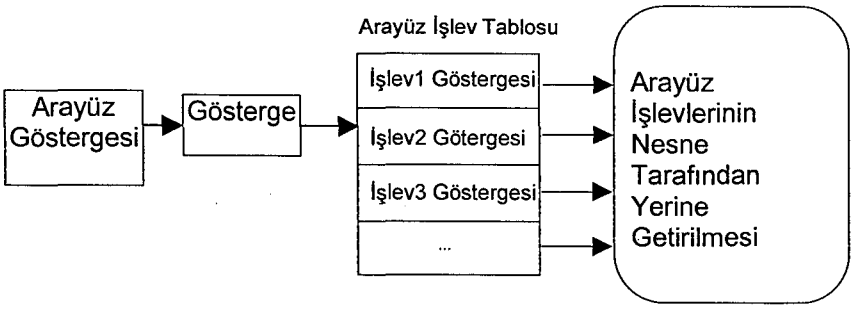
Şekil 3-4 COM nesnesi arayüzü tanımlamaları

Nesneyi kullanmak isteyen bir uygulama nesnenin arayüzüne ait bir gösterge bulundurmak zorunda ve nesne arabirimine bu gösterge aracılığı ile erişmek zorundadır. Şekil 3-5' de uygulamaların bir nesneye ulaşma biçimi gösterilmiştir. İstemci uygulama tarafından nesneye ulaşmak için kullanılan gösterge aslında nesnenin arayüzünde bulunan üye işlevlere göstergelerin bulunduğu bir gösterge dizisi şeklindedir.



Şekil 3-5 İstemci uygulama COM nesnesi haberleşmesi

İşlevlere olan göstergelerin tutulduğu bu tablo yapısı "İşlev Gösterge Tablosu" olarak adlandırılır ve istemci uygulamanın tek bir gösterge ile nesne ara yüzünde bulunan tüm işlevlere erişmesini sağlar.



Şekil 3-6 COM nesnesi arayüz işlev tablosu

Arayüzlerin genel özellikleri şunlardır:

- Arayüzler zamanla değişmezler. Yeni sürümleri olamaz. Eğer yeni bir yönteme eklenecekse, bu arabirim tamamen yeni bir arabirimdir ve yeni bir biricik tanımlayıcı ile tanımlanmalıdır.
- Bir istemci bir nesneye erişimi varsa aslında bu erişim o nesnenin yöntemlerine erişebileceği ve nesnenin desteklediği arayüzlerden birine göstergeden başka bir şey değildir.
- Bir nesne birden fazla arabirimi destekleyebilir. Yani bir nesne birçok arabirimde tanımlanan yöntemi gerçekleştirmiş olabilir. Nesnenin hangi arabirimi desteklediği sorgusu yapılabilmelidir.

Bir nesneye birden fazla arayüz göstergesiyle ulaşmak mümkün olduğunda birden fazla arayüzün desteklenmesi de önemli bir sorunu ortaya çıkarır. Bir istemci bir nesneye ilk olarak erişimi sağladığında elinde sadece bir arayüze gösterge vardır. İstemcinin, nesnenin desteklediği diğer arayüzlere ulaşabilmesi QueryInterface yöntemi ile olur. Bu yöntem tüm arayüzlerde kalıtım yoluyla bulunur. QueryInterface yöntemi ile istemci, nesneye hangi hizmetleri sağlayabileceği sorgusunu iletebilir. İstemci, QueryInterface yöntemine parametre olarak istediği servis temsil eden ara yüzün biricik tanımlayıcı değerini gönderir. İstemci, nesneye erişimi sağladığında, elinde en azından nesnenin desteklemek zorunda olduğu temel arayüz IUnknown' a ait bir gösterge bulunur. Bu gösterge ile, nesnenin kullanım sürecini denetleyebilir. Bu göstergeyi kullanarak yapmak istediği işlemle ilgili ara yüzün desteklenip desteklenmediğini QueryInterface ile sorgulayabilir.

4. BİLEŞEN GERÇEKLEMESİ

Bu çalışmada gerçeklenmesi amaçlanan yapay sinir ağı bileşeninde Bölüm 2' de incelenen yapay sinir ağı modelleri ve eğitim yöntemleri kullanılmıştır. Bileşenin gerçeklenmesinde Microsoft tarafından geliştirilmiş, Bölüm 3'de ele alınan Component Object Model teknolojisi kullanılmıştır[10]. COM nesnesinin oluşturulması için yapay sinir ağı sınıfları ve ara yüzleri tasarlanmıştır.

Bileşeninin oluşturulmasında, Windows 9x/NT/2000 işletim sistemlerinde çalıştırılabilecek yazılım geliştirmeye imkan sağlayan bir görsel programlama aracı olan Microsoft Visual C++ 6.0 kullanılmıştır[27].

4.1 Yapay Sinir Hücresi ve Ağının Modellemesi

Bileşen gerçekleştirirken nesneye yönelik programlama teknikleri kullanılmıştır. Bu amaçla yapay sinir ağına ait sınıflar oluşturulmuştur. Sinir ağında kullanılan iki temel bileşen olan sinir hücresi ve sinir ağı bileşeni birer sınıf olarak tasarlanmıştır.

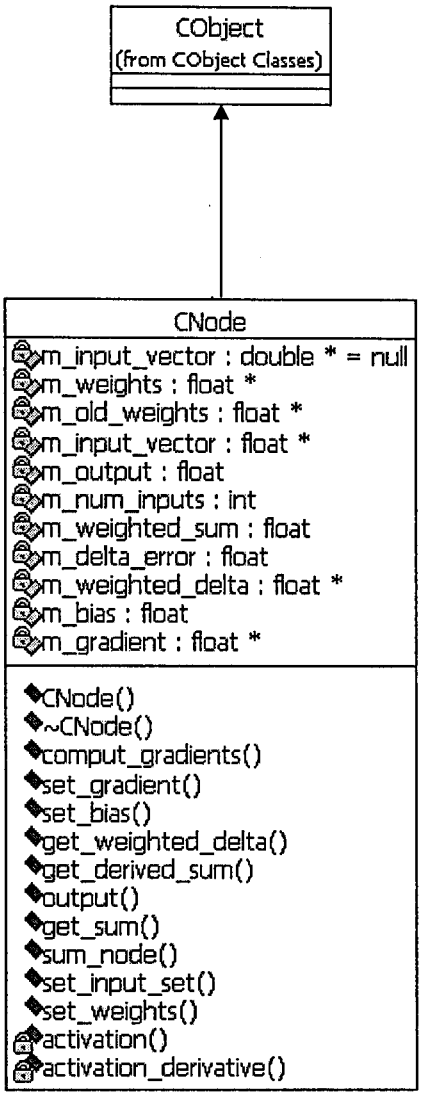
4.1.1. Yapay sinir hücresi modellemesi

Yapay sinir hücresi yapay sinir ağı içerisinde işlemelerin en alt düzeyde yapıldığı bileşendir. Bir yapay sinir hücresi kendisine gelen girişleri işleyerek bunları çıkışlara dönüştürmektedir. Bu işlemleri yaparken hücrenin ağın bütününden haberdar olması gerekmemektedir. Bu bağlamda yapay sinir hücresi sadece girişleri alıp ağırlıklandırarak çıkışlara dönüştürmektedir. Yapay sinir hücresinin yapısal biçimi bölüm 2.2.1' de anlatılmaktadır.

Yapay sinir hücresi gerçeklenirken hücre bir sınıf olarak tasarlanmıştır. Yapay sinir hücresi MFC' de tanımlanmış olan CObject sınıfından türetilmiş bir sınıftır. CObject sınıfı MFC' de tanımlanan en temel sınıf olup tanımlı diğer sınıflar CObject sınıfını temel olarak almaktadır[28].

Yapay sinir hücresinin UML ile gösterimi Şekil 4-1' de verilmiştir.

Yapay sinir hücresi sınıfı hücreye ait giriş değerlerini ve ağırlıklarını tutmakta ve hücreye ait bir çıkış oluşturmaktadır. Hücre girişleri ağırlıklarla çarpılıp hücrenin toplam girişini oluşturmakta aktivasyon fonksiyonundan geçirilen toplam giriş hücre çıkışı olarak bulunmaktadır.



Şekil 4-1 CNode nesnesi UML gösterimi

4.1.1.1. Yapay sinir hücresi sınıfı üye değişkenleri

Yapay Sinir hücresi sınıfına ait üye değişkenleri şunlardır:

m_weights

Sinir ağı hücresine ait ağırlık değerlerinin tutulduğu dizi değişkenidir. Sinir hücresine gelen girdiler, ağırlık vektöründe saklanan ağırlık değerleri ile çarpılıp toplam girdi elde edilir.

m_old_weights

Sinir hücresine ait ağırlıkların bir önceki adımda aldığı değerlerin tutulduğu dizi değişkenidir.

m_input_vector

Sinir hücresine gelen girişlerin tutulduğu dizi değişkenidir.

m_output

Sinir hücresinin çıkışının tutulduğu değişkendir.

m_num_inputs

Sinir hücresine olan girişlerin sayısının tutulduğu değişkendir.

m_weighted_sum

Sinir hücresine gelen girişlerin ağırlıklandırılmış değerlerinin tutulduğu dizi değişkenidir.

m_delta_error

Hata geri yayılım algoritmasında kullanılan geri yayılacak hata değerinin tutulduğu değişkendir.

m_weighted_delta

Hata geri yayılım algoritmasında kullanılan bir önceki katmandaki hücrelere geri yayılacak olan ağırlıklandırılmış hata değerinin tutulduğu dizi değişkenidir.

m_bias

Sinir hücresine gelen beslemenin ağırlık değerinin tutulduğu değişkendir.

m_gradient

Sinir hücresine ait gradyan değerinin tutulduğu değişkendir.

4.1.1.2. Yapay sinir hücresi sınıfı üye işlevleri

CNode(int num_inputs=1)

Sinir hücresi sınıfına ait kurucu fonksiyondur.

virtual ~CNode()

Sinir hücresine ait yok edici fonksiyondur.

void compute_gradients(float delta)

Sinir hücresine ait gradyan değerini hesaplar.

void set_gradient(float *)

Sinir hücresine ait gradyan değerine atama yapmak için kullanılmaktadır.

void set_bias(float bias)

Sinir hücresine ait besleme değerine ait ağırlık değerine ilk değerini atamak için kullanılmaktadır.

float * get_weighted_delta(float delta)

Hata geri besleme algoritmasında kullanılan hata değerlerinin ağırlıklandırılmış biçiminin hesaplanması için kullanılmaktadır.

float get_derived_sum(int type) const

Aktivasyon fonksiyonun türevinden geçirilmiş çıkış değerini oluşturur.

void set_input_set(float * input)

Sinir hücresine ait giriş değerlerinin alınması amacıyla kullanılmaktadır.

bool set_weights(float * weights)

Sinir hücresine ait ağırlıkların yüklenmesi amacıyla kullanılmaktadır.

float output(int type=1)

Sinir hücresinin çıkış değerini oluşturur.

float get_sum(int type) const

Sinir hücresinin aktivasyon fonksiyonundan geçirilmiş çıkış değerini oluşturur.

void sum_node()

Sinir hücresinin ağırlıklandırılmış giriş değerini hesaplar.

float activation(float x,int type=1,float slope =1) const

Sinir hücresinin ağırlıklandırılmış toplam girişini aktivasyon fonksiyonundan geçirilmesi için kullanılmaktadır.

float activation_derivative(float x,int type=1) const

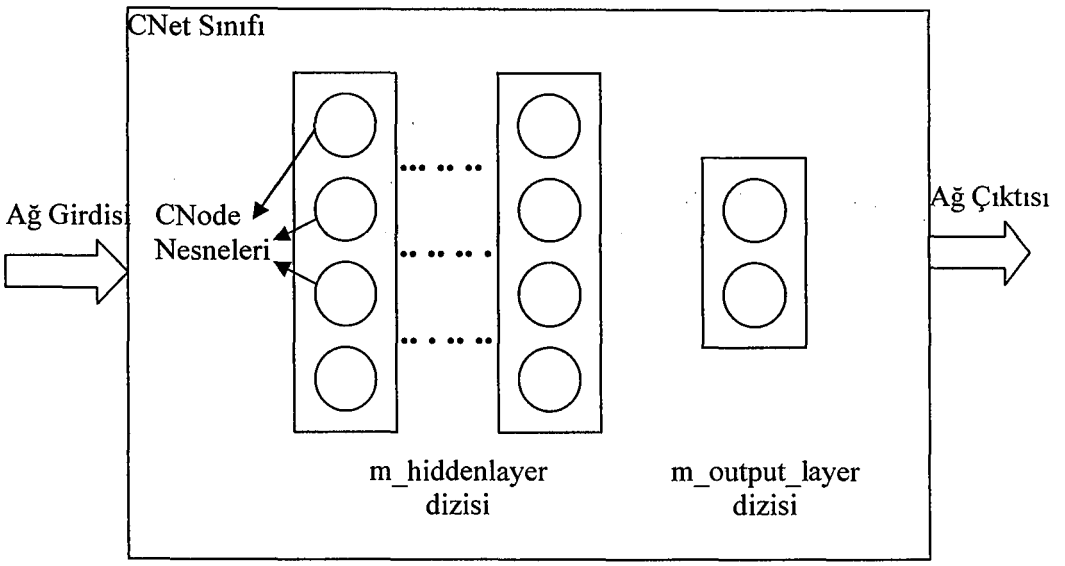
Hata geri yayılım algoritmasında kullanılmak üzere çıkış değerini aktivasyon fonksiyonunun türevinden geçirir.

4.1.2. Yapay sinir ağı modellemesi

Yapay sinir ağı sınıfı yapay sinir ağı topolojisinin tutulduğu ve ağın eğitiminin gerçekleştirildiği sınıftır. Yapay sinir ağı sınıfı CObject sınıfından türetilmiş bir sınıftır. Yapay sinir ağı sınıfında yapay sinir hücresi sınıfından elemanlar bulunur. Hücreler arasındaki bağlantılar, ağ topolojisine uygun olacak şekilde sınıf içerisinde tutulmaktadır.

Yapay sinir ağına ait her bir katmana ait hücreler nesne dizileri halinde sinir ağı sınıfında tutulur. Sinir ağına ait tüm üst seviye işlemler bu sınıf içerisinde gerçekleştirilmektedir.

Yapay sinir ağı sınıfı içerisinde katmanların ve sinir hücrelerinin yerleşimi Sekil 4-2' de verilmiştir.



Şekil 4-2 Yapay sinir ağı sınıfı yapısı

4.1.2.1 Yapay sinir ağı sınıfı üye değişkenleri

Yapay sinir ağı sınıfının UML ile gösterimi Şekil 4-3' de verilmiştir. Yapay sinir ağı sınıfına ait üye değişkenleri şu şekildedir:

int m_num_hidden_layer;

Sinir ağında gizli katman sayısının tutulduğu değişkendir.

int m_num_input;

Sinir ağına ait giriş sayısının tutulduğu değişkendir.

int m_num_output;

Sinir ağına ait çıkış sayısının tutulduğu değişkendir.

int *m_num_hidden_layer_nodes;

sinir ağının gizli katmanlarında bulunan sinir hücresi sayılarının tutulduğu dizi tipinde değişkendir.

float *m_network_weights;

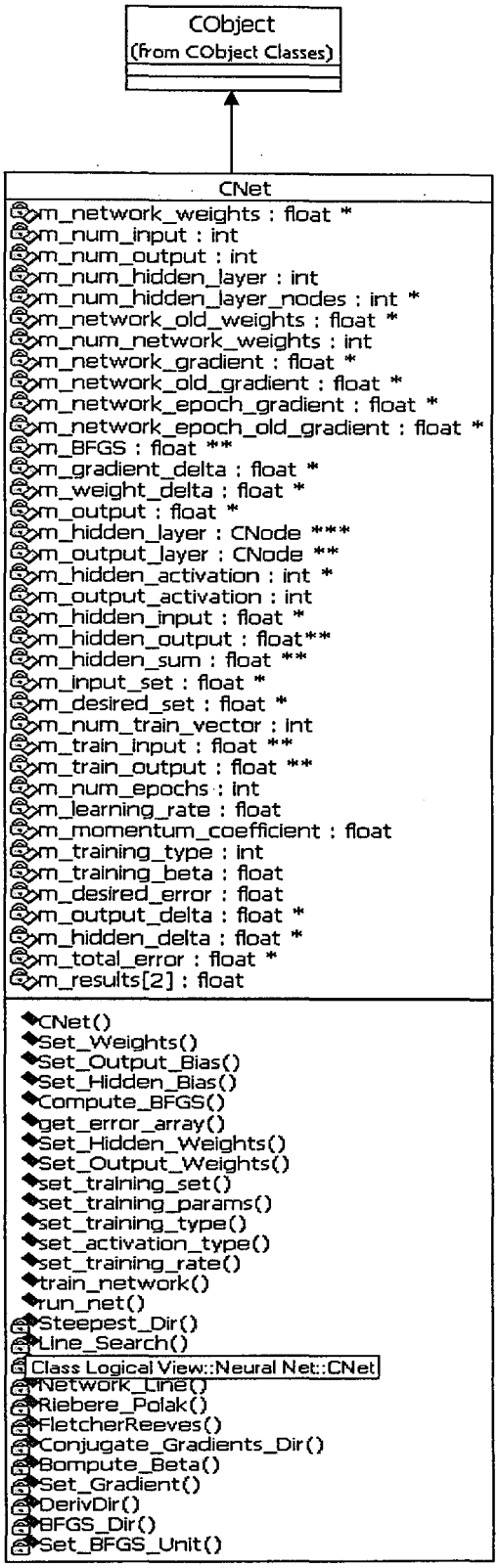
Sinir ağına ait ağırlık değerlerinin tutulduğu dizi değişkenidir.

float *m_network_old_weights;

Sinir ağına ait, eğitim ağırlık değerlerinin, eğitim esnasında bir önceki adımda aldığı değerlerin tutulduğu dizi değişkenidir.

int m_num_network_weights;

Sinir ağında yer alan ağırlık sayısının tutulduğu değişkendir.



Şekil 4-3 CNet nesnesi UML gösterimi

float * m_network_gradient;

Sinir ağında yer alan hücrelerin yerel gradyan değerlerinin tutulduğu dizi tipi değişkendir.

float * m_network_old_gradient;

Sinir ağında yer alan hücrelerin yerel gradyan değerlerinin, eğitim esnasında bir önceki adımda aldığı değerlerin tutulduğu dizi tipi değişkendir.

float * m_network_epoch_gradient;

Sinir ağında yer alan hücrelerin anlık yerel gradyan değerlerinin tutulduğu dizi tipi değişkendir.

float * m_network_epoch_old_gradient;

Sinir ağında yer alan hücrelerin anlık yerel gradyan değerlerinin tutulduğu dizi tipi değişkendir.

float ** m_BFGS;

BFGS eğitim algoritmasında kullanılan Hessian matrisinin yaklaşık tersinin tutulduğu iki boyutlu dizi tipi değişkendir.

float * m_gradient_delta;

Sinir ağının eğitimi esnasında eğitim adımları arasındaki gradyan değişiminin tutulduğu dizi tipinde değişkendir.

float * m_weight_delta;

Sinir ağının eğitimi esnasında eğitim adımları arasındaki ağırlık değişiminin tutulduğu dizi tipi değişkendir.

float * m_output;

Sinir ağına ait çıkış değerlerinin tutulduğu değişkendir.

CNode * m_hidden_layer;**

Sinir ağına ait gizli katmanlardaki sinir hücrelerinin tutulduğu iki boyutlu dizi tipinde değişkendir. Dizinin her bir elemanı bir yapay sinir hücresidir.

CNode ** m_output_layer;

Sinir ağına ait giriş katmanındaki sinir hücrelerinin tutulduğu dizi tipindeki değişkendir.

int *m_hidden_activation;

Sinir ağının gizli katmanlarının aktivasyon fonksiyonu tiplerinin tutulduğu dizi tipi değişkendir.

int m_output_activation;

Sinir hücresinin çıkış katmanına ait aktivasyon fonksiyonunun tipinin tutulduğu değişkendir.

float * m_hidden_input;

Sinir hücresinde giriş katmanından gizli katmanlara olan girişlerin değerlerinin saklandığı dizi tipi değişkendir.

float ** m_hidden_output;

Sinir hücresinde gizli katmanların çıkış değerlerinin tutulduğu iki boyutlu dizi tipi değişkendir.

float * m_input_set;

Sinir hücresinin eğitiminde kullanılan eğitim setine ait giriş değerlerinin tutulduğu değişkendir.

float * m_desired_set;

Sinir hücresinin eğitiminde kullanılan eğitim setine ait çıkış değerlerinin tutulduğu değişkendir.

int m_num_train_vector;

Sinir ağının eğitiminde kullanılacak olan eğitim setinde bulunan eğitim çifti sayısının tutulduğu değişkendir.

float **m_train_input;

Sinir hücresinin eğitiminde kullanılan eğitim setinin giriş değerlerinin tutulduğu değişkendir.

float **m_train_output;

Sinir hücresi eğitiminde kullanılan eğitim setinin çıkış değerlerinin tutulduğu değişkendir.

int m_num_epochs;

Sinir ağının eğitimi esnasında kullanılacak en çok adım sayısının tutulduğu değişkendir.

float m_learning_rate;

Sinir ağının eğitimi esnasında kullanılacak eğitim oranı sabitinin tutulduğu değişkendir.

float m_momentum_coefficient;

Sinir ağının eğitimi esnasında kullanılacak momentum sabitinin tutulduğu değişkendir.

int m_training_type;

Sinir ağının eğitiminde kullanılacak eğitim tipini içeren değişkendir.

float m_training_beta;

Sinir ağının eğitimi esnasında değişken eğitim oranı kullanılması durumda eğitim oranının arttırılma katsayısının tutulduğu değişkendir.

float m_training_gamma;

Sinir ağının eğitimi esnasında değişken eğitim oranı kullanılması durumda eğitim oranının azaltılma katsayısının tutulduğu değişkendir.

float m_desired_error;

Sinir ağının eğitiminde ulaşılmak istenen hata miktarını belirten değişkendir.

float m_results[2]

Sinir Ağı eğitimini sonuçlarını tutan dizi tipi değişkendir. Dizinin ilk elemanı eğitimin son adımında ulaşılan hata değerini, ikinci elemanı ise eğitimin sonlandırıldığı adım sayısını tutmaktadır.

4.1.2.2 Yapay sinir ağı sınıfı Üye işlevleri

CNet(int input=1, int hidden=1,int *hidden_nodes=NULL, int output=1)

Sinir ağı sınıfının kurucu fonksiyonudur.

Fonksiyonun parametreleri

int input : sinir ağının giriş katında bulunan hücre sayısıdır.

int hidden: sinir ağının gizli katmanında bulunan katman sayısıdır.

int * hidden:Sinir ağının gizli katmanlarında bulunan hücre sayısının tutulduğu dizi parametresidir

int output: Sinir ağının çıkış katmanında bulunan hücre sayısıdır.

virtual ~CNet()

Sinir ağı sınıfının yok edici fonksiyonudur.

bool Compute_BFGS();

BFSG eğitim yöntemi için kullanılan Hessian matrisinin yaklaşık tersini hesaplamaktadır.

float * get_error_array();

Ağın eğitimi sonlandığında her bir adım için ağın yaptığı hata miktarının tutulduğu diziyi almak için kullanılmaktadır.

void Set_Weights(float *weights);

Sinir ağının ağırlık vektörüne değer atamak için kullanılmaktadır.

İşlevin parametreleri

float *weights: Ağırlık değerlerinin bulunduğu dizi parametresidir.

void Set_Output_Bias(float *bias);

Sinir ağının çıkış katmanı beslemelerinin ağırlık değerlerini atamak için kullanılmaktadır.

İşlevin parametreleri

float *bias: Ağırlık değerlerinin bulunduğu dizi parametresidir.

void Set_Hidden_Bias(float *bias);

Sinir ağının gizli katman beslemelerinin ağırlık değerlerini atamak için kullanılmaktadır.

İşlevin parametreleri

float *bias: Ağırlık değerlerinin bulunduğu dizi parametresidir.

void Set_Hidden_Weights(float **weights);

Sinir ağının gizli katmanlarının ağırlık vektörüne değer atamak için kullanılmaktadır.

İşlevin parametreleri

float **weights: Ağırlık değerlerinin bulunduğu dizi parametresidir.

void Set_Output_Weights(float **weigths);

Sinir ağının çıkış katmanının ağırlık vektörüne değer atamak için kullanılmaktadır.

İşlevin parametreleri

float **weights: Ağırlık değerlerinin bulunduğu dizi parametresidir.

void set_training_set(int num_train_vector, float **input, float **output);

Sinir ağının eğitiminde kullanılacak olan eğitim setine değer atamak için kullanılmaktadır.

İşlevin parametreleri

int_num_train_vector : Eğitim setinde bulunan eğitim çifti sayısını gösteren parametredir.

float **input: Eğitim seti giriş vektörlerinin bulunduğu parametredir.

float **output: Eğitim seti çıkış vektörlerinin bulunduğu parametredir.

void set_training_params(int num_epochs,float desired_error=0);

Sinir ağının eğitiminde kullanılacak parametrelerin atanması için kullanılmaktadır.

İşlevin parametreleri

int_num_epochs : Eğitimin devam etmesi istenen adım sayısının üst sınırını belirtmektedir.

float desired_error: Eğitimin sonlandırılacağı hata değerini belirtmektedir.

void set_training_type(int training_type);

Sinir ağının eğitiminde kullanılacak eğitim tipini belirtmek için kullanılmaktadır.

İşlevin parametreleri

int training_type : Eğitim tipini belirtmektedir.

void set_activation_type(int *hidden_act,int output_act);

Sinir ağının katmanlarında kullanılacak olan aktivasyon fonksiyonlarının tiplerini belirtmek için kullanılmaktadır.

İşlevin parametreleri

int *hidden_act : Gizli katmanların aktivasyon fonksiyonlarının tipinin tutulduğu dizi parametresidir.

int output_act : Çıkış katmanının aktivasyon fonksiyonunun tipinin tutulduğu parametredir.

void set_training_rate(float learning_rate, float training_beta,float training_gamma,float momentum_coef);

Sinir ağının eğitiminde kullanılacak eğitim sabitlerinin değerini atamak için kullanılmaktadır.

İşlevin parametreleri

float learning_rate : Sinir ağının eğitim katsayısıdır.

float training_beta : Değişken öğrenme katsayısı için kullanılan arttırma değeridir.

float training_gamma : Değişken öğrenme katsayısı için kullanılan azaltma değeridir.

float momentum_coef : Ağırlık değişimi için kullanılan momentum değeridir.

float * train_network();

Sinir ağının eğitimine başlamak için kullanılan işlevdir.

float* run_net(float * input);

Sinir ağını çalıştırmak için kullanılan işlevdir.

İşlevin parametreleri

float *input : Sinir ağının girişine uygulanacak olan giriş vektörüdür.

BOOL Line_Search(float * dir, float *buffer);

Sinir ağını eğitimi esnasında verilen doğrultuda maliyet fonksiyonunun en küçük olduğu noktayı bulmak için kullanılan işlevdir.

İşlevin parametreleri

float * dir : Arama yapılacak doğrultunun tutulduğu dizi tipi parametredir.

float buffer: Arama yapılan yönde en düşük maliyetli noktaya ulaşmak için ağırlıklarda yapılacak değişikliklerin döndürüldüğü dizi tipi değişkendir.

float Get_Epoch_Error();

Sinir ağının eğitimi esnasında ağın o anki durumunda yaptığı hata miktarını döndüren işlevdir.

void Network_Line(float * origin, float* dir, float alpha);

Sinir ağı ağırlıklarının verilen yönde, verilen adım miktarı kadar değiştiren işlevdir.

İşlevin parametreleri

float * origin : Ağın mevcut durumdaki ağırlık değerlerinin tutulduğu dizi tipi değişkendir.

float *dir: Ağırlıkların değişim yönünün tutulduğu dizi tipi değişkendir.

float alpha: Ağırlıkların değişim miktarını belirten değişkendir.

void RieberePolak(float*dir);

Ribiere Polak eğitim yöntemine göre ağırlık değişimlerinin yapılması gereken yönü hesaplayan işlevdir.

void FletcherReeves(float*dir);

Fletcher Reeves eğitim yöntemine göre ağırlık değişimlerinin yapılması gereken yönü hesaplayan işlevdir.

void Conjugate_Gradients_Dir(float *dir, float beta);

Konjüge Gradyan eğitim yöntemlerinde gradyan doğrultusunun bulunmasında kullanılan işlevdir.

float Compute_Beta(int type);

Konjüge gradyan yöntemlerde adım büyüklüğünü belirleyen işlevdir.

float DerivDir(float * dir);

Yaklaşık Newton yöntemini kullanan eğitim algoritmalarında arama doğrultusu ile gradyan doğrultusunun dikliğini belirleyen işlevdir.

void BFGS_Dir(float *dir);

Yaklaşık Newton yöntemini kullanan eğitim algoritmalarında ilerleme yönünü belirleyen işlevdir.

void Set_BFGS_Unit();

Yaklaşık Newton yöntemini kullanan eğitim algoritmalarında kullanılan BFGS matrisinin değerini birim matrise çeviren işlevdir.

4.2. Yapay Sinir Ağı Bileşeni Modellemesi

Yapay sinir Ağı Bileşeninin gerçeklenmesinde Visual C++ ile birlikte sağlanan MFC desteği kullanılmıştır. Yapay sinir ağı bileşeninin temel sınıfı MFC içerisinde yer alan COleControl sınıfıdır. MFC kullanılarak gerçeklenen ActiveX bileşenlerinin bu sınıftan türetilmesi zorunludur.

COleControl sınıfından türetilen yapay sinir ağı bileşeni sınıfına temel sınıfın özelliklerine ilave olarak metotlar, özellikler ve olaylar eklenmiştir. Bileşen sınıfının UML ile gösterimi Şekil 4-4' de verilmiştir.

Bileşene ait metotlar ve özellikler bileşenin dış dünya ile olan ara yüzünün oluşturmaktadır. Dış dünyadan uygulamalar bu ara yüz ile bileşene ait verilere ulaşabilmekte ve bileşene isteklerini iletmektedirler.

4.2.1 Yapay sinir hücresi bileşeni özellikleri

[id(1)] long GirisSayisi

Yapay sinir ağı bileşeninin giriş sayısını belirtmektedir.

[id(2)] long CikisSayisi

Yapay sinir ağı bileşeninin çıkış sayısını belirtmektedir.

Şekil 4-4 CIBYSABileseniCtrl' ait UML cösterimi

[id(3)] long GizliKatmanSayisi

Yapay sinir ağı bileşeninin gizli katman sayısını belirtmektedir.

[id(4)] double IstenenHata

Yapay sinir ağının eğitiminde ulaşılmak istenen hata miktarını belirtmektedir.

[id(5)] ACTIVATION CikisAktivasyonu;

Yapay sinir ağı bileşeninin çıkış aktivasyon fonksiyonu tipini belirtmektedir.

[id(6)] long EgitimAdimSayisi

Yapay sinir ağının eğitiminin en çok adım sayısını belirtmektedir.

[id(7)] long EgitimOrnekSayisi

Yapay sinir ağının eğitiminde kullanılacak eğitim çiftlerinin sayısını belirtmektedir.

[id(8)] long GKatmanHucre1

Yapay sinir ağı bileşeninin gizli katmanındaki hücre sayısını belirtmektedir.

[id(11)] ACTIVATION GKatmanAktivasyon

Yapay sinir ağının gizli katmanındaki aktivasyon fonksiyonunun tipini belirtmektedir.

[id(14)] TRAININGMETHOD EgitimMetodu

Yapay sinir ağının eğitiminde kullanılacak olan eğitim metodunu belirtmektedir.

4.2.2 Yapay sinir hücresi bileşeni metotları

[id(15)] short AgYarat(short AgTipi)

Belirlenmiş özelliklere sahip bir yapay sinir ağı yaratmak için kullanılan metottur.

[id(18)] double Egit();

Yapay sinir ağını eğitmek için kullanılan metottur.

[id(19)] void EgitimeHazirla();

Yapay sinir ağının eğitiminden önce sinir ağını eğitime hazırlamak için kullanılan metottur.

[id(20)] void EgitimGirdiVektoruEkle(double* InputVector, long index);

Yapay sinir ağının eğitimi için kullanılacak eğitim çiftlerini ağa tanıtmak için kullanılan metottur.

[id(21)] void EgitimCikisVektoruEkle(double* CikisVectoru, long index);

Yapay sinir ağının eğitimi için kullanılacak eğitim çiftlerini ağa tanıtmak için kullanılan metottur.

[id(22)] void VeriIsle(double* Input, double* Output);

Eğitilmiş yapay sinir ağını çalıştırmak için kullanılan metottur.

[id(23)] long AgirlikYukle(double* weights);

Yapay sinir ağına önceden belirlenmiş ağırlıkları yüklemek için kullanılan metottur.

5. ÖRNEK UYGULAMALAR

Gerçeklenen yapay sinir ağı bileşeni, biri doğrusal olmayan fonksiyon tahmini diğeri ise sınıflandırma tipinde olan iki probleme uygulanmış, uygulamalar Windows 98 işletim sistemine sahip, Intel PIII işlemcili ve 256 MB belleğe sahip bir makine üzerinde yapılmıştır. Her bir problem ve eğitim yöntemi için yapılan 25 denemenin sonuçları Matlab Neural Network Toolbox kullanılarak elde edilen sonuçlarla karşılaştırılmıştır.

Fonksiyon tahmini tipinde olan problem sinüs fonksiyonunun tahmin edilmesidir. Sınıflandırma tipinde kullanılan problem ise kanser hastalığının teşhis problemidir. İlk problemde sinüs fonksiyonuna ait giriş çıkış değerleri kullanılarak yapay sinir ağı yardımı ile fonksiyonun tahmin edilmesi amaçlanmaktadır. İkinci problemde ise kanser hastalığına ait belirtiler kullanılarak hastalığın teşhisi amaçlanmaktadır.

5.1. Sinüs Fonksiyonun Tahmin Problemi

5.1.1. Problemin tanımı

Sinüs fonksiyonu doğrusal olmayan trigonometrik bir fonksiyonudur. Sinüs fonksiyonunun eşitliği aşağıda verilmiştir:

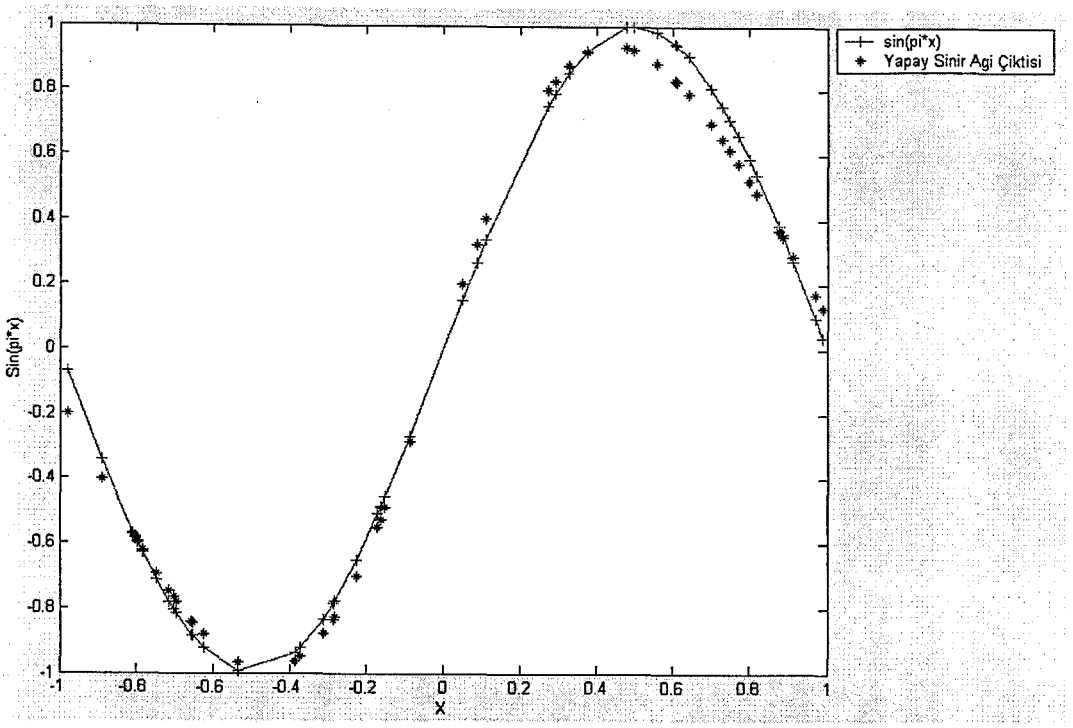
$$y = \sin(\pi x) \tag{5-1}$$

Yapay sinir ağı tarafından sinüs fonksiyonunun [-π , π] aralığındaki bir periyodu tahmin edilmeye çalışmıştır.Bu amaçla eğitim seti olarak [-π , π] aralığında rasgele seçilen 50 nokta kullanılmıştır. Sinir ağını test etmek için [-π , π] aralığında seçilen 50 nokta kullanılmıştır. Eğitim amacıyla kullanılan verilerin ve yapay sinir ağı tarafından tahmin edilen fonksiyonun grafiği Şekil 5-1' de verilmiştir.

5.1.2. Öğrenme algoritmaların karşılaştırılması

Yapay sinir ağı tarafında sinüs fonksiyonunun öğrenilmesi için bir giriş, bir çıkışı ve bir gizli katmanı olan, gizli katmanında 8 sinir hücresi olan bir ağ kullanılmıştır. Aktivasyon fonksiyonu olarak gizli katmanda hiperbolik tanjant çıkış katmanında ise doğrusal aktivasyon fonksiyonu kullanılmıştır. Ağın başlangıç durumunda ağırlıklara (-1,1) aralığında rasgele değerler atanmış, ağ hata oranı 0.001' e düşene kadar eğitim sürdürülmüştür. Değişik eğitim algoritmaları için elde edilen adım sayıları ve eğitim süreleri Çizelge 5-1' de verilmiştir. Eğitim

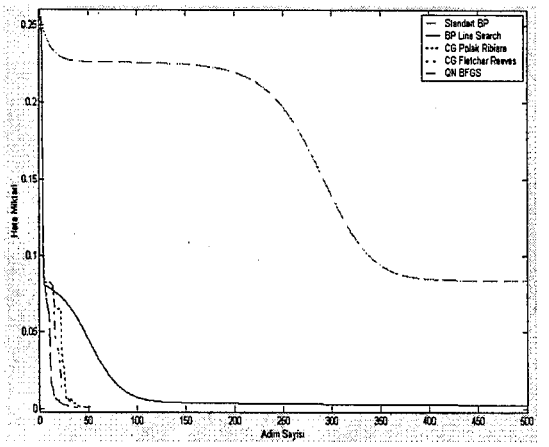
algoritmaları için adım sayısı ve hata miktarının değişim grafikleri Şekil 5-2'de verilmiştir.



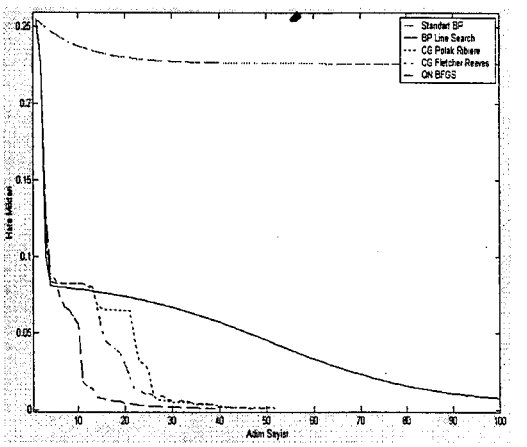
Şekil 5-1 Hesaplanan ve tahmin edilen sinüs fonksiyonun grafiği

Çizelge 5-1 Sinüs problemi, eğitim yöntemleri için ortalama adım sayısı ve eğitim süresi

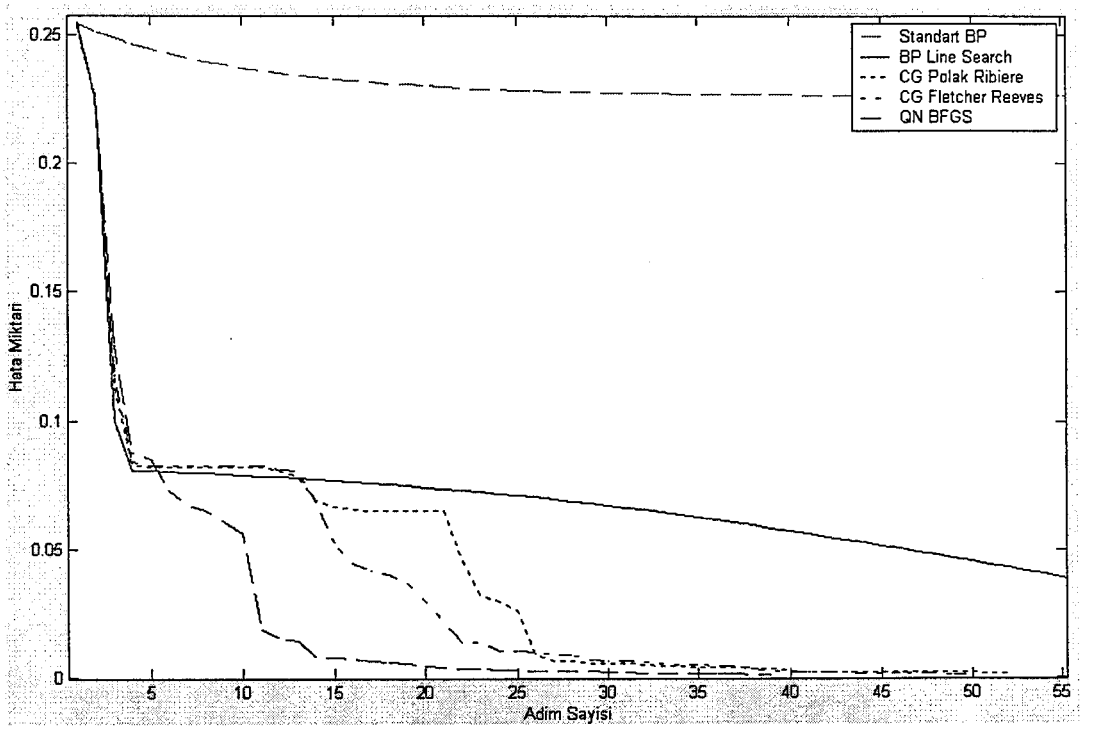
	IBYSA Bileşeni			**Matlab Neural Network Toolbox**		
	Ortalama Adım Sayısı	**Ortalama Süre(ms)**	**Ortalama Adım Süresi(ms)**	**Ortalama Adım Sayısı**	**Ortalama Süre(ms)**	**Ortalama Adım Süresi(ms)**
BackPropogation	45430	70598	1,554	57850	432833,7	7,482
BackPropogation Line Search	2240	10212	4,559	--------	-------	-------
Konjuge Gradyan Polak Ribiere	74	380,878	5,147	76	2042,576	26,876
Konjuge Gradyan Fletcher Reeves	67	330,176	4,928	71	1824,416	25,696
Quasi Newton BFGS	46	274,988	5,978	38	1147,41	30,195



(a)



(b)



(c)

Şekil 5-2 Eğitim algoritmaları için hatanın adım sayısıyla değişme grafiği, (a) ilk 500 adım, (b) ilk 100 adım, (c) ilk 50 adım

5.2. Kanser Hastalığı Teşhis Problemi

5.2.1. Problem tanımı

Bu çalışmada kullanılan göğüs kanseri veritabanı University of Wisconsin Hospitals, Madison Dr.William H. Wolberg 'den elde edilmiştir[29]. 699 örnekten

her biri dokuz özellik ve iyi huylu ya da kötü huylu olmak üzere bir sınıf bilgisi içermektedir.

Özellikler şunlardır:

a) Kütle kalınlığı
b) Hücre büyüklüğünün düzgünlüğü
c) Hücre şeklinin düzgünlüğü
d) Marjinal yapışma
e) Tek epithelial hücre büyüklüğü
f) Yalın çekirdek
g) Hafif kromatin
h) Normal Nükleoli
i) Mitozlar

Bu özellikler dokuz farklı skalada dış görünüm ve kromozom değişikliklerini ölçmektedir. Bütün veriler 1 ile 10 arasında değişen değerlere sahip olup sınıf bilgisi olarak kötü huylu ve iyi huylu olarak ikiye ayrılmaktadır.

5.2.2. Öğrenme algoritmalarının karşılaştırılması

Kanser teşhis problemi için dokuz girişi, bir çıkışı ve bir gizli katmanı olan, gizli katmanında 15 sinir hücresi olan bir ağ kullanılmıştır. Aktivasyon fonksiyonu olarak gizli katmanda hiperbolik tanjant çıkış katmanında ise doğrusal aktivasyon fonksiyonu kullanılmıştır. Ağın başlangıç durumunda ağırlıklara (-1,1) aralığında rasgele değerler atanmış, ağ hata oranı 0.012' ye düşene kadar eğitim sürdürülmüştür. Eğitim esnasında veri kümesinin ilk 200 elemanı kullanılmıştır. Değişik eğitim algoritmaları için elde edilen adım sayıları ve eğitim süreleri Çizelge 5-2' de verilmiştir.

Çizelge 5-2 Kanser problemi, eğitim yöntemleri için ortalama adım sayısı ve eğitim süresi

	IBYSA Bileşeni			Matlab Neural Network Toolbox		
	Ortalama Adım Sayısı	**Ortalama Süre(ms)**	**Ortalama Adım Süresi (ms)**	**Ortalama Adım Sayısı**	**Ortalama Süre(ms)**	**Ortalama Adım Süresi (ms)**
BackPropogation	7528	85405	11,345	91321	1388718,4	15,207
BackPropogation Line Search	2372	71022,4	29,942	--------	---------	--------
Konjuge Gradyan Polak Ribiere	90	3104,46	34,494	100	4242,8	42,428
Konjuge Gradyan Fletcher Reeves	83	2790,46	33,620	108	4491,29	41,586
Quasi Newton BFGS	60	3368,52	56,142	73	5936,65	81,324

6. SONUÇ VE ÖNERİLER

Yapay sinir ağlarının, görsel programlama araçları ile geliştirilen uygulama programları içerisinde kolayca kullanılmasını sağlayacak bir yapay sinir ağı bileşeni oluşturulmuştur.

Oluşturulan yapay sinir ağı bileşenin eğitiminde ikinci dereceden gradyan bilgisi kullanan eğitim yöntemlerinin uygulanması sinir ağı eğitim süresini önemli ölçüde azaltmıştır. Bileşen içerisinde kullanılan ikinci derece gradyan bilgisine dayanan eğitim yöntemleri ile standart geri yayılım algoritmasına oranla eğitim adımı sayısında yaklaşık 90 kat azalma sağlanmış, eğitim hızı süre olarak 30 kat azaltılmıştır. İkinci dereceden gradyan bilgisine dayanan eğitim algoritmaları kullanılması ile ağ eğitiminde kullanıcı tarafından belirlenen parametre sayısı azaltılmıştır. Doğru arama algoritmaları kullanımı ile özellikle deneme yanılma yoluyla belirlenen öğrenme sabiti, momentum gibi parametreler olmadan sinir ağlarının eğitilmesi sağlanmıştır.

Yapay sinir ağı bileşenin performansı örnek problemler ile sınanmış, sonuçlar Matlab Neural Network Toolbox kullanılarak elde edilen sonuçlarla karşılaştırılmıştır. Karşılaştırma sonucunda sinir ağının yakınsaması için gereken adım sayısının yaklaşık aynı olduğu ancak yakınsama için geçen sürenin, sinir ağı bileşeni kullanıldığında %40 oranında azaldığı görülmüştür.

Oluşturulan yapay sinir ağı bileşeni, örnek bir uzman sistem uygulamasında kullanılmış ve uzman sistemin uygulama programı geliştirilirken, yapay sinir ağı kullanımın kolaylaştırdığı ve uygulama geliştirme süresinin azaltıldığı gösterilmiştir. Yapay sinir ağı bileşeninin yapısının sadece spesifik uygulamalara özgü olmayışı, yapay sinir ağlarının uygulanabileceği her türlü alanda uygulama yapılmasına imkan sağlamaktadır. Ağın uygulama anında eğitilebilmesi yada eğitilmiş ağın yüklenerek hazır bir şekilde kullanılabilmesi uygulama programlarında kullanım için büyük bir avantaj sağlamaktadır.

Yapay sinir ağı bileşeni kullanımı ile örüntü tanımadan, bilgisayarlı kontrol uygulamalarına, zaman serisi tahmin problemlerinden uzman sistem oluşturmaya alanda kadar bir çok kendi başına çalışabilen yapay sinir ağı simülatörlerinden bağımsız bilgisayar uygulaması geliştirilmesi daha kısa sürede daha az çaba harcayarak mümkün olacaktır.

Yapay sinir ağı alt yapısının hazır olarak sunulmuş olması uygulama geliştiricilere üzerinde çalıştıkları problemin detayları ile ilgilenmeleri için zaman kazandıracak, uygulama geliştirme sürecinde ayrıca yapay sinir ağı gerçeklemesi için zaman harcanmayacaktır. Yapay sinir ağlarının uygulama programlarında zahmetsizce kullanılabilmesi, yapay sinir ağlarının farklı uygulama alanlarına ve uygulama programlarında kullanılmasının yolunu açacak uygulama geliştiricileri programlar içerisinde yapay sinir ağı kullanmaya yönlendirecektir.

Sonuç olarak geliştirilen yapay sinir ağı bileşeni ile, farklı uygulamalar için rahatlıkla kullanılabilecek, uygulamanın tipine göre özelleştirilebilecek ve bağımsız uygulamalar geliştirilmesine olanak sağlayacak, kullanımı kolay ve esnek bir yapay sinir ağı alt yapısı oluşturulmuştur.

KAYNAKLAR

[1] HAYKIN, S., *Neural networks: a comprehensive foundation*, Prentice Hall, New Jersey, USA (1999).

[2] FREDRIC, M.H., *Principles of neurocomputing for science and engineering*, McGraw Hill, New York USA (2001).

[3] DARPA, *Classification and clustering models*, Neural Networks Study, AFCEA Internatioanl Press (1988) .

[4] SAĞIROĞLU, Ş., *Mühendislikte yapay zeka uygulamaları-I*, Ufuk Yayıncılık, Kayseri, Türkiye (2003).

[5] BISHOP,C.M., *Neural networks for pattern recognition*, Oxford University Press, Oxford, UK (1995).

[6] MATHWORKS, *Matlab neural network toolbox*, http://www.mathworks.com (2004).

[7] MLPX, *Neural network ActiveX component*, Windale Technologies, http://www.windale.com (2004).

[8] NEUNET PRO, *Neural network simulator*, CorMac Technologies, http://www.cormactech.com/neunet/ (2004).

[9] NEURO OFFICE, *Neural network simulator*, Alpha Systems, http://www.user.cityline.ru/~alphasys/ (2004)

[10] MICROSOFT, *The component object model specification(introduction)* draft version 0.9, Microsoft Corporation and Digital Equipment Corporation http://www.microsoft.com/oledev/olecom/title.htm (1995).

[11] ARBIB, M.A., *Brains, machines, and mathematics*, Springer-Verlang, NewYork, USA (1987).

[12] SHEPHERD, G.M., ve KOCH, C., *Introduction to synaptic circuits, the synaptic organization of brain*, Oxford University Press, New York, USA (1990).

[13] SHEPHERD, G.M., *The significance of real neuron architectures for neural network simulations*, Computational Neuroscience, E.L. Schwatz ed s82-96, MIT Press, Combridge, USA (1990).

[14] MINSKY, M.L., ve PAPERT, S.A., *Perseptrons*, MIT Press, MA, USA (1969).

[15] McCULLOCH, W.S., ve PITTS, W., *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics, **5**, 115-133 (1943).

[16] ROSENBALT, F., *The perseptron: aprobabilistic model for information storage and organization in the brain*, Phsycological Review, **65**, 386-408 (1958).

[17] ORR, M.J.L., *Introduction to radial basis function networks*, University of Edinburgh, Scotland, UK, http://www.cns.ed.ac.uk/people/mark/rbf.tar.z (1996).

[18] RUMELHART, D.E., HINTON, G.E. ve WILLIAMS, R.J., *Learning internal representations by error propogation*, in Rumelhart, D.E. and McClelland, J.L., Parallel distributed Processing, Explorations in the Microstructure of Cognition, Cambridge, MA, The MIT Press, **1**, 318-362 (1986).

[19] SARKAR, D., *Methods to speed up error back propogation learning algorithm*, ACM Computing Surveys, 27, **4**, 519-541 (1995).

[20] JOHANSSON, E.M., DOWLA, F.U. ve GOODMAN, D.M., *Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient metho*d, Lawrence Livermore National Laboratory, Preprint UCRL-JC-104850 (1990).

[21] FLETCHER,R., ve REEVES, C.M., *Function minimization by conjugate gradients*, Computer Journal, **7**, 149-154.

[22] PRESS, W.H., FLANNERY, B.P., TEUKOLSKY, S.A., ve VETTERILING, W.T., *Numerical recipes in C: the art of scientific computing*, Cambridge, Cambridge University Press, UK (1988).

[23] NOCEDAL,J. ve WRIGHT,S.J., *Numerical optimizatio*n. New York: Springer-Verlag, (1999).

[24] DENNIS, J.E., ve SCHNABEL, R.B., *Numerical methods for unconstrained optimization and nonlinear equation*s. Englewood Cliffs, NJ, Prentice-Hall, (1983).

[25] WILLIAMS S., *The component object model: a technical overview*, Microsoft Corporation,Washington, USA (1994).

[26] HUSAIN, K., *ActiveX developer's resource*, Prentice Hall, New Jersey, USA (1997).

[27] KRUGLINKSI, D.J., *Programming visual C++*, Microsoft Press, Washington, USA (1998).

[28] PROSISE, J., *Programming windows with MFC*, Microsoft Press, Washington, USA (1999).

[29] WOLBERG, W.H., *Wisconsin breast cancer database*, Madison, USA http://ftp.ics.uci.edu/pub/ml-repos/machine-learning-databases/ (1992).

EK - 1 Yapay Sinir Ağı Bileşeni Kaynak Kodları Başlık Dosyaları

Node.h Dosyası

```
#if
!defined(AFX_NODE_H__F7C57641_2998_11D7_AF1F_00C02665C635__INCLUDE
D_)
#define
AFX_NODE_H__F7C57641_2998_11D7_AF1F_00C02665C635__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Node.h : header file
// CNode window
class CNode : public CObject
{
      public:
      void compute_gradients(float delta);
      void set_gradient(float *);
      void set_bias(float bias);
      float * get_weighted_delta(float delta);
      float get_derived_sum(int type) const;
      CNode(int num_inputs=1);
      void set_input_set(float * input);
      bool set_weights(float * weights);
      float output(int type=1);
      float get_sum(int type) const;
      void sum_node();
      // Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CNode)
      //}}AFX_VIRTUAL
      virtual ~CNode();
      private:
      float activation(float x,int type=1,float slope =1) const;
      float activation_derivative(float x,int type=1) const;
      float *m_weights;//node agirliklari
      float *m_old_weights;//node agirliklari
      float *m_input_vector;//girişler
      float  m_output;//node çıkışı
      int    m_num_inputs;//giriş sayısı
      float  m_weighted_sum;//girişler*ağırlıklar toplamı
      float  m_delta_error;//
      float * m_weighted_delta ;
      float m_bias;
      float *m_gradient;
};
//{{AFX_INSERT_LOCATION}}
#endif//
!defined(AFX_NODE_H__F7C57641_2998_11D7_AF1F_00C02665C635__INCLUDE
D_)
```

Net.h Dosyası

```
#if
!defined(AFX_NET_H__F7C57642_2998_11D7_AF1F_00C02665C635__INCLUDED
_)
#define AFX_NET_H__F7C57642_2998_11D7_AF1F_00C02665C635__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Net.h : header file
#include <Afx.h>
#include "Node.h"
////////SABITLER
#define SIGMOID 1
#define TANH 2
#define HARDLIM 3
#define LINEAR 4
#define PATTERN 1
#define BATCH 2
#define STEEPEST_DESCENT 3
#define CG_RIBIERE_POLAK 4
#define CG_FLETCER_REEVES 5
#define CG_BFGS 7
class CNet : public CObject
{
// Construction
public:
      CNet(int input=1, int hidden=1,int *hidden_nodes=NULL,int
      output=1);
      virtual ~CNet();
      public:
      // Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CNet)
      //}}AFX_VIRTUAL
      // Implementation
      void  Set_Weights(float *weights);
      void  Set_Output_Bias(float *bias);
      void  Set_Hidden_Bias(float *bias);
      void  Set_Hidden_Weights(float **weights);
      void  Set_Output_Weights(float **weigths);
      void  set_training_set(int num_train_vector, float **input,
                              float **output);
      void  set_training_params(int num_epochs,float
                              desired_error=0);
      void  set_training_type(int training_type);
      void  set_activation_type(int *hidden_act,int output_act);
      void  set_training_rate(float learning_rate, float
            training_beta,float training_gamma,float
            momentum_coef);
      float *train_network();
      float *run_net(float * input);
      bool  Compute_BFGS();
      float *get_error_array();
      private:
      void  Steepest_Dir(float *dir);
      BOOL  Line_Search(float * dir, float *buffer);
      float Get_Epoch_Error();
      void  Network_Line(float * origin, float* dir, float alpha);
```

```
        void  RieberePolak(float*dir);
        void   FletcherReeves(float*dir);
        void   Conjugate_Gradients_Dir(float *dir, float beta);
        float  Compute_Beta(int type);
        void   Set_Gradient();
        float *m_network_weights;
        float *m_network_old_weights;
        int    m_num_network_weights;
        float *m_network_gradient;
        float *m_network_old_gradient;
        float *m_network_epoch_gradient;
        float *m_network_epoch_old_gradient;
        float  m_last_alpha;
        float **m_BFGS;
        float *m_gradient_delta;
        float *m_weight_delta;
        float  DerivDir(float * dir);
        void   BFGS_Dir(float *dir);
        void   Set_BFGS_Unit();
        float  m_Tau;
        float *m_output;
        int    m_num_hidden_layer;
        int    m_num_input;
        int    m_num_output;
        int   *m_num_hidden_layer_nodes;
        CNode *** m_hidden_layer;
        CNode ** m_output_layer;
        int   *m_hidden_activation;
        int    m_output_activation;
        float *m_hidden_input;
        float **m_hidden_output;
        float **m_hidden_sum;
        float *m_input_set;
        float *m_desired_set;
        int    m_num_train_vector;
        float **m_train_input;
        float **m_train_output;
        int    m_num_epochs;
        float  m_learning_rate;
        float  m_momentum_coefficient;
        int    m_training_type;
        float  m_training_beta;
        float  m_training_gamma;
        float  m_desired_error;
        float *m_output_delta;
        float *m_hidden_delta;
        float *m_total_error;
        float  m_results[2]; // 0->last error 1-> last epoch number
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_NET_H__F7C57642_2998_11D7_AF1F_00C02665C635__INCLUDED
_)
```

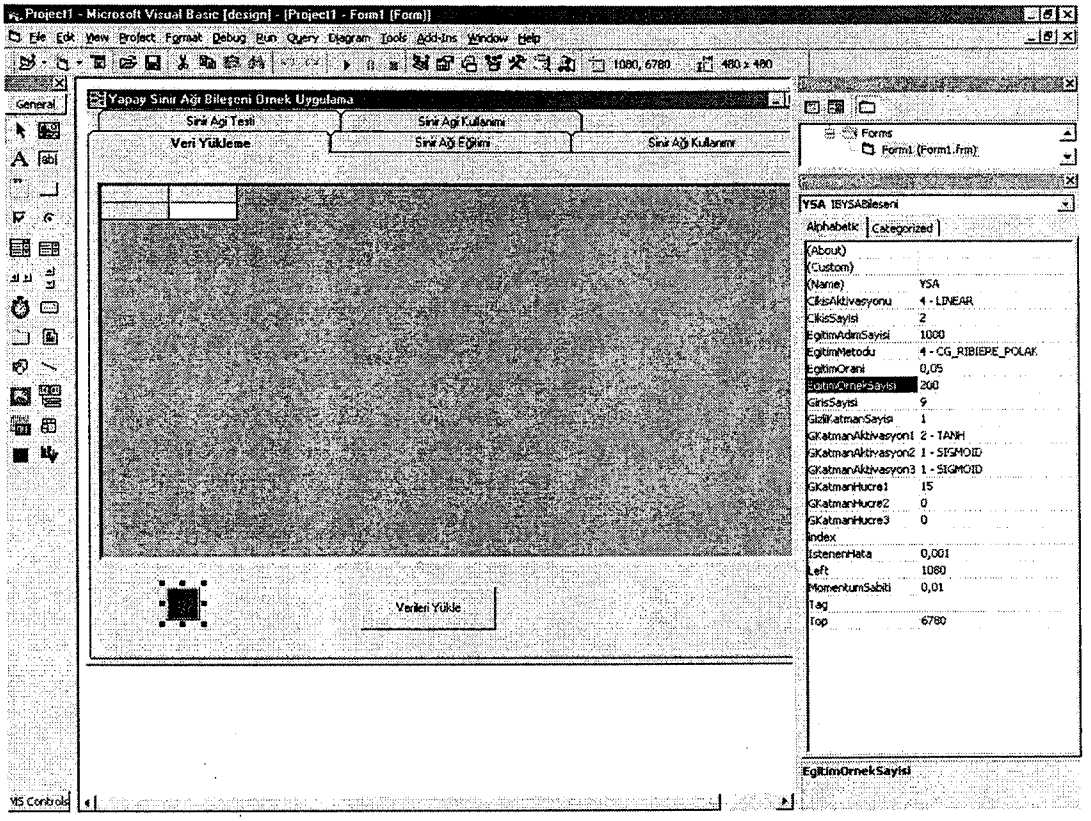
IBYSABileseniCtrl.h Dosyası

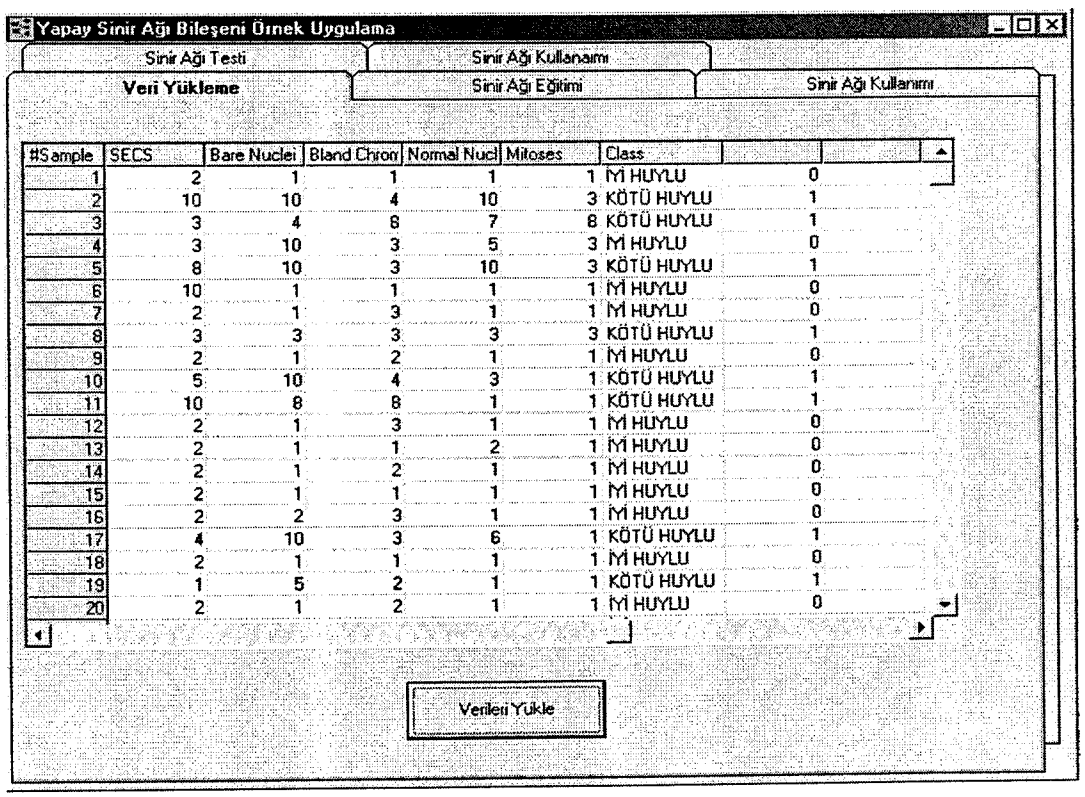
```
#if
!defined(AFX_IBYSABILESENICTL_H__13318274_CAB6_11D8_AFAB_BE237F79B
          E60__INCLUDED_)
#define
AFX_IBYSABILESENICTL_H__13318274_CAB6_11D8_AFAB_BE237F79BE60__INCL
UDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "net.h"
// IBYSABileseniCtl.h : Declaration of the CIBYSABileseniCtrl
ActiveX Control class.
// CIBYSABileseniCtrl : See IBYSABileseniCtl.cpp for
implementation.
class CIBYSABileseniCtrl : public COleControl
{
      DECLARE_DYNCREATE(CIBYSABileseniCtrl)
// Constructor
public:
      CIBYSABileseniCtrl();
// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CIBYSABileseniCtrl)
      public:
      virtual void OnDraw(CDC* pdc, const CRect& rcBounds, const
                        CRect& rcInvalid);
      virtual void DoPropExchange(CPropExchange* pPX);
      virtual void OnResetState();
      //}}AFX_VIRTUAL
// Implementation
protected:
      ~CIBYSABileseniCtrl();
      DECLARE_OLECREATE_EX(CIBYSABileseniCtrl) // Class factory
                                               and guid
      DECLARE_OLETYPELIB(CIBYSABileseniCtrl)   // GetTypeInfo
      DECLARE_PROPPAGEIDS(CIBYSABileseniCtrl)  // Property page
                                               IDs
      DECLARE_OLECTLTYPE(CIBYSABileseniCtrl)   // Type name
                                               and misc status
// Message maps
      //{{AFX_MSG(CIBYSABileseniCtrl)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
// Dispatch maps
      //{{AFX_DISPATCH(CIBYSABileseniCtrl)
      long m_num_input;
      afx_msg void OnGirisSayisiChanged();
      long m_num_output;
      afx_msg void OnCikisSayisiChanged();
      long m_num_hidden_layer;
      afx_msg void OnGizliKatmanSayisiChanged();
      double m_desired_error;
      afx_msg void OnIstenenHataChanged();
      short m_output_activation;
      afx_msg void OnCikisAktivasyonuChanged();
      long m_num_training_epoch;
      afx_msg void OnEgitimAdimSayisiChanged();
```

```
        long m_num_train_vector;
        afx_msg void OnEgitimOrnekSayisiChanged();
        long m_num_hidden_layer_node_1;
        afx_msg void OnGKatmanHucre1Changed();
        long m_num_hidden_layer_node_2;
        afx_msg void OnGKatmanHucre2Changed();
        long m_num_hidden_layer_node_3;
        afx_msg void OnGKatmanHucre3Changed();
        short m_hidden_layer_activation_1;
        afx_msg void OnGKatmanAktivasyon1Changed();
        short m_hidden_layer_activation_2;
        afx_msg void OnGKatmanAktivasyon2Changed();
        short m_hidden_layer_activation_3;
        afx_msg void OnGKatmanAktivasyon3Changed();
        short m_training_type;
        afx_msg void OnEgitimMetoduChanged();
        afx_msg short CreateNet(short AgTipi);
        afx_msg double Train_Net();
        afx_msg void Initiate_Training();
        afx_msg void AddTrainInputVector(double FAR* InputVector,
                                         long index);
        afx_msg void AddTarinOutoutVector(double FAR* CikisVectoru,
                                         long index);
        afx_msg void RunNet(double FAR* Input, double FAR* Output);
        afx_msg long LoadWeights(double FAR* weights);
        //}}AFX_DISPATCH
        DECLARE_DISPATCH_MAP()
        afx_msg void AboutBox();
// Event maps
        //{{AFX_EVENT(CIBYSABileseniCtrl)
        //}}AFX_EVENT
        DECLARE_EVENT_MAP()
// Dispatch and event IDs
public:
        enum {
        //{{AFX_DISP_ID(CIBYSABileseniCtrl)
        dispidGirisSayisi = 1L,
        dispidCikisSayisi = 2L,
        dispidGizliKatmanSayisi = 3L,
        dispidIstenenHata = 4L,
        dispidCikisAktivasyonu = 5L,
        dispidEgitimAdimSayisi = 6L,
        dispidEgitimOrnekSayisi = 7L,
        dispidGKatmanHucre1 = 8L,
        dispidGKatmanHucre2 = 9L,
        dispidGKatmanHucre3 = 10L,
        dispidGKatmanAktivasyon1 = 11L,
        dispidGKatmanAktivasyon2 = 12L,
        dispidGKatmanAktivasyon3 = 13L,
        dispidEgitimMetodu = 14L,
        dispidAgYarat = 15L,
        dispidEgitimGirdiVerisiEkle = 16L,
        dispidEgitimCikisVerisiEkle = 17L,
        dispidEgit = 18L,
        dispidEgitimeHazirla = 19L,
        dispidEgitimGirdiVektoruEkle = 20L,
        dispidEgitimCikisVektoruEkle = 21L,
        dispidVeriIsle = 22L,
        dispidAgirlikYukle = 23L,
```

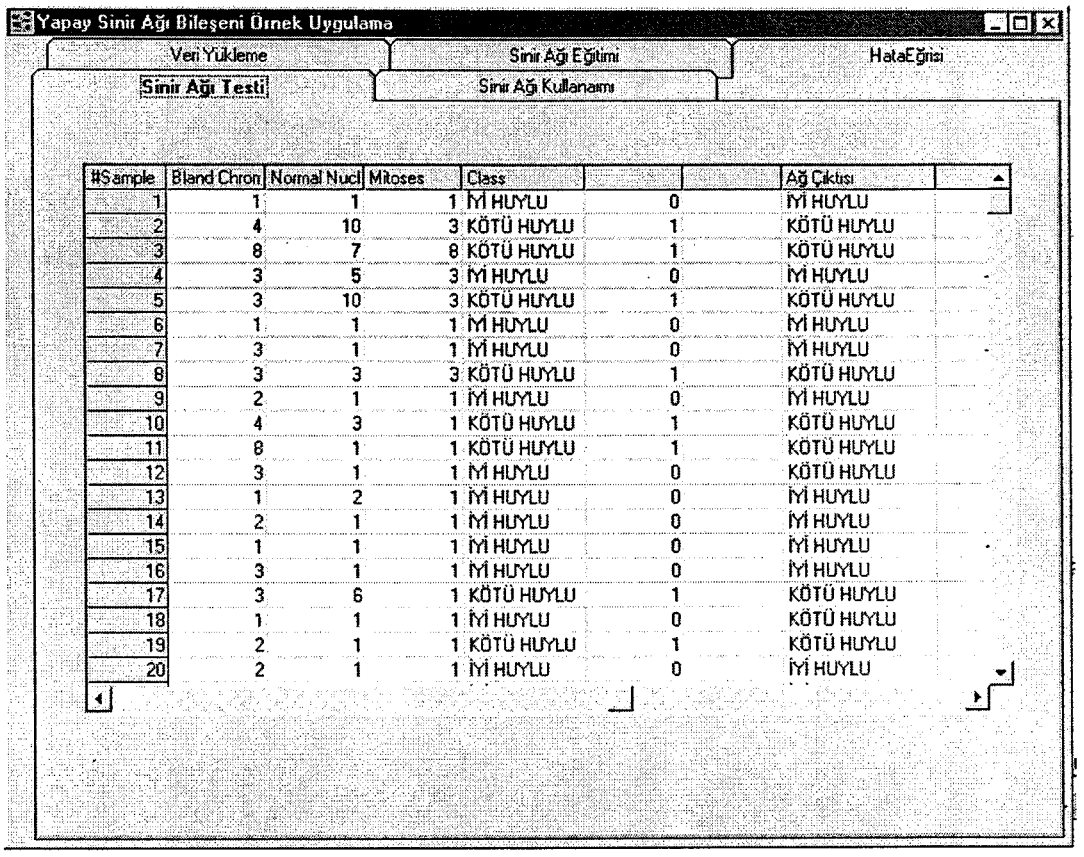
```
      //}}AFX_DISP_ID
      };

private:
      HICON my_icon;
      CNet * m_net;
      int m_num_hidden_layer_nodes[5];
      int m_hidden_activation[5];
      float ** m_train_input;
      float ** m_train_output;
      int m_result_epoch;
//CG için ağın weight matrisi
      float * m_weights;
      int  m_num_network_weights;
      void Set_Random_Weights();
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_IBYSABILESENICTL_H__13318274_CAB6_11D8_AFAB_BE237F79B
E60__INCLUDED)
```

EK - 2 Yapay Sinir Ağı Bileşeni Örnek Uygulama Kanser Teşhis Sistemi Ekran Görüntüleri

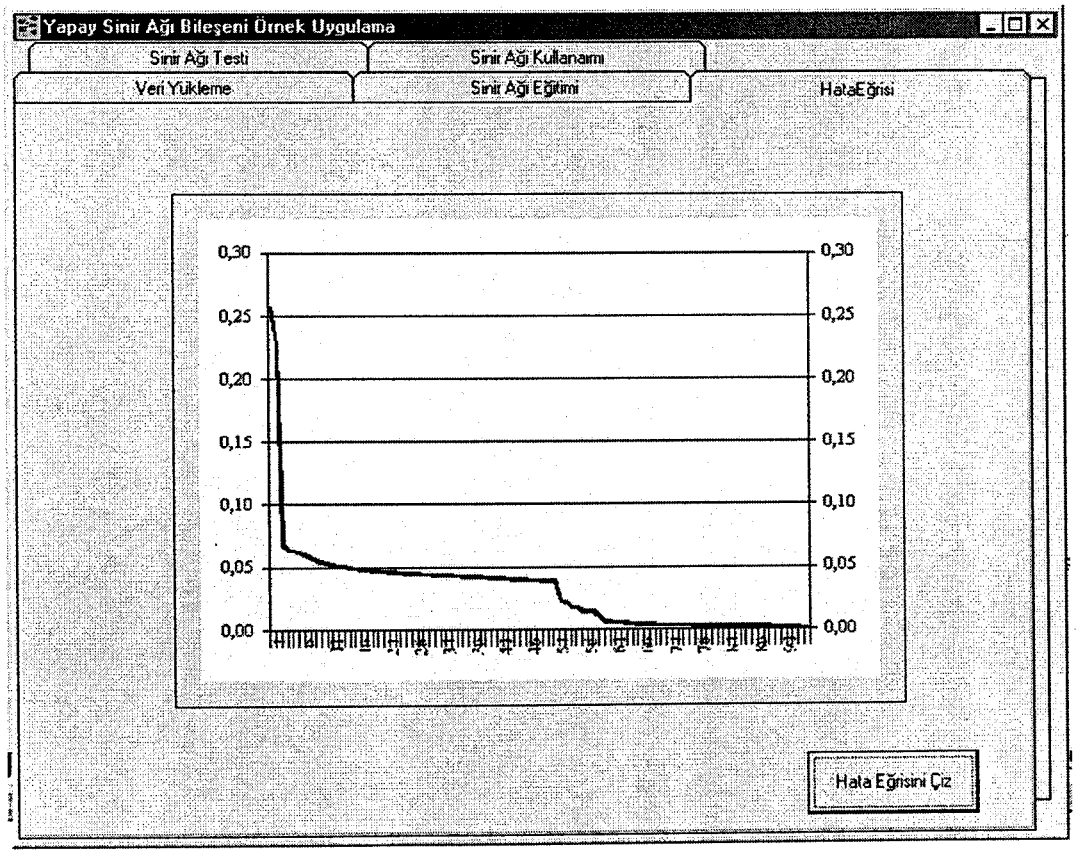




#Sample	SECS	Bare Nuclei	Bland Chron	Normal Nucl	Mitoses	Class	
1	2	1	1	1	1	İYİ HUYLU	0
2	10	10	4	10	3	KÖTÜ HUYLU	1
3	3	4	8	7	8	KÖTÜ HUYLU	1
4	3	10	3	5	3	İYİ HUYLU	0
5	8	10	3	10	3	KÖTÜ HUYLU	1
6	10	1	1	1	1	İYİ HUYLU	0
7	2	1	3	1	1	İYİ HUYLU	0
8	3	3	3	3	3	KÖTÜ HUYLU	1
9	2	1	2	1	1	İYİ HUYLU	0
10	5	10	4	3	1	KÖTÜ HUYLU	1
11	10	8	8	1	1	KÖTÜ HUYLU	1
12	2	1	3	1	1	İYİ HUYLU	0
13	2	1	1	2	1	İYİ HUYLU	0
14	2	1	2	1	1	İYİ HUYLU	0
15	2	1	1	1	1	İYİ HUYLU	0
16	2	2	3	1	1	İYİ HUYLU	0
17	4	10	3	6	1	KÖTÜ HUYLU	1
18	2	1	1	1	1	İYİ HUYLU	0
19	1	5	2	1	1	KÖTÜ HUYLU	1
20	2	1	2	1	1	İYİ HUYLU	0



#Sample	Bland Chron	Normal Nucl	Mitoses	Class			Ağ Çıktısı
1	1	1	1	İYİ HUYLU		0	İYİ HUYLU
2	4	10	3	KÖTÜ HUYLU		1	KÖTÜ HUYLU
3	8	7	8	KÖTÜ HUYLU		1	KÖTÜ HUYLU
4	3	5	3	İYİ HUYLU		0	İYİ HUYLU
5	3	10	3	KÖTÜ HUYLU		1	KÖTÜ HUYLU
6	1	1	1	İYİ HUYLU		0	İYİ HUYLU
7	3	1	1	İYİ HUYLU		0	İYİ HUYLU
8	3	3	3	KÖTÜ HUYLU		1	KÖTÜ HUYLU
9	2	1	1	İYİ HUYLU		0	İYİ HUYLU
10	4	3	1	KÖTÜ HUYLU		1	KÖTÜ HUYLU
11	8	1	1	KÖTÜ HUYLU		1	KÖTÜ HUYLU
12	3	1	1	İYİ HUYLU		0	KÖTÜ HUYLU
13	1	2	1	İYİ HUYLU		0	İYİ HUYLU
14	2	1	1	İYİ HUYLU		0	İYİ HUYLU
15	1	1	1	İYİ HUYLU		0	İYİ HUYLU
16	3	1	1	İYİ HUYLU		0	İYİ HUYLU
17	3	6	1	KÖTÜ HUYLU		1	KÖTÜ HUYLU
18	1	1	1	İYİ HUYLU		0	KÖTÜ HUYLU
19	2	1	1	KÖTÜ HUYLU		1	KÖTÜ HUYLU
20	2	1	1	İYİ HUYLU		0	İYİ HUYLU



EK - 3 Yapay Sinir Ağı Bileşeni Örnek Uygulama Kaynak Kodları

```
Dim X() As Double        ' Giriş Veri Seti
Dim D() As Double        ' Çıkış Veri Set
Dim InputVec() As Double
Dim OutputVec() As Double
Private Sub Ag_Yarat_Click()
      YSA.AgYarat
End Sub

Private Sub EgitimSetiAta()
      For I = 1 To YSA.EgitimOrnekSayisi
        YSA.EgitimGirdiVektoruEkle X(I, 1), I - 1
      Next I
      For I = 1 To YSA.EgitimOrnekSayisi
        YSA.EgitimCikisVektoruEkle D(I, 1), I - 1
      Next I
End Sub

Private Sub Ag_Test_Et_Click()
      For I = 1 To Num_Train_Set
          YSA.VeriIsle X(I, 1), OutputVec(1)
             If (OutputVec(1) < 0.5) Then '0 İYİ 1 KÖTÜ
                Grid.Text = "İYİ HUYLU"
             ElseIf (OutputVec(1) >= 0.5) Then
                Grid.Text = "KÖTÜ HUYLU"
          End If
      Next I
End Sub

Private Sub Ag_Egit_Click()
      YSA.EgitimeHazirla
      EgitimSetiAta
      YSA.Egit
End Sub
```