**T.C.**

**HALİÇ ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

# YAZILIM PROJELERİNDE YAPAY SİNİR AĞI UYGULAMASI İLE MALİYET TAHMİNİ

**YÜKSEK LİSANS TEZİ**

**Hazırlayan**

**AYSUN SEZER**

**Tez Danışmanı**

**Prof. Dr. ALİ OKATAN**

**Haziran 2008**
**İSTANBUL**

**T.C.**

**HALİÇ ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜNE**

Bilgisayar Mühendisliği Programı Yüksek Lisans öğrencisi Aysun SEZER tarafından hazırlanan **"Yazılım Projelerinde Yapay Sinir Ağı Uygulaması ile Maliyet Tahmini "** adlı bu çalışma jürimizce Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Sınav Tarihi : 13.06.2008

( Jüri Üyesinin Ünvanı , Adı , Soyadı ve Kurumu ) : İmzası :

Jüri Üyesi: Prof.Dr.Ali OKATAN
(Danışman-HÜ.Bilgisayar Müh.ABD Öğr.Üyesi) .....

Jüri Üyesi : Prof.Dr.Süleyman SEVİNÇ
(H.Ü.Bilgisayar Mühendisliği ABD Öğr.Üyesi ) .....

Jüri Üyesi : Prof.Dr.Sami ERCAN
(H.Ü.Endüstri Mühendisliği ABD Öğr.Üyesi) .....

**T.C.**
**HALİÇ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜNE**

Bilgisayar Mühendisliği Programı Yüksek Lisans öğrencisi Aysun SEZER tarafından hazırlanan **"Yazılım Projelerinde Yapay Sinir Ağı Uygulaması ile Maliyet Tahmini "** adlı bu çalışma jürimizce Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Sınav Tarihi : 13.06.2008

( Jüri Üyesinin Ünvanı , Adı , Soyadı ve Kurumu ) : İmzası :

Jüri Üyesi: Prof.Dr.Ali OKATAN
(Danışman-HÜ.Bilgisayar Müh.ABD Öğr.Üyesi) .....

Jüri Üyesi : Prof.Dr.Süleyman SEVİNÇ
(H.Ü.Bilgisayar Mühendisliği ABD Öğr.Üyesi ) .....

Jüri Üyesi : Prof.Dr.Sami ERCAN
(H.Ü.Endüstri Mühendisliği ABD Öğr.Üyesi) .....

# ÖNSÖZ

Yazılım projelerinin başarısı doğru yapılan gereksinim analizi, planlama ve iyi yapılmış tahminlere bağlıdır. Bilgisayar yazılımlarının maliyet tahmini her geçen gün daha fazla önem kazanmaktadır. Çünkü yazılım geliştirme maliyeti her geçen gün artmakta ve firmalar yazılım geliştirme için daha fazla bütçeyi tahsis etmektedirler. Buna rağmen dünya genelinde çöken proje sayısı da hızla artmaktadır. Yazılım maliyet tahmini hem uluslararası hem de organizasyonlar için büyük bir problem teşkil etmektedir. T.C. Haliç Üniversitesi Bilgisayar Mühendisliği bölüm başkanı sayın Prof. Dr. Ali Okatan hocam ile yazılım projelerinde maliyet tahmini üzerinde çalışma yaparak bu sorunların analizini ve çözümü üzerinde araştırma yapmaya karar verdik. Böylece tez konum ortaya koyuldu.

Yüksek lisans öğrenimini yaptığım süre içerisinde ve bu tezin hazırlanması esnasında kıymetli bilgi ve yardımlarını esirgemeyerek büyük fedakârlık gösteren saygı değer hocam Prof. Dr. Ali Okatan'a ve çalışmalarım sırasında bana gerekli kaynakları sağlayan, çalışmalarımda teknik yardımda bulunan Öğr. Gör. Oğuz Karan'a sonsuz teşekkürlerimi sunarım.

Tezi hazırlamamda bana çalışma ortamını hazırlayan, desteğini hiç esirgemeyen sevgili eşim Opt. Dr. Hasan Basri Sezer'e katkılarından dolayı teşekkür ederim.

**İÇİNDEKİLER**

## ŞEKİLLER LİSTESİ

**TABLOLAR LİSTESİ**

## ÖZET

Günümüzde, yazılım sistemleri bankacılıktan otomotiv sanayisine, sağlık bilgi sistemlerinden şirket yönetimine, telekomünikasyon sistemlerinden hava taşımacılığına kadar çok geniş alanlarda kullanılan bilgisayar sistemlerinin çok önemli ve kritik bir parçasını oluşturmaktadır. Yazılım geliştirme ise yazılım sistemlerinin mühendislik prensipleri çerçevesinde tasarımı, üretimi ve işletilmesini hedefler.

Yazılımların giderek karmaşık ve kompleks yapı kazanmasından ölçme işlemini gerçekleştirmek zorlaşmaktadır. Yazılımların kullanıcılar tarafından kolay anlaşılır olması, hazırlanan modüllerin tekrar kullanılabilir olması, bakım yapılabilir olması, kullanıcıların beklentilerini sağlaması, beklenen maliyette tamamlanması ve güvenilir olması hedeflenmektedir. Belirlenen hedeflere ulaşabilmek için doğru gereksinim analizi, iyi bir fizibilite çalışması ve planlama yapılmalıdır. Yapılan planlamalar her zaman iyi tahminler üzerine olmalıdır. Yapılan doğru yazılımın büyüklüğü, işgücü, takvimi ile optimum yazılım maliyet tahmini yapılabilir. Yazılım maliyetini doğru kestirim ile gerçekleşebilecek yazılım riskleri önceden tespit edilerek uygun şekilde yönetimi sağlanabilir.

Yazılım geliştirme maliyeti her geçen gün giderek artmakta ve yazılım firmaları proje geliştirmek için daha fazla harcama yapmaktadırlar. Bu çalışmada yazılım projelerinin maliyetini tahmin etmek amacıyla yapay sinir ağı uygulaması gerçekleştirilmiştir. Yazılım maliyetini en çok etkileyen faktörleri belirlemek için yazılım kestirim metotları olan satır sayısı, fonksiyon puanları, COCOMO 81 modeli ve COCOMO II modeli incelendi. Ayrıca yayın taraması yapılarak uzman görüşleri alındı. Belirlenen faktörlere göre yazılım firmalarından gerçekleşmiş yazılımların belirlenen kriterlere göre veriler toplandı. Elde edilen yazılım proje örneklerinin maliyetleri farklı tarihlere ait olduğu için bütün projelerin maliyeti günümüz maliyetine taşındı. Veriler üzerinde normalizasyon işleminden sonra test ve eğitim verileri olarak iki gruba ayrıldı. Yapay sinir ağı olarak çok katmanlı ileri beslemeli yapay sini ağı seçildi. Eğitim algoritması olarak Delta Algoritmasına karar verildi. Eğitim, örnek veri seti kullanılarak tamamlandıktan sonra test verileri ağa sunularak hedef çıktı elde edildi. Yapay sinir ağı uygulamasından elde edilen veriler COCOMO 2000 verileri ile karşılaştırıldı.

**ABSTRACT**

Nowadays, software systems having a so wide range of use; from banking to automotive sector, from health services to management and from telecommunication to air transport business etc. compose a very important and critic part of computer systems. Software innovation aims software design, production and operation on an engineering basis.

Measurement procedures are getting harder due to the software systems which get even more complex everyday. The aim is to produce programs which are more reliable, understandable to users, to make modules to be reused, to supply an affordable and easy service, to overcome most of the customer expectations. For these purposes to achieve, exact analysis of requirements, a good feasibility search and planning are crucial. The plans must be constructed on realistic expectations. An optimum cost expectation can be done only with a high rate of accurate software and a fine work force calendar. To predict and manage the software costs, the risks must be predicted first.

The cost of software innovation is expanding every day increasing the budget of software firms. In this paper we used neural network applications to predict the cost of software projects. To determine the factors which affect the software costs mostly, we observed the software estimation methods like source lines of codes, function points, COCOMO 81 and COCOMO II models. Besides we made literature search and take authorized support. We take data samples from actual Works of real firms. The costs were also updated. After normalization the data was separated into two groups as test and control groups. We selected multilayer feedforward backpropagation neural network and the Delta Algorithm as training algorithm. After training with example data the test data was given to the neural network application and outputs are obtained. The data taken from neural network application was compared to the COCOMO 2000 data .

# GİRİŞ

Günümüzde teknolojik gelişmelerin ivmelenerek hızla artması üretim kavramını da etkilemiştir. Dinamik sektörde üretim kavramı sadece fiziksel anlamda katma değer sağlayacak ürünü pazara sunmak değildir. Üretim kavramı artık esnek üretim olarak ele alınmaktadır. Esnek üretim doğru zamanda, doğru şekilde, belirlenen kısıtları aşmadan müşteri taleplerini karşılayacak şekilde olmalı. Esnek üretim talepteki değişime adapte olabilmenin yanında değişime çok kısa sürede tepki verebilmeyi gerektirir.

Esnek üretim perspektifinden bakılarak yazılım mühendisliğinde proje geliştirme ve yeni ürünler ortaya koymaya çalıştığımızda doğru ölçümler ve tahminlerin gerçekleştirilmesi gerekliliği kaçınılmaz. Doğru tahminlerin gerçekleşmesi için doğru analizler ve ölçümler yapılmalı. Fakat yazılım projelerinin başarısını etkileyen faktörler arasında ölçülebilen kavramlar kadar ölçülemeyen kavramlarda vardır. Ölçülemeyen kavramlarla belirlenen hedeflere ulaşmakta imkânsızlaşmıştır. Ölçme ve değerlendirme yapabilmek için iyi bir plan oluşturmak ve sistematik olarak çalışmak gerekmektedir.

Dünya genelinde yazılım projelerinin başarısızlığı incelendiğinde projenin kısıtlarının tam olarak belirlenememesi, doğru maliyet tahmininin yapılamaması, değişen müşteri beklentilerinin karşılanamaması, çalışanların teknik donanımının yetersizliği, müşterinin beklentilerini tam olarak yansıtamaması yatmaktadır. Fakat yazılım projelerinin büyük bir kısmı yanlış maliyet tahmini ve zaman aşımından dolayı çökmektedir. Yapılan yanlış tahminlerden dolayı yazılım maliyetleri hızla artmakta. Bu da hem ülke genelinde hem de dünya genelinde önemli bir sorun olarak karşımıza çıkmaktadır.

Bu çalışma yazılım projelerinden elde edilen örnek veri kümeleri üzerinde yapay sinir ağı uygulaması ile maliyet tahmini üzerine yapılmıştır. Öncelikle yazılım kestirim metotları olan fonksiyon noktası yöntemi, satır sayısı yöntemi, COCOMO 81, COCOMO II ve COCOMO 2000 modelleri incelendi. Bu modellerin özellikleri ve yapılan yayın taramaları sonucunda yazılım projelerinde maliyeti en çok etkileyen faktörler ortaya koyuldu. Yazılımın maliyetini en çok etkileyen faktörler yazılım ürününün büyüklüğü, çalışanların yeterliliği, çalışan kişi sayısı, müşteri beklentilerinin

zamanla değişmesi, kullanılan teknoloji, yazılım geliştirme ortamı, tecrübe, donanım riski, çalışan riski, proje kısıtlarının değişme riski ve zaman kullanım durumu olarak saptanmıştır. Bu belirlenen faktörlerin dışında maliyeti etkileyen birçok faktör ve bunların alt başlıklarını oluşturabilecek faktörler mevcuttur. Fakat bu bütün bileşenleri kullanarak yapay sinir ağında maliyet tahminini gerçekleştirebilmek için çok fazla sayıda örnek setine ihtiyaç duyulmakta. Eğitimin sağlanması da güçleşmektedir. Dolayısıyla bu faktörlerden proje maliyetine en fazla etkisi olanlar seçildi. Yazılım firmalarından daha önce gerçekleştirmiş oldukları yazılım projelerinin maliyetleri, çalışan kişi sayısı, yazılan kod satır sayısı, bozulan (defect) kod satır sayısı, proje büyüklüğüne dair bilgiler toplanıldı. Her projenin yapım tarihi farklı olduğundan elimizdeki maliyetleri nakit akış diyagramları ve dönemin faiz oranlarını kullanarak projelerin her birinin günümüzdeki maliyet değeri hesaplandı. Verilerin bir bölümü eğitim bir bölümü de test verileri olarak iki gruba ayrıldı. Yapay sinir ağı olarak çok katmanlı ileri beslemeli yapay sinir ağı, eğitim algoritması olarak Delta Algoritması seçilmiştir. Eğitim işlemi tamamlandıktan sonra öğrenmeyi tamamlayan ağa test giriş verileri sunularak yazılımın maliyetini tahmin etmesi sağlanmıştır. Elde edilen değerler COCOMO II 2000 kullanılarak elde edilen değerlerle karşılaştırılmıştır.

# 1. YAZILIM PROJE MALİYET TAHMİNİ YÖNTEMLERİ

## 1.1. Bilgisayar Yazılımlarında Ölçme ve Değerlendirme Kriterleri

Yazılım geliştiren her kurumun temel amacı yazılım projelerinin verimini artırmak ve başarılı projeler üretmektir. Bu da ancak yazılım süreçlerinde yapılacak iyileştirme ve gelişmelerle mümkün olabilir. Günümüzde, yazılım sistemleri bankacılıktan otomotiv sanayisine, sağlık bilgi sistemlerinden şirket yönetimine, telekomünikasyon sistemlerinden hava taşımacılığına kadar çok geniş alanlarda kullanılan bilgisayar sistemlerinin çok önemli ve kritik bir parçasını oluşturmaktadır. Yazılım geliştirme ise yazılım sistemlerinin mühendislik prensipleri çerçevesinde tasarımı, üretimi ve işletilmesini hedefler.

Yazılımların giderek karmaşık ve kompleks yapı kazanmasından ölçme işlemini gerçekleştirmek zorlaşmaktadır. Yazılımların kullanıcılar tarafından kolay anlaşılır olması, hazırlanan modüllerin tekrar kullanılabilir olması, bakım yapılabilir olması ve güvenilir olması hedeflenmektedir. Yalnız belirlenen hedeflere net bir ölçüm getirilememektedir. Ölçülemeyen hiçbir şey kontrol altına alınamaz. 2002 yılında The Standish Group tarafından yapılan araştırmaya göre; ABD'de farklı ölçekteki firmalarda yapılan yazılım projeleri analizi sonucu yapılan projelerin sadece % 34'ü başarılı olmuştur. Bu başarısızlığın sebepleri ise belirlenen maliyet ve zaman sınırının üstüne çıkılmasıdır. Müşteri beklentilerinin gereksinim analizi aşamasında tam olarak belirlenememesi, müşteri taleplerinin zaman içerisinde değişiklik göstermesi ve proje sınırının tam olarak belirlenememesi. Müşterinin teknik bilgi yetersizliğinin projenin anlaşılmasını engellemektedir. Yazılımcıların teknik yetersizliği ve kodlama hatası. Geliştirme ortamının doğru seçilmemesi. Projede kullanılan teknik donanım ile çalışacak donanımların uyumsuzluğu proje başarısını kötü yönde etkilemiştir.

Yazılım projelerinin çoğunda baştan maliyet ve zaman tahmininin doğru yapılamamasından kaynaklanan başarısızlık yatmaktadır. Yazılım projelerinde maliyet tahmini daha önce yapılan projeler ile karşılaştırma yapılarak yürütülmektedir. Yazılım süresince yazılımın diğer projelerden bağımsız doğru maliyet tahmini henüz yapılamamaktadır. Bilgisayar yazılımlarında maliyet ve risk tahmini son zamanlarda oldukça önem kazanmıştır. Yazılım maliyet tahmini hem kurumlar hem de devlet için önemli bir problemdir. Yazılım projelerinde zaman, maliyet, gerekli iş gücü gibi tahminlerin yapılabilmesi için öncelikle ölçümlerin yapılması gerekir. Bu ölçülebilir

büyüklükler dolaylı ve direk olarak sınıflandırılabilir. Direk olarak ölçülebilen büyüklükler yazılım maliyeti, ortaya çıkan hata sayısı, yazılımcı gayreti, yazılan kod satır sayısı, yazılımcı iş gücüdür. Diğer ölçülebilen dolaylı büyüklükler ise yazılımın işlevselliği, güvenirliği, bakım ve onarım kolaylığı, yazılımın kalitesi olarak sınıflandırabiliriz.

### 1.2. Satır Sayısı Yöntemi ile Kestirim

Satır sayısı yöntemi yazılım projesinde yazılan kod satır sayımı ile gerçekleştirilir. Yazılım büyüklüğü ölçümünde çok kullanılan bir yöntem olmasına rağmen çok fazla sınırlaması olan bir yöntemdir. Çünkü kod satır sayımı fiziksel olarak yapılabileceği gibi lojik olarak da yazılım diline ve projesine göre gerçekleştirilebilir. Fiziksel olarak bir satır kodu "enter" tuşuna basılıncaya kadar yazılan kodun tamamı gerçekleştirir. Lojik olarak bir satır kod sayımı ise programlama diline bağlı olarak virgül, noktalı virgül veya bir period belirler. Satır sayısı belirleme de karşılaşacağımız üç temel problem vardır. Bunlar satır sayısını belirleyen herhangi bir standardın olmayışı ve tekrar kullanılabilen (reuse code) kodlar için standart bir yaklaşımın mevcut olmamasıdır. Ayrıca bir yazılım projesinde birden fazla programlama dili kullanılabilir. Çoklu dil kullanıldığında da kod satır sayımı için kesin bir standart mevcut değildir. Kod satır sayımındaki bu karmaşıklıkları engellemek için Software Engineering Institute (SEI) tarafından kod sayımı için standart bir form belirlenmiştir.(Park, R.E. 1992) Bakınız Şekil1.2 Bu form daha sonra B. Boehm tarafından gözden geçirilerek COCOMO 2 modelinde kullanılmıştır.

Bu yöntemde, proje tahmin edilen alt birimlerine ayrıştırılır. Parçala ve yönet stratejisi sonucunda ortaya çıkan, üzerinde tahmin yapılması daha kolay olan daha küçük her birim için satır sayıları önerilir. Bu kestirimler yapılırken de en küçük, en olası, ve en büyük ihtimaller belirlenip bunlarla bir ortalama işlemi yapılabilir. Bir birim için tahmin edilecek en küçük satır sayısına k, en olası satır sayısı tahminine o, ve en büyük tahmin değerini de b olarak adlandırılırsa satır sayısı kestirimi aşağıdaki gibi hesaplanır.(İrfan Macit)

Satır sayısı kestirimi: (k + 4o + b) / 6 şeklinde hesaplanabilir.

Yazılım oluşturulması sırasında temel değerlendirme kriteri olarak satır kaynak kod sayısı göz önüne alınır. Satır kaynak kod sayısı (LOC: Lines of Codes) bazı durumlarda yetersiz olabilir. Satır kaynak kod yerine işlevsel gösterge ölçüm birimini kullanmak daha yararlı ve sonucu alma açısından önemli olmaktadır. Sayısal olarak

yazılımın değerlendirilmesini sağlayan bu yöntemde istenilen bileşenleri için değişik ağırlıklar verilerek yazılıma katkısı araştırılabilir. (Albrecth,1979) Bu tip değerlendirme sistemlerinde ağırlık ve sonuç göstergeleri yazılım yöneticisi tarafından değişik şekillerde bir araya getirilerek sayısal göstergeleri oluşturulabilir. Bu göstergelerde bulunan bileşenler yer değiştirilerek gösterge üzerinde anlamlı ifadeler çıkarılabilir. Kullanılan bilgisayar programlama dillerinden birisinin diğerinin yerine kullanılması değerlendirilebilir. Fortran bilgisayar programlama dili ile yazılan bilgisayar program yerine C++ bilgisayar programlama dili kullanılarak bazı kodların fazladan yazılması engellenebilir. Nesne yönelimli dillerden birisi ile oluşturulan bilgisayar programı yerine yapısal programlama dillerinden birisi ile yapılan programlarda daha fazla kod yazmak maliyetleri ve hata oranını arttırabilmektedir. Bunun için işlevsel göstergelerin neler olduğunu veya nasıl hesaplanması gerektiğinin iyi belirlenmesi gerekmektedir. Belirleme işlemleri sırasında eş değer bilgisayar yazılım dilleri tablosu oluşturularak hangi programlama dilinin temel fonksiyonlardan birinin kaç satır kod ile yazılabildiğinin ortaya konulması gerekir. Bunun için temel olarak belirlenen matematiksel veya mantıksal birkaç genel kabul görmüş algoritma bilgisayar programlama dili ile programlanır ve kaç satır kod ile oluştuğu belirlenir. Bu bize bir fonksiyon veya alt programın hangi programlama dilinde kaç satır koda (KLOC) karşılık geldiği belirlenmiş olur. Satır kod sayısı işlemine programlama dillerinin işlevselliğini de katacak olursak yapılması gereken işlevsel göstergenin satır sayısı (KLOC)/ işlevsel gösterge oranının bulunması gerekecektir. Bu oransal değer bize satır kod sayısının ne kadar işlevsel olduğu hakkında oransal bilgi verecektir. Oransal değer dönüşüm tablosu oluşturmak işlemleri daha kolaylaştıracak ve yazılım kodları konusunda daha açıklayıcı bilgi edinmemizi sağlayacaktır.

## Definition Checklist for Source Statement Counts

Definition name: _____ Date: _____

_____ Originator: _____

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Measurement unit:** | | **Physical source lines** ☐ **Logical source statements** ☐ | | | | | |
| **Statement type** | **Definition** ☐ | **Data array** ☐ | | | | **Includes** | **Excludes** |
| *When a line or statement contains more than one type, classify it as the type with the highest precedence.* | | | | | | | |
| 1 Executable | | | **Order of precedence ->** | 1 | | | |
| 2 Nonexecutable | | | | | | | |
| 3 Declarations | | | | 2 | | | |
| 4 Compiler directives | | | | 3 | | | |
| 5 Comments | | | | | | | |
| 6 On their own lines | | | | 4 | | | |
| 7 On lines with source code | | | | 5 | | | |
| 8 Banners and nonblank spacers | | | | 6 | | | |
| 9 Blank (empty) comments | | | | 7 | | | |
| 10 Blank lines | | | | 8 | | | |
| 11 | | | | | | | |
| 12 | | | | | | | |
| **How produced** | **Definition** ☐ | **Data array** ☐ | | | | **Includes** | **Excludes** |
| 1 Programmed | | | | | | | |
| 2 Generated with source code generators | | | | | | | |
| 3 Converted with automated translators | | | | | | | |
| 4 Copied or reused without change | | | | | | | |
| 5 Modified | | | | | | | |
| 6 Removed | | | | | | | |
| 7 | | | | | | | |
| 8 | | | | | | | |
| **Origin** | **Definition** ☐ | **Data array** ☐ | | | | **Includes** | **Excludes** |
| 1 New work: no prior existence | | | | | | | |
| 2 Prior work: taken or adapted from | | | | | | | |
| 3 A previous version, build, or release | | | | | | | |
| 4 Commercial, off-the-shelf software (COTS), other than libraries | | | | | | | |
| 5 Government furnished software (GFS), other than reuse libraries | | | | | | | |
| 6 Another product | | | | | | | |
| 7 A vendor-supplied language support library (unmodified) | | | | | | | |
| 8 A vendor-supplied operating system or utility (unmodified) | | | | | | | |
| 9 A local or modified language support library or operating system | | | | | | | |
| 10 Other commercial library | | | | | | | |
| 11 A reuse library (software designed for reuse) | | | | | | | |
| 12 Other software component or library | | | | | | | |
| 13 | | | | | | | |
| 14 | | | | | | | |
| **Usage** | **Definition** ☐ | **Data array** ☐ | | | | **Includes** | **Excludes** |
| 1 In or as part of the primary product | | | | | | | |
| 2 External to or in support of the primary product | | | | | | | |
| 3 | | | | | | | |

Definition name: _____

| Delivery | Definition ☐ | Data array ☐ | Includes | Excludes |
|---|---|---|---|---|
| 1 Delivered | | | | |
| 2 Delivered as source | | | | |
| 3 Delivered in compiled or executable form, but not as source | | | | |
| 4 Not delivered | | | | |
| 5 Under configuration control | | | | |
| 6 Not under configuration control | | | | |
| 7 | | | | |

| Functionality | Definition ☐ | Data array ☐ | Includes | Excludes |
|---|---|---|---|---|
| 1 Operative | | | | |
| 2 Inoperative (dead, bypassed, unused, unreferenced, or unaccessed) | | | | |
| 3 Functional (intentional dead code, reactivated for special purposes) | | | | |
| 4 Nonfunctional (unintentionally present) | | | | |
| 5 | | | | |
| 6 | | | | |

| Replications | Definition ☐ | Data array ☐ | Includes | Excludes |
|---|---|---|---|---|
| 1 Master source statements (originals) | | | | |
| 2 Physical replicates of master statements, stored in the master code | | | | |
| 3 Copies inserted, instantiated, or expanded when compiling or linking | | | | |
| 4 Postproduction replicates—as in distributed, redundant, or reparameterized systems | | | | |
| 5 | | | | |

| Development status | Definition ☐ | Data array ☐ | Includes | Excludes |
|---|---|---|---|---|
| *Each statement has one and only one status, usually that of its parent unit.* | | | | |
| 1 Estimated or planned | | | | |
| 2 Designed | | | | |
| 3 Coded | | | | |
| 4 Unit tests completed | | | | |
| 5 Integrated into components | | | | |
| 6 Test readiness review completed | | | | |
| 7 Software (CSCI) tests completed | | | | |
| 8 System tests completed | | | | |
| 9 | | | | |
| 10 | | | | |
| 11 | | | | |

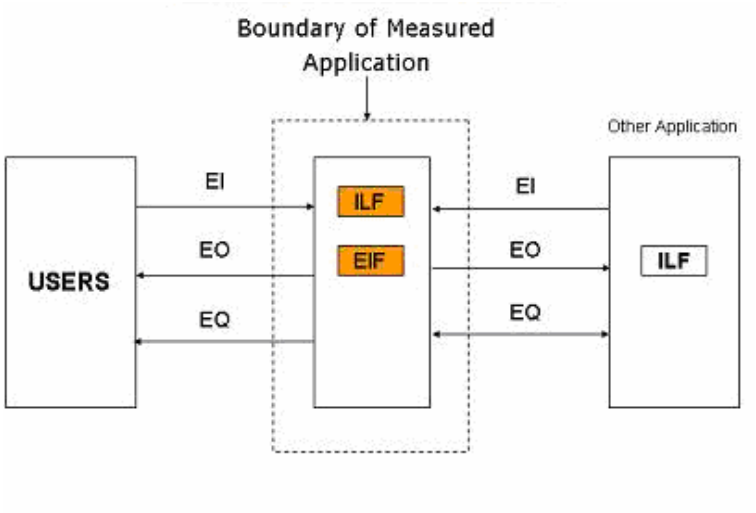
| Language | Definition ☐ | Data array ☐ | Includes | Excludes |
|---|---|---|---|---|
| *List each source language on a separate line.* | | | | |
| 1 | | | | |
| 2 Job control languages | | | | |
| 3 | | | | |
| 4 Assembly languages | | | | |
| 5 | | | | |
| 6 Third generation languages | | | | |
| 7 | | | | |
| 8 Fourth generation languages | | | | |
| 9 | | | | |
| 10 Microcode | | | | |
| 11 | | | | |

Şekil: 1.2. Satır sayımı için belirlenen Checklist Tanımı

### 1.3. Fonksiyon Noktası Yöntemi ile Kestirim

Dolaylı ölçümde, fonksiyon noktası (Function Point FP) adı verilen bir ölçek kullanılmaktadır. Bu ölçeğin hesaplanması için kullanılan yönteme FPA (Function point analaysis) adı verilir.(Dreger, J.B. 1989) Yazılım projesi ile ilgili sisteme gelen girdiler, çıktılar gibi bilgiler biliniyorsa bunlar kullanılarak geliştirilecek sisteme ait

yeni bilgiler elde edilebilir. Satır sayısı tekniğin tersine bu yöntemde bir yazılım birimi için doğrudan büyüklük tahmini yapma zorluğu yoktur. Bu yöntemle elde edilen veriden yazılımın büyüklüğü hakkında bir bilgiye ulaşılabilir. Elde edilen veriden SLOC hesaplanabilir. Fonksiyon noktası IBM araştırmacısı olan Allan Albert tarafından geliştirilmiştir. Fonksiyon noktasını hesaplamak için öncelikle veri fonksiyonelliğini ve transaction fonksiyonelliğini belirlemek gerekir. Veri fonksiyonelliği için ILF ( Internal Logical Files) ve EIF (External Interface Files) belirlenmeli. Transaction fonksiyonellik için EI (External Input), EO (External Output), EQ (ExternalQueries) belirlenmeli.(IFPUG, 2000)



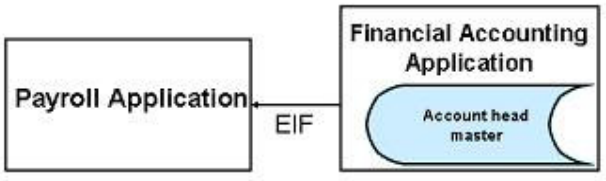

Şekil: 1.3. Fonksiyon noktası yönteminin modeli

EI kullanıcı tarafından bilgisayara gönderilen veriyi temsil eder. EO bilgisayar tarafından son kullanıcı için işlenen veriyi simgeler. ILF Kullanıcı tarafından lojik olarak görülebilen uygulama için depolanan bilgi için kullanılır.

Fonksiyon noktasını hesaplamak için uygulanacak adımlar:

1) Uygulamanın sınırları (application boundary) belirlenmeli. Yani kullanıcı gözüyle sistemin geneli belirlenmeli.

2) ILF ve EIF veri fonksiyonelliği belirlenmeli. Bunlarla birlikte RET(record element type) ve DET(data element type) ortaya koyulmalı.




3) EI, EO, ve EQ olarak adlandırılan transaction fonksiyoneller belirlenmeli. Yani sisteme dışarıdan kullanıcı tarafından girilen veriler. Sitemin işlemleri gerçekleştirdikten sonar dışarıya verdiği veriler ortaya koyulmalı. Yukarıdaki bilgiler elde edildikten sonra UFP(Unadjusted Function Points) değeri hesaplanabilir.

4)Tablolar yardımı ile UFP değeri EI, EO, EQ, EIF ve ILF değerlerinin toplanması ile elde edilir.UFP değerini elde ettikten sonra VAF (Value added Factor) değeri hesaplanır. TDI (Total Degree of Influence )değeri ise sistem karakteristiklerinin toplamına eşittir.

VAF = 0.65 + (0.01 * TDI)

5) Son olarak Fonksiyon noktası elde edilen UFP değeri ile VAF değerinin çarpımına eşittir

FP = UFP * VAF

Tablo: 1.1 External Inputs (EI)

| File Type Referenced (FTR) | ">Data Elements (DET) | | |
|---|---|---|---|
| | 01.Nis | May.15 | Greater than 15 |
| Less than 2 | Low (3) | Low (3) | Average (4) |
| 2 | Low (3) | Average (4) | High (6) |
| Greater than 2 | Average (4) | High (6) | High (6) |

Tablo: 1.2. External Outputs (EO)

| ">File Type Referenced (FTR) | ">Data Elements (DET) | | |
|---|---|---|---|
| | 01.May | Haz.19 | Greater than 19 |
| Less than 2 | Low (4) | Low (4) | Average (5) |
| 2 or 3 | Low (4) | Average (5) | High (7) |
| Greater than 3 | High (7) | High (7) | High (7) |

Tablo: 1.3. External Inquiry (EQ)

| ">File Type Referenced (FTR) | ">Data Elements (DET) | | |
|---|---|---|---|
| | 01.May | Haz.19 | Greater than 19 |
| Less than 2 | Low (3) | Low (3) | Average (4) |
| 2 or 3 | Low (3) | Average (4) | High (6) |
| Greater than 3 | Average (4) | High (6) | High (6) |

Tablo:1.4. Internal Logical File (ILF)

| ">Record Element Types (RET) | ">Data Elements (DET) | | |
|---|---|---|---|
| | Oca.19 | 20-50 | 51 or More |
| 1 RET | Low (7) | Low (7) | Average (10) |
| 2 to 5 RET | Low (7) | Average (10) | High (15) |
| 6 or more RET | Average (10) | High (15) | High (15) |

Tablo: 1.5. External Interface File (EIF)

| ">Record Element Types (RET) | ">Data Elements (DET) | | |
|---|---|---|---|
| | Oca.19 | 20-50 | 51 or More |
| 1 RET | Low (5) | Low (5) | Average (7) |
| 2 to 5 RET | Low (5) | Average (7) | High (10) |
| 6 or more RET | Average (7) | High (10) | High (10) |

### 1.4. COCOMO 81 Modeli

COCOMO (Constructive Costing Model) 1981 yılında Barry Boehm tarafından geliştirilmiş bir modeldir. Yapılacak hesapların kapsamları açısından üç değişik modelden oluşur. Basit model, orta model ve detaylı modeller. Ayrıca bu modellerde kullanılacak problemler organik, yarı ayrık ve gömülü sınıfları adı altında toplanmıştır. Organik problemler için küçük boyuttaki programcı takımları, bildikleri ortamlarda iyi anlaşılmış uygulamaları geliştirirler. İletişim kamburu azdır ve elemanlar işlerini hızlı bir şekilde yapabilecek durumdadırlar. Yarı ayrık problemlerde ise ekipte tecrübeli ve tecrübesiz elemanlar bulunabilir. İlgili sistemler konusunda deneyimleri sınırlı olabilir ve geliştirilen sistemin her bir sürecini bilmeyebilirler. Gömülü problemlerde ise

geliştirilecek yazılım, sistemin donanım, kurallar, işletim süreçleri veya yazılım gibi diğer bileşenleri ile çok kuvvetli bağlantılar oluşturur. Gereksinim değişiklikleri ile problemleri halletmek olanaksızlaşmıştır. İhtiyaç belirtiminin geçerlilik irdelemesi çok pahalıdır. Elemanların her şeyi bilme olasılığı oldukça düşüktür.

Tablo: 1.4.1 COCOMO Modelleri ve Parametreleri

| **Geliştirme Modu** | **A** | **B** | **c** | **D** |
|---|---|---|---|---|
| Organik | 3.2 | 1.05 | 2.5 | 0.38 |
| Yarı Ayrık | 3.0 | 1.12 | 2.5 | 0.35 |
| Gömülü | 2.8 | 1.20 | 2.5 | 0.32 |

| **Geliştirme Modu** | **Proje Karakteristik Özellikleri** | | | |
|---|---|---|---|---|
| | *Boyut* | *Keşif* | *Deadline/constraints* | *Geliştirme Ortamı* |
| Organik | Küçük | Küçük | Küçük | Kararlı |
| Yarı Ayrık | Orta | Orta | Orta | Orta |
| Gömülü | Geniş | Geniş | Geniş | Komplex donanım/ müşteri arayüzü |

COCOMO bu model ve problem sınıfı saptamalarından sonra ortaya çıkan formüllerle tahmin hesaplama yolunu önerir. Bu model gerekli olan iş gücünü yazılımın kod satır sayısı(SLOC) ve maliyet etkenlerini (ürün, donanım, çalışanlar ve proje ilişkileri) kullanarak hesaplar.

Basit COCOMO modelinin hesaplama adımları:

1) Öncelikle projenin geliştirilme modu belirlenir. Proje organik, yarı ayrık veya gömülü olabilir. (B. Barry , 1995)
2) Effort Adjustment Factor (EAF) değeri tabloda verilen "cost driver"ların çarpımı ile hesaplanır.

EAF = (cost driver1 * cost driver2 * cost driver3*……. cost driver15)

3) Gerekli olan iş gücü organik yarı ayrık ve gömülü sistemlere göre ayrı ayrı hesaplanır. (KDSI: kilo DSI (Delivered Source Instruction))

Organik Mod: Gerekli iş gücü = EAF* 3.2 $(KDSI)^{1,05}$

Yarı Ayrık Mod : Gerekli iş gücü = EAF* 3.2 $(KDSI)^{1,12}$

Gömülü Mod: Gerekli iş gücü = EAF* 3.2 $(KDSI)^{1,20}$

4) Geliştirmek için gerekli olan zaman (Time for Devellopment (TDEV)) hesaplaması ay bazında gerçekleştirilir.

TDEV = 2.5 * Gerekli İş Gücü$^{0,38}$

5) Gerekli olan ortalama çalışan sayısı ise

Ortalama Çalışan Sayısı = Gerekli İş Gücü / TDEV

olarak hesaplanır.

Tablo: 1.4.2. COCOMO Cost Driver Ranking Tablosu

| **Cost Driver** | **Very Low** | **Low** | **Nominal** | **High** | **Very High** | **Extra High** |
|---|---|---|---|---|---|---|
| ACAP Analyst Capability | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | - |
| AEXP Applications Experience | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | - |
| CPLX Product Complexity | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| DATA Database Size | - | 0.94 | 1.00 | 1.08 | 1.16 | - |
| LEXP Language Experience | 1.14 | 1.07 | 1.00 | 0.95 | - | - |
| MODP Modern Programming Practices | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | - |
| PCAP Programmer Capability | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | - |
| RELY Required Software Reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | - |
| SCED Required Development Schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | - |
| STOR Main Storage Constraint | - | - | 1.00 | 1.06 | 1.21 | 1.56 |
| TIME Execution Time Constraint | - | - | 1.00 | 1.11 | 1.30 | 1.66 |
| TOOL Use of Software Tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | - |
| TURN Computer Turnaround Time | - | 0.87 | 1.00 | 1.07 | 1.15 | - |
| VEXP Virtual Machine Experience | 1.21 | 1.10 | 1.00 | 0.90 | - | - |
| VIRT Virtual Machine Volatility | - | 0.87 | 1.00 | 1.15 | 1.30 | - |

## 1.5. COCOMO II

COCOMO II modeli Bary Boehm tarafından ilk olarak Software Engineering (1981) kitabında yayınlanan COCOMO 81 modelinin geliştirilmiş halidir. En son hali ise COCOMO II 2000 dir.

Detaylı COCOMO modeli ise projenin evrelerine bağlı olarak süreç içinde değişiklikleri hesaba katarak arada bir kestirim hesaplamasını önerir. Bu modelde zamana bağlılık temel değişikliktir. Projenin farklı evrelerinde çaba yoğunluğu ve yapılacak işin karmaşıklığı değişecektir. Benzer bir anlayış, yazılım eğrisi denilen bir sonucu yansıtan Putnam modelinde de kendisini gösterir. Evrelere bağlılığın bir sonucu olarak Raleigh adam-ay eğrileri ortaya çıkar. Bu eğriler, proje başlangıcındaki az iş gücü ile gereksinimlerin ilk çabalarını düşük düzeylerle temsil eder. Hızla tırmanan eğri, geliştirmenin analiz, tasarım ve uygulama gibi evrelerinde yükseklerdedir. Daha sonra geliştirme sonrası faaliyet azalır. İleride bakım ve onarım yine bazı geçici yükselmeler yapabilir.

COCOMO II modeli üç bölümde (stage) de ele alınabilir. Birinci bölüm uygulama için gerekli gücün tahmini ve prototip belirleme tahminini destekler. Bu bölümde relative size tahmini yüksektir. İkinci bölümde cost driver'ları hakkında bilgi yeterince bulunmuyorsa ve projenin tasarım aşamasındaysa destek sağlıyor. Üçüncü bölüm projenin Post Architecture stage için tahmin desteği sağlar.

## 1.6. Yazılım Kestiriminde İzlenecek Yol

Birden fazla tahmin tekniği kullanılmalıdır. Bunun yanında, herhangi bir teknik için veri toplarken uzman görüşlerden yararlanmalı ve bireylerin özellikleri de düşünülerek veriler değerlendirilmelidir. Tahminlerin doğruluğu, önceden edinilen bilgilerin düzenli tutulmasına da bağlıdır. Önce sistem tanımındaki veriler kullanılarak bir İşlev Puanı hesabı yapılabilir ve sonuçlar satır sayısına çevrilebilir. Daha önceki verimlilik ve pahalılık gibi bilgiler ışığında bu satır sayısından maliyet, çaba ve süre tahmini yapılır. İkinci bir yol olarak elde edilmiş satır sayısından hareket ederek COCOMO modeli aracılığı ile çaba ve süre hesaplanabilir. Daha sonra diğer bir yol olarak da satır sayısı yöntemine başvurulabilir. Bu durumda sistemi üst seviyedeki bileşenlerine ayırabilmek gerekir. Bu alt sistemler için satır sayısı tahminlerinde bulunulur ve ayrı ayrı çaba, süre gibi hesaplar yapılır. Alt sistemlere ait bilgiler toplanarak sistem için izlediğimiz üçüncü kestirimler elde edilmiş olur. Ayrıca alt sistemler için de önce İşlev Puanı yöntemi uygulanabilir. Sözü geçen değişik

kestirimler birbiriyle karşılaştırılarak arada çok büyük farklar gözlendiği takdirde hangi yöntemde hatalı varsayımlarımız olduğu ortaya çıkarılmaya çalışılır. Büyük fark olduğunda hangi kestirimin hatalı olabileceği yönünde bir ipucu olmak üzere bir veya birkaç diğer yöntem sonuçları da kullanılabilir. Belirli bir kanaate ulaşılınca da bu durum kabul edilir ve bir süre olduğu gibi bırakılır. İlk kestirmeler en yönlendirici olanlardır. Bilgi azlığı nedeniyle en hatalı olanlardır. Geliştirme ilerledikçe tahmin hesapları arada bir yapılabilir. Projenin sonlarında ise daha önce yapılan kestirimlerin hatalarının en çok hangi parametrelerden kaynaklandığı bulunmaya çalışılır: artık elde kesin bilgiler vardır, bu sonuç bilgileri, kestirim formüllerine geri konarak bu formüllerdeki sabit katsayılar, ayarlama katsayıları gibi değerlerde oynama yapılarak firma için uyarlanmış daha geçerli formülleri ortaya çıkarmış oluruz. Bu çalışmalar, ilerde geliştirilecek projeler için daha doğru kestirim yapmaya yöneliktir. Ayrıca, ölçme bilgilerinin de tutulmasında düzenlemeler ve düzeltmeler de benzeri tedbirlerdir.

# 2. YAPAY SİNİR AĞLARI

## 2.1. Yapay Sinir Ağlarının Genel Tanımı

Teknolojik gelişmelerin hızlı bir şekilde artması, insanoğlunun kendini tanımaya yönelik çalışmalar yapmasına neden olmuştur. İnsanın en önemli özellikleri olan düşünebilme ve öğrenebilme yetenekleri incelenerek, bu özellikleri bilgisayar ortamına taşıma girişiminde bulunulmuştur. İnsanın düşünme yapısını anlamak ve bunun benzerini ortaya çıkaracak bilgisayar işlemlerini geliştirmeye çalışmak olarak tanımlanan yapay zekâ, aslına programlanmış bilgisayarlara düşünme yeteneği sağlama girişimidir (DPT, 2005).

İnsanlar doğuştan belirli bir zekâya sahiptirler. Akıl ve zekâ terimleri genellikle birbirine karıştırılır. Akıl düşünme, kavrama, anlama, önlem alma yeteneğidir. Zekâ ise belirli bir konuda çalışılarak öğretilerek, eğitilerek, edinilen bilgi ve birikimlerle, deneyimlere dayalı becerilerle geliştirilebilir. Daha önce yaşanmamış bir durum veya ani yaşanan olaylar karşısında analiz yapabilme, öğrenme, anlama, uyum sağlama gibi olgular zekâ aracılığıyla gerçekleştirilir. Zekânın yazılımla veya tümleşik yongalar ile taklit edilebilmesine "Yapay Zeka" denir (Elmas,2003, S.21).

İnsanı taklit etmeye yönelik olan yapay zekâ çalışmaları 1950 yılından beri devam etmektedir. İnsan beyni çok karmaşık bir yapıya sahiptir. Beynin yoğun bağlantılı ve karışık yapısının sadece beyine özel bir özellik olduğu bilinmektedir. Başka hiçbir yerde veya dijital bilgisayarda bulunmayan bu yapıya yakınsamak günümüz teknolojilerine bile çok uzaktır. Yapay sinir ağlarını oluşturmak için kullanılan yapay nöronlar, beyindekine kıyasla oldukça basit kalırlar. Basit olmalarına rağmen genel yapı olarak tutarlıdırlar. Yapay zeka çalışmalarına destek veren bilimlerden bir tanesi de yapay sinir ağı teknolojisidir. Yapay zeka alanının bir alt dalını oluşturan yapay sinir ağı teknolojisi öğrenebilen sistemlerin temelini oluşturmaktadır.

Günümüzde altmışın üzerinde yapay zeka teknolojisinin varlığından bahsedilmektedir. Çünkü karşılaşılan problemler ve olaylar sürekli değişmektedir. Belirli bir olay karşısında farklı bölge insanları farklı düşünce yapısı ile çözmeye çalışır. Bilgisayarların insanların karar verme ve problem çözme mekanizmalarını taklit etmesinin sağlanması da dolayısıyla farklı teknolojilerin doğmasına neden olmuştur. Bu teknolojilerin birçoğu laboratuar çalışması düzeyindedir. Bunlardan en

yaygın olarak kullanılanlar ise uzman sistemler, makine öğrenmesi ve yapay sinir ağları, genetik algoritmalar, bulanık önermeler mantığı ve zeki etmenlerdir.

Yapay sinir ağı teknolojisi insanlığın doğayı taklit etme ve araştırma çabalarının en son ürünüdür. Beynin üstün özellikleri bilim adamları tarafından çalışılarak beynin nörofiziksel yapısından matematiksel model ortaya çıkartılmıştır. Beynin bütün davranışlarını tam olarak modelleyebilmek için fiziksel bileşenlerinin doğru olarak modellenmesi gerektiği düşüncesi ile çeşitli yapay hücre ve ağ modelleri geliştirilmiştir. Böylece yapay sinir ağları olarak isimlendirilen ve günümüz bilgisayarlarının algoritmik hesaplama yönteminden farklı bir bilim dalı ortaya çıkmıştır.

Genel anlamda yapay sinir ağları beynin bir işlevi yerine getirme yöntemini modellemek için tasarlanan bir sistem olarak tanımlanabilir. Yapay sinir ağları, yapay sinir hücrelerinin birbirleri ile çeşitli şekillerde bağlanmasından oluşur ve genellikle katmanlar şeklinde düzenlenir.

## 2.2. Yapay Sinir Ağlarının Genel Özellikleri

Yapay sinir ağlarının karakteristiği yapılacak olan uygulama ile belirlenmektedir. Dolayısıyla Yapay sinir ağlarının çok sayıda farklı çeşitleri vardır. Bu farklılığın kaynağı olarak ağın mimarisi, kullanılan öğrenme yöntemi, bağlantı yapısı şeklinde özetleyebiliriz. Yapay sinir ağlarının genel özellikleri bütün modelleri kapsayacak şekilde ele alınacaktır(Öztemel, 2003, S.31).

### 2.2.1. Yapay Sinir Ağları Makine Öğrenmesi Gerçekleştirirler

Genel olarak yapay sinir ağları, insan beyninin sinir ağlarını taklit eden bilgisayar programlarıdır. Yapay sinir ağları bir anlamda paralel bilgi işleme sistemi olarak düşünülebilir. Yapay sinir ağlarına bilgiler yapılacak uygulamaya ait örnekler kullanılarak eğitim süresince verilir. Verilen örnekler üzerine yapay sinir ağı genelleme yapar. Bu genelleme sonuçlarını kullanarak ortaya çıkacak ya da o ana kadar karşılaşılmamış olaylara çözümler üretir.

Yapay sinir ağları, ilgilendiği problemi öğrendikten sonra eğitim sırasında karşılaşmadığı test örnekleri içinde arzu edilen tepkiyi üretebilir. Örneğin karakter tanıma amacıyla eğitilmiş bir yapay sinir ağı, bozuk karakter girişlerinde de doğru karakteri verebilir. Sistemin eğitilmiş yapay sinir ağı modeli eğitim süresinde verilmeyen giriş sinyalleri için de sistemle aynı davranışı gösterebilir.

Yapay sinir ağları makine öğrenmesi gerçekleştirirler. Yapay sinir ağlarının temel işlevi bilgisayarların öğrenmesin sağlamaktır. Olayları öğrenerek benzer olaylar karşısında benzer kararlar vermeye çalışırlar. Yapay sinir ağı metedolojisi veriden öğrenebilme, genelleme yapabilme ve sınırsız sayıda değişkenle çalışabilme özelliğinden diğer alanlarda olduğu gibi öngörü modellemesinde de yaygın olarak kullanılmaktadır.

### 2.2.2. Yapay Sinir Ağları Örnekleri Kullanarak Öğrenirler

Yapay sinir ağlarının en önemli özelliği öğrenme özelliğidir. Yapay sinir ağları insan beyninden esinlenerek ortaya koyulmuştur. İnsan beyni gibi eğitme ve başlangıç tecrübesi sayesinde veriyi kullanarak öğrenme işlemini gerçekleştirir. Bu özelliği sayesinde geleneksel teknikler için çok karmaşık kalan problemlere çözüm sağlayabilmektedir. Ayrıca, insanların kolayca yapabildiği ama geleneksel metotların uygulanamadığı basit işlemler için de oldukça uygundur.

Yapay sinir ağlarının bir durum karşısında sonuç çıkarabilmesi için ilgili olay üzerine örneklerin belirlenmesi gerekir. Örnekler olayın bütününü yansıtacak şekilde olmalı. Ağ adaptif öğrenme yöntemi özelliği ile örnekleri kullanarak öğrenme işlemini gerçekleştirir. Adaptif öğrenmede örnek bulunamıyorsa ağ öğrenmeyi gerçekleştiremez. Aynı şekilde alınan örnekler olayın genelini kapsamıyorsa ağ istenen performansı sergileyemez. Bu durum ağın problemli olduğundan çok olayın ağa iyi gösterilemediğinden kaynaklanır. Dolayısıyla seçilecek örnekler özenle belirlenmeli ve olayın genelini kapsaması sağlanmalı.

Yapay sinir ağının örnekler kullanılarak öğretilebileceğini örnek bir olayla inceleyelim. Doktor hastasına sorular sorar ve aldığı cevaplara göre hastayı teşhis eder. Sorulan sorular ve koyulan teşhis bir örnek olay olarak nitelendirilir. Doktorun belirli bir zaman içerisinde gördüğü hastalardan aldığı hasta hikâyelerinden ve teşhislerden oluşan örnek olay kümesi kullanılarak ağın benzer olaylar karşısında hastalığı teşhis etmesi sağlanır. Örnekler gerçekleşmiş olaylardan alınmalı. Alınan örnekler olayın bütününü kapsadığı takdirde ağ olayı bütün yönleri ile ele alarak en uygun sonucu ortaya koyar.

### 2.2.4. Görülmemiş Örnekler Hakkında Bilgi Üretebilirler

Yapay sinir ağına belirli bir durum hakkında gerçekleşmiş olaylar verilerek ağın çıkarım yapması sağlandığı gibi örneklerden genelleme yaparak görmediği örnekler hakkında da bilgiler üretebilmesi sağlanır.

### 2.2.5. Algılamaya Yönelik Olaylarda Kullanılabilirler

Ağlar daha çok algılamaya yönelik bilgileri işlemede kullanılırlar. Bu konuda başarılı oldukları yapılan uygulamalarda görülmektedir. Bilgiye dayalı çözümlerde uzman sistemler kullanılmaktadır. Bazı durumlarda yapay sinir ağı ve uzman sistemler birleştirilerek daha etkin sistemler kurulabilir.

### 2.2.6. İlişkilendirme ve Sınıflandırma Yapabilirler

Yapay sinir ağları insanlar tarafından gerçekleştirilmiş örnekleri kullanarak olayları öğrenebilen, çevreden gelen olaylara karşı nasıl tepkiler üretileceğini belirleyebilen bilgisayar sistemleridir. İnsan beyninin fonksiyonel özelliklerine benzer şekilde öğrenme, ilişkilendirme, sınıflandırma, genelleme, özellik belirleme ve optimizasyon gibi konularda başarılı bir şekilde uygulamaktadırlar. Örneklerden oluşturdukları deneyimlerden yararlanarak benzer olaylar hakkında karar verirler.

Genel olarak ağların çoğunun amacı kendisine örnekler halinde verilen örüntülerin kendisi veya diğerleri ile ilişkilendirmesidir. Diğer bir amaç ise sınıflandırma yapmaktır. Verilen örneklerin kümelendirilmesi ve belirli sınıflara ayrıştırılarak daha sonra gelen bir örneğin hangi sınıfa gireceğine karar vermesi hedeflenir.

### 2.2.7. Örüntü tamamlama Gerçekleştirebilirler

Eğitilmiş ağa bazı durumlarda eksik örüntü veya şekil verilerek ağın bu eksik bilgileri tamamlaması beklenir. Örneğin fotoğrafın yarısı ağa verilerek ağın fotoğrafı tamamlayıp kime ait olduğu söylemesi beklenir. Bu tür durumlarda ağ eksik bilgileri tamamlayarak örüntü tamamlama işlevini başarıyla gerçekleştirir.

### 2.2.8. Kendi kendini Organize Etme ve Öğrenebilme Yetenekleri Vardır

Yapay sinir ağlarının öğrenebilmesi için örneklerin belirlenmesi, bu örneklerin ağa gösterilerek istenen çıktılara göre ağın eğitilmesi gerekmektedir. Ağın başarısı, seçilen örnekler ile doğru orantılıdır, ağa olay bütün yönleri ile gösterilemezse ağ yanlış çıktılar üretebilir. Yapay sinir ağları eğitimleri sırasında kendilerine verilen

örneklerden genellemeler çıkarırlar ve bu genellemeler ile yeni örnekler hakkında bilgi üretebilirler. Yapay sinir ağlar online olarak öğrenebilirler ve kendi kendilerini eğitebilirler.

### 2.2.9. Eksik Bilgi ile Çalışabilmektedirler

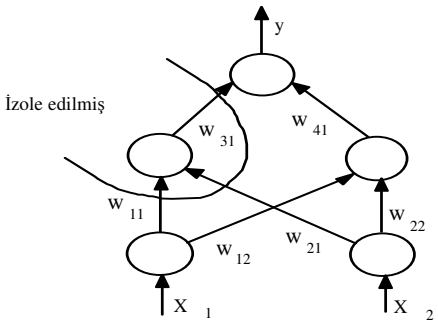
Geleneksel sistemlerin aksine Yapay sinir ağları eğitildikten sonra veriler eksik bilgi içerse dahi, çıktı üretebilirler. Bu durum bir performans kaybı yaratmaz, performans kaybı eksik bilginin önemine bağlıdır. Burada bilginin önem derecelesi eğitim sırasında öğrenilir.

### 2.2.10. Hata Toleransına Sahiptirler

Klasik hesaplama sistemleri çok az bir zarardan bile etkilenirler. Verilerde eğer bir eksiklik söz konusu olursa, klasik yöntemler çalışmazlar. Yapay sinir ağları için durum farklıdır. Bu farklılık yapay sinir ağlarının hata toleranslı olmasıdır. İyi eğitilmiş ve genelleme kapasitesi yüksek bir sinir ağı kendisine sunulan veriler eksik bile olsa karar verme işlemine devam eder. Aynı şekilde yapay sinir ağları üzerinde birtakım problemler ve bozukluklar olabilir.

İşlem elemanlarının az da olsa zarar görmesi sistemin bütününü etkiler. Yapay sinir ağları paralel dağılmış parametreli bir sistem olduğundan her bir işlem elemanı izole edilmiş bir ada olarak düşünülebilir. Şekil'de çok katmanlı perceptron (MLP) için bu durum gösterilmiştir.

Verilerdeki eksiklik veya yapay sinir ağlarındaki yapısal bozukluktan dolayı daha çok işlem elemanının zarar görmesi ile sistemin davranışı biraz daha değişir. Performans düşer ama sistem hiçbir zaman durma noktasına gelmez. YSA sistemlerinin hata toleranslı olmasının nedeni bilginin tek bir yerde saklanmayıp, sisteme dağıtılmasıdır. Bu özellik sistemin durmasının önemli bir zarara neden olacağı uygulamalarda önem kazanır.




Şekil: 2.3.1 Çok katmanlı perceptron'da bir işlem elemanının izole edilmiş hali

### 2.2.11. Dereceli Bozulma Gösterirler

Bir ağ, zaman içerisinde yavaş ve göreceli bir bozulmaya uğrar. Ağlar problemin ortaya çıktığı anda hemen bozulmazlar. Yapay sinir ağlarının hatalara karşı toleranslı olmaları bozulmalarının da dereceli olmasına neden olmaktadır. Bir ağ zaman içinde yavaş ve zarif bir şekilde bozulur. Bu bozulma girilen örnek olay küme verilerinin eksikliğinden ya da zamanla kayba uğrayan hücrelerden meydana gelir. Ağda herhangi bir problem ortaya çıktığında hücrelerde ani bozulma görülmez. Hücrelerde bozulma zamanla ve yavaş yavaş görülür.

### 2.2.12. Dağıtık Belleğe Sahiptirler

Yapay sinir ağlarında bilgi ağa dağılmış bir şekilde tutulur. Hücrelerin bağlantı ve ağırlık dereceleri, ağın bilgisini gösterir. Bu nedenle tek bir bağlantının kendi başına anlamı yoktur. Klasik bilgisayar ortamında bilgi belirli bir alanda depolanırken yapay sinir ağlarında veri tüm ağ boyunca dağıtılır. Yapay sinir ağları dağıtılmış bellek olarak çalışırlar. Bilgisayar ortamında sıfır ve birlerle ifade edilen veri belirli bir bellek alanında saklanır. Bilgisayardan bilgi belirli bellek bölgesine ulaşılarak alınır ve işlenir. Yapay sinir ağına bilgi tanımlanmamış giriş ya da gürültü olarak ağa sunulur. Yapay sinir ağları, giriş ile tüm ihtimalleri birleştirerek en uygun örneği çıkış olarak tanımlar. Bu çıkış değeri ağdaki giriş değeri hakkındaki bilgiyi temsil eder.

### 2.2.13. Sadece Numerik Bilgiler ile Çalışabilmektedirler

Yapay sinir ağları sadece numerik bilgilerle çalışabilmektedir. Örnek olay kümesi seçimi kadar giriş ve çıkış verileri belirlemekte önemlidir. Girdi ve çıktı çiftleri ağa tanıtılırken numerik olarak girilmeli. Ağa verilecek bilgilerin içeriğinde sayısal olmayan faktörler yer alıyorsa onları numerik olarak ifade edilmeli. Bu dönüşüm

işlemi çeşitli yollarla gerçekleştirilmekte bu da ağın performansını direk olarak etkilemektedir. Sayısal omayan verilerin dönüşümü kullanıcının tasarımına bırakılmıştır. Örneğin bir fotoğraftaki resmin bayan yada erkek olduğunu belirlemek için resmin gri tonları kullanılır. Resmin her noktasını gri tonunu gösteren 0-256 arasında bir sayı vardır. Fakat uygulama anlamında sıkıntı yaratacak durum söz konusudur. Bu resim için 65536 adet girdi ünitesine ihtiyaç duyulur. Bunun yerine aynı rakamlar sayılarak girdi sayısı 257 indirgenebilir. Ayrıca bu resim onarlı gruplara ayrılarak sınıflandırma işlemi yapılır. Her gruba ait girdi değerleri gri tonlar hesaplanarak ağa girdi olarak verilebilir. Böylelikle ağın girdi sayısı 65536 değerinden 10 değerine kadar düşürülebilir. Bu örnektende anlaşılacağı gibi ağın giriş ve çıkış bilgilerini numerik hale getirmek ve tasarlamak kullanıcıya bırakılmıştır. Ağın performansı kullanıcının uyguladığı yönteme göre değişebilmektedir.

### 2.2.14. Programların Çalışma Stili ve Bilginin Saklanması

Yapay sinir ağına yapılacak olan herhangi bir giriş verisinin tanımlanabilmesi ve bu verinin daha sonra kullanılabilmesi için verinin ağda temsil edilme biçimini, saklama şeklini ve verinin geri çağrılma yöntemlerinin bilinmesi gerekir. Bilgisayar ortamında veriler sıfır ve birlerden oluşan veri setleri ile ifade edilir. Fakat yapay sinir ağlarında veriler matematiksel işlevler ile temsil edilir. İşlem elemanları arasındaki bağlantı ağırlıkları ise bu işlevin değişkenleri olarak görev yapar. Bu ağırlıkların kendi başına bir anlamı olmamasına rağmen saklanan bilginin tanımlanmasını sağlarlar.

Klasik bilgisayar ortamında bilgi belirli bir alanda depolanırken yapay sinir ağlarında veri tüm ağ boyunca dağıtılır. Yapay sinir ağları dağıtılmış bellek olarak çalışırlar. Bilgisayar ortamında sıfır ve birlerle ifade edilen veri belirli bir bellek alanında saklanır. Bilgisayardan bilgi belirli bellek bölgesine ulaşılarak alınır ve işlenir. Yapay sinir ağına bilgi tanımlanmamış giriş ya da gürültü olarak ağa sunulur. Yapay sinir ağları, giriş ile tüm ihtimalleri birleştirerek en uygun örneği çıkış olarak tanımlar. Bu çıkış değeri ağdaki giriş değeri hakkındaki bilgiyi temsil eder.

Yapay sinir ağı programları çalışma sitili bilinen programlama yöntemlerinden oldukça farklıdır. Geleneksel programlama ve yapay zeka yöntemlerinin uygulandığı bilgi işleme yöntemlerinden tamamen farklı bir bilgi işleme yöntemi vardır.Yapay sinir ağları sayısal bilgisayarlardan çok farklı özellikler gösterirler. Bu özellikler tabloda genel olarak verilmiştir (Elmas,2003,S.24).

Tablo: 2.3.1 Sayısal Bilgisayarlar ile Yapay Sinir Ağlarının Karşılaştırılması

| **Sayısal Bilgisayarlar** | **Yapay Sinir Ağları** |
|---|---|
| *Tümden Gelim Yöntemi:* Giriş bilgilerine bilinen kurallar uygulanarak çıkış verisi elde edilir. | Tüme Varım Yöntemi: Eğitilen örneklerin giriş ve çıkış bilgileri ağa verilir. Kurallar uygulama türüne göre kullanıcının kendisi belirler. |
| Hesapla merkezi, eş zamanlı ve ardışıldır. | Hesaplama toplu, eşzamansız ve öğrenmeden sonra paraleldir. |
| Bellek paketlenmiş, hazır bilgi depolanmış ve yer adreslenmiştir. | Bellek ayrılmıştır, dahilidir ve içerik adreslenebilir. |
| Hata toleransı yoktur. | Eğer bilgi, gürültülü veya kısmi ise kurallar bilinmiyorsa ya da karışıksa hata toleransı uygulanabilir |
| Hızlıdır. | Yavaştır. |
| Bilgiler ve Algoritmalar kesindir. | Yapay sinir sistemleri deneyiminden yaralanılır. |

### 2.2.15. Yapay Sinir Ağlarının Güvenli Çalıştırılabilmesi için Performans Testine İhtiyaç Duyarlar

Yapay sinir ağlarda probleme uygun ağ yapısının belirlenmesi için geliştirilmiş bir kural yoktur. Uygun ağ yapısı deneyim ve deneme yanılma yolu ile belirlenmektedir. Ağın parametre değerlerinin belirlenmesinde de bellirli bir kural mevcut değildir**.** Yapay sinir ağlarda öğrenme katsayısı, hücre sayısı, katman sayısı gibi parametrelerin belirlenmesinde belirli bir kural seti bulunmaz. Bu değerlerin belirlenmesi için belirli bir standart olmamakla birlikte her problem için farklı bir yaklaşım söz konusu olabilmektedir. Öğrenilecek problemin ağa gösterimi önemli bir problemdir**.** Yapay sinir ağlar nümerik bilgiler ile çalışabilmektedirler. Problemler YSA'lara tanıtılmadan önce nümerik değerlere çevrilmek zorundadırlar. Burada belirlenecek gösterim mekanizması ağın performansını doğrudan etkileyecektir. Bu da kullanıcının yeteneğine bağlıdır. Ağın eğitiminin ne zaman bitirilmesi gerektiğine ilişkin belli bir yöntem yoktur**.** Ağın örnekler üzerindeki hatasının belirli bir değerin altına indirilmesi eğitimin tamamlandığı anlamına gelmektedir. Burada optimum

neticeler veren bir mekanizma henüz yoktur ve YSA ile ilgili araştırmaların önemli bir kolunu oluşturmaktadır.

Yapay sinir ağlarda eğitim ve test aşaması için iki ayrı örnek setleri oluşturulur. Eğitme işlemi örnek olay kümeleri ile tamamlandıktan sonra test aşamasına geçilir. Test için ayrılan örnek olay kümeleri kullanılarak çıkış değerlerinin doğru elde edilip edilmediği kontrol edilir. Eğer çıkış hedef vektörüne yakın bir değer elde edildiyse performans testi başarı ile tamamlanmıştır. Eğer çıkış vektörü hedef vektöründen uzak değerler elde ediliyorsa ağ öğrenme işlemini gerçekleştirememiştir. Bu durumda öncelikle örnek olay kümesi gözden geçirilmeli. Gerekli olduğu kanısına varılırsa örnek olay kümesi genişletilmeli. Sorun çözülmediği takdirde ağ yapısı tekrar gözden geçirilir. Geriye dönüp bütün süreçleri tekrarlama işlemi ağın öğrenmeyi tamamlayıp test verilerine optimuma yakın sonuç verinceye kadar devam edilir.

## 2.3. Yapay Sinir Ağlarının Genel Kullanım Alanları

Yapay sinir ağları kullanıcı tarafından verilen örnek kümelerini kullanarak olayları öğrenebilen, olaylar karşısında nasıl tepki üreteceğini belirleyebilen bilgisayar sistemleridir. Yapay sinir ağları öğrenme, ilişkilendirme, sınıflandırma, genelleme, özellik belirleme ve optimizasyon alanlarında başarılı bir şekilde uygulanmaktadır. Gerçekleşmiş örneklerden elde ettikleri bilgiler ile kendi deneyimlerini oluştururlar. Bu deneyimlerini kullanarak benzer konularda karar verebilirler.

Günümüzde birçok problem şekil tanıma problemi haline getirilmekte ve ondan sonra çözülmekte. Bu sebeple yapay sinir ağları bir çok alanda kullanılabilmekte. Her problemi yapay sinir ağı ile çözmekte yanlış bir yaklaşım olur. Elimizde yeterince gerçekleşmiş örnek sayısı mevcut değilse ve belirlenen problemin çözümü etkin ve verimli olarak başka bir yöntem kullanarak çözümlenebiliyorsa, problemi yapay sinir ağı uygulaması haline getirmek yanlış olur.

Bir problemin yapay sinir ağı ile çözümlenebilmesi için sağlaması gereken koşullar mevcuttur. Öncelikle yapay sinir ağı kullanarak problemlere pratik çözümler üretilebilme durumunu sağlamalı. Problem başka metotlar uygulanarak çözümlenebilirken yapay sinir ağının tercih edilmesi için yapay sinir ağının diğer metotlardan daha verimli ve etkin sonuçlar üretebiliyor olması gerekir. Yapay sinir ağları doğrusal olmayan, çok boyutlu gürültülü, karmaşık, kesin olmayan, eksik, kusurlu ve problemin çözümü için özellikle bir matematiksel modelin ve algoritmanın bulunmaması hallerinde kullanılması faydalıdır .

Geliştirilmiş ağların başlıca gerçekleştirebildiği fonksiyonlar olasılık, sınıflandırma, optimizasyon, örüntü tanıma, doğrusal olmayan sistem modelleme, doğrusal olmayan sinyal modelleme, zaman serileri analizi, sinyal filtreleme, veri sıkıştırma olarak sıralayabiliriz (Öztemel, 2003,36).

Yapay sinir ağları yukarıda belirttiğimiz fonksiyonları gerçekleştirebildikleri için yaygın bir kullanım alanı oluşturur. Bunları sektörlere göre endüstriyel uygulamalar, finansal uygulamalar, askeri ve savunma uygulamaları, sağlık uygulamaları ve diğer alanlar olarak sınıflandırabiliriz.

Yapay sinir ağlarının kullanım alanlarına sektör bazında ele alınırsa uzay alanında uçuş simülasyonları, otomatik pilot uygulamaları, komponentlerin hata denetimlerinde kullanılır. Otomotiv sektöründe otomatik yol izleme, rehber, garanti aktivite analizi, yol koşullarına göre sürüş analizi gerçekleştirir. Bankacılık alanında kredi uygulamaları geliştirilmesi, müşteri analizi ve kredi müracaat değerlendirilmesi, bütçe yatırım tahminlerinde yer alır. Savunma alanında ise silah yönlendirme, hedef seçme, radar, sensör sonar sistemleri, sinyal işleme, görüntü işleme uygulamalarında kullanılır. Elektronik alanda kod sırası öngörüsü, çip bozulma analizi, non-lineer modellemeler de yer alır. Eğlence alanında ise animasyonlar, özel efektler, pazarlama öngörüsüne yardımcı olur. Finans sektöründe kıymet biçme, pazar performans analizi, bütçe kestirimi, hedef belirleme gibi bir çok pozitif getiri sağlar. Sigortacılık anlamında ürün optimizasyonu, uygulama politikası geliştirme yönünde avantajları vardır. Üretim alanında ise üretim işlem kontrolü, ürün dizaynı, makina yıpranmalarının tespiti, dayanıklılık analizi, kalite kontrolü, iş çizelgeleri hazırlanması gibi olanaklar sağlar. Sağlık sektöründe uygulamaları ise göğüs kanseri erken teşhis ve tedavisi, EEG, ECG, MR, kalite artırımı, ilaç etkileri analizi, kan analizi sınıflandırma, kalp krizi erken teşhis ve tedavisi sağlar. Petro kimya alanıda arama, verim analizi sağlarken robotik alanında yörünge kontrol, forklift robotları, görsel sistemler, uzaktan kumandalı sistemler, optimum rota belirleme uygulamalarında yardımcı olur. Dilde ise sözcük tanıma, yazı ve konuşma çevrimi, dil tercüme konularında başarıyla kullanılabilir. Telekomünikasyon sektöründe görüntü ve data karşılaştırma, filtreleme, eko ve gürültü sönümlendirilmesi, ses ve görüntü işleme, trafik yoğunluğunun kontrolü ve anahtarlama uygulamalarını gerçekleştirir. Güvenlik açısından parmak izi tanıma, kredi kartı hileleri saptama, retina tarama, yüz eşleştirme olarak özetlenebilir.

Bu örnekler çoğaltılabilir. Görüldüğü gibi YSA'lar günlük hayatımızda farkında olmadığımız pek çok alanda kullanılmaktadır. Gün geçtikçe uygulama alanları genişlemekte ve gelişmektedir.

### 2.3.1. Yapay Sinir Ağlarının Mühendislik Uygulamaları

Yapay sinir ağları birçok mühendislik alanında problemlerin çözümlenmesi aşamasında yaygın olarak kullanılmaktadır. Kimya mühendisliği, inşaat ve yapı mühendisliği, elektrik ve elektronik mühendisliği, imalat ve makine mühendisliği, sistem ve kontrol mühendisliği alanlarında kullanımını genel olarak ele alacağız.

*Kimya Mühendisliği*; kimyasal reaktör seçimi, dinamik işlemlerde hata belirlenmesi, endüstriyel polimerisasyonda eritme akış indisi tahmini, endüstriyel mayalama işleminin modellenmesi, biyokimyasal işlemlerde mikrobik konsantrasyonların tahmininde uygulanmıştır.

*İnşaat ve Yapı Mühendisliği;* konstrüksiyon projelerinde kaynak seviyelerini belirleme, bir rezervuardan çıkışı kontrol, biyolojik bilgiler yardımıyla nehir suyu kalitesinin sınıflandırılması, nehirlerin akışının tahmin edilmesi, sonlu-eleman-temelli yapısal analiz işleminin modellenmesi, yapı malzemelerinin iç yapılarındaki çatlakların tespit edilmesi, depreme maruz betonarme çerçevelerde emniyetli yatay taşıyıcı tahmininde uygulanmıştır.

*Elektrik ve Elektronik Mühendisliği;* hastaların alarmla kontrolü, gürültülü resimlerin kalitesini artırma, görüntülerin sıkıştırılması, gürültü filtreleme ve resimlerdeki kenar bilgisinin çıkarılması, güç sisteminde harmoniklerin tahmini, gezgin haberleşme sisteminde kanal dağıtımı, ultrasonik müziklerden objelerin sınıflandırılması, optik okuyucu sistemler için resimlerin ön işlemesi alanlarında kullanılmıştır.

*İmalat ve Makine Mühendisliği*; metal kesme tezgahının kontrolü, akustik salınım ve iş parçası kuvvetiyle iş parçası yatağının kontrolü ve güç tüketimi ve iş parçası ivmesi, hücreli imalat için grup teknolojisi parça gruplarının tasarımı, hareket eden nesneler için engelsiz yol planlaması, makine parametrelerinin optimizasyonu, makine arızalarının sınıflandırılması, malzemelerin ısı transferinin belirlenmesi, uçak kanat kutularının yapısının tasarımında uygulanmıştır.

*Sistem ve Kontrol Mühendisliği*; esnek kollu robotun kontrolü, bir model helikopterin havada kontrolü, çok değişkenli (mafsallı) robotun yörünge koordinasyonu, iki sıvı tank sisteminin akış seviye kontrolü, anestezi derinliği kontrolü ve ölçülmesi, bir robot için optimal yolun bulunması, banyo suyu sıcaklığının kontrolü, endüstriyel robot kontrolü, sistem kimliklendirme gibi birçok alanda uygulanmıştır.

*Anten ve Uygulamaları;* yapay sinir ağlarında bir dinamik öğrenme algoritması (DÖA) kullanılarak anten dizi elemanlarından elde edilen işaretler arasındaki faz farklılıklarının karşılaştırılmasıyla radar izleme gerçekleştirilmiştir. Mikrodalga parlaklığına bağlı jeofiziksel parametrelerin tayini, DÖA sinir ağı kullanılarak gerçekleştirilmiştir. Uzaktan kontrollü görüntü sınıflama işlemi, danışmanlı DÖA ve danışmansız (ART2) öğrenme algoritmaları kullanılarak yapılmıştır. YSA'ları kullanılarak anten dizi analizi yapılmıştır. Aynı zamanda, anten dizi tasarımında, genetik algoritma ve tabu araştırma algoritması kullanılmıştır. Osilatör YSA ile elektromagnetik işaretlerin etkileşim modelleri oluşturulmuştur.

## 2.4. Yapay Sinir Ağlarının Avantajları

Yapay sinir ağları karar hızı açısından insan beyni ile henüz tam olarak eşleşmese bile yapısallıkları ve hassas eşleştirmeleri ile gün geçtikçe önem kazanmaktadırlar. Yapay sinir ağları doğrusal olmayan problemleri çözüme ulaştırması, öğrenme işlemini gerçekleştirebilmesi, eğitilme işleminden sonra karar verebilmesi, ilişkilendirme, sınıflandırma yapabilmesi, esnek bir yapıya sahip olması, gerçek zamanlı kullanılabilmesi, ucuzluğunun yanı sıra pratik olması ve sınırsız sayıda değişken ve parametre ile çalışabilmesi gibi birçok avantaj sağlamaktadır.

Yapay sinir ağının en önemli özelliği öğrenebilme yetisine sahip olmasıdır. Gerçek hayattan alınan örnek uygulamalar kullanılarak geleneksel teknikler için çok karmaşık olan problemlerin çözümü ağın eğitilmesi ile gerçekleştirilebilir. İnsanların kolay bir şekilde gerçekleştirebildiği ama geleneksel metotların uygulanamadığı problemlerin çözümünde kullanılabilir. Çünkü yapay sinir ağları tam olarak insan beynini yansıtmasa da esin kaynağını insan beyninden almıştır. Yapay sinir ağları matematiksel olarak modellenemeyen karmaşık problemleri çok rahat bir şekilde modelleyebilmektedir. Önemli olan seçilen örnek olayların problemin kısıtlarını tam olarak ortaya koymasıdır. Uygulamanın başarısı seçilen örneklerin problemi tam

olarak yansıtmasına doğru orantılı olarak bağlıdır. Yapay sinir ağlarında örnekler dışında herhangi bir ön bilgiye ihtiyaç yoktur.

Yapay sinir ağları gerçekleşmiş örneklerden yararlanarak eğitimini gerçekleştirir. İnsanların gerçek olayların arkasındaki değişik faktörleri belirleme, faktörleri ilişkilendirme ve bu faktörlerin birbiri üzerine etkilerini belirlemeleri zordur. Eğitimini tamamlamış ağa kullanıcıların problemle ilgili olaylar arasındaki ilişkiyi bilmesi ve ağa belirtmesi beklenmemektedir. Ayrıca bu olayların ve olayların arkasındaki faktörlerin doğrusal olması da gerekmez. Yapay sinir ağlarının en önemli özelliklerinden biride doğrusal olmayan yapıları da dikkate alabilmesidir.

Doğrusal olmayan gerçek hayattaki problemlerin ilişkilendirilmesi ve sonuç çıkartılmasına yönelik model oluşturmak zordur. Dolayısıyla problemlerin çözümü için bazı varsayımlar yapmak gerekir. Buda modellenen sistemin gerçek sisteme uygunluğunu azaltır ve sistemin davranışı kontrol altına alınmasını engeller. Yapay sinir ağlarında ilişkilerin doğrusal olup olmaması önemli değildir. Çünkü ilişkileri gerçekleşen örnekler üzerinde kendiliğinden hesaba katılmaktadır. Bu yüzden ilişkilerin modele katılması geleneksel sistemdeki kadar zor değildir.

Yapay sinir ağları uygulamaları hem pratik hem de maliyet açısından ucuzdurlar. Belirlenen problem için örnek olay kümesini oluşturduktan sonra basit bir bilgisayar yazılımı ile olay sonuçlandırılabilir. Alt yapı çalışmaları içinde büyük bir maliyet gerekmemektedir. Ayrıca daha önce yapılan bir uygulamanın üzerine yapılacak değişiklikler için ek bir yatırım yapmaya gerek yoktur. Yapılacak değişikliklere göre ağ yeniden eğitilebilir, bazen bu değişiklikler için ağın yeniden eğitilmesine gerek kalmayabilir. Çünkü ağ ortama uyumu öğrenerek problemi çözüme ulaştırabilir.

Geleneksel işlemcilerde tek bir merkezi işlem elemanı her hareketi sırasıyla gerçekleştirirken yapay sinir ağlarında problem alt parçalara ayrılarak, her problem parçası ile birden fazla işlem elemanı eş zamanlı çalışabilmektedir. Yapay sinir ağları bağlantı ağırlıklarını ayarlayabilmesinden dolayı esnek bir yapıya sahiptir. Bu esnek yapı sayesinde ağın bir kısmı zarar görmesi sadece performans açısından ağı etkiler. Modelin işlevini yitirmesi söz konusu bile değildir. Ayrıca alt problemlere atanan işlem elemanlarının yoğun bağlantılı yapısı sayesinde en karmaşık problemlere bile uygulanıp tatminkâr sonuç alınmasını sağlamaktadır.

Yapay sinir ağlarının paralel çalışabilmeleri sayesinde gerçek zamanlı kullanılabilmelerini sağlamıştır. Genelleme özelliklerinden dolayı hatalı ya da

gürültülü verilere çözüm üretebilmektedirler. Yapay sinir ağları iyi bir sınıflandırıcılardır. Yapay sinir ağının işlem elemanları arasındaki ağırlıklı bağlantılar sayesinde dağıtılmış bellekte bilgi saklayabilirler.

Klasik bilgisayar ortamında bilgi belirli bir alanda depolanırken yapay sinir ağlarında veri tüm ağ boyunca dağıtılır. Yapay sinir ağları dağıtılmış bellek olarak çalışırlar. Bilgisayar ortamında sıfır ve birlerle ifade edilen veri belirli bir bellek alanında saklanır. Bilgisayardan bilgi belirli bellek bölgesine ulaşılarak alınır ve işlenir. Yapay sinir ağına bilgi tanımlanmamış giriş ya da gürültü olarak ağa sunulur. Yapay sinir ağları, giriş ile tüm ihtimalleri birleştirerek en uygun örneği çıkış olarak tanımlar. Bu çıkış değeri ağdaki giriş değeri hakkındaki bilgiyi temsil eder.

Yapay sinir ağları sayısız değişken ve parametre ile rahatlıkla çalışabilen yapısı vardır. Bu özelliği ile en karmaşık problemlere bile mükemmel öngörü doğruluğunda genel çözümler üretebilmektedir.

### 2.5. Yapay Sinir Ağlarının Dezavantajları

Yapay sinir ağının eğitilmesi için probleme ilişkin gerçek örnek olay setlerine ihtiyaç vardır. Problemin sınırlarını belirleyecek örnek olay verileri bulunamadığında ya da problem tam olarak örnek olaylarla yansıtılamadığında ağ sağlıklı çözümler üretmekte yetersiz kalır. Ayrıca ağın ürettiği sonuçların optimum olduğu söylenemez. Üretilen sonuçlar feasible alanda olmasına rağmen optimum değerde olmayabilir.

Diğer önemli bir dezavantajı ise örneklerin seçilip tasarlanması için belirli bir kural zinciri yoktur. Kullanıcı tarafından tecrübesine dayanarak örnek olayları belirlemesi ve formulize etmesi beklenir. Dolayısıyla aynı problem farklı kullanıcılar tarafından değişik şekilde modelleneceğinden farklı performanslar elde edilecektir. Doğru çözüme yaklaşmak yine kullanıcının tecrübelerine kalmıştır.

Yapay sinir ağının oluşturulması, model tespiti, ağın topolojisinin belirlenmesi ağın performansı üzerinde oldukça etkilidir. Fakat bu faktörlerin belirlenmesinde belirli bir kural seti mevcut değildir. Bu noktada yine ağın kullanıcısının tecrübelerinden yararlanılır. Aynı zamanda ağın davranışlarının da açıklanması mümkün değildir. Buda ağa olan güveni azaltmaktadır. Ağın eğitilmesi uzun zaman alabilir. Bu sebeplerden dolayı ağın kullanım alanları azalmaktadır.

Basit olarak görülebilecek modelleme yapılarına rağmen uygulaması zor ve karmaşık olabilir. Bazı durumlarda yakınsama yapmak bile zor olabilir. Bu durum uygulama alanına bağlıdır. Genellikle karmaşık problemlerde karşımıza çıkabilir.

Diğer önemli bir nokta ise eğitim süresi için kesin bir kural yoktur. Ağ eğitimini tamamladıktan sonra bulunan noktanın en iyi sonuç olduğu fark edilmeyip eğitime devam edildiğinde ağ veri setinden hatalı veriler çıkarmaya başlayabilir. Bu yüzden aşırı eğitme sorununa dikkat edilmeli ve eğitme uzunluğu kullanıcı tarafından iyi ayarlanmalıdır.

Ağın problemi çözememesi durumunda ağın yapısı, girdi ve çıktı verileri, tabaka sayısı, her tabakadaki eleman sayısı, tabakalar arasındaki bağlantılar, toplama, transfer ve eğitme fonksiyonları ve hatta başlangıç ağırlıkları gözden geçirilmeli. Bu da çözüme ulaştırılamamış her sorun için başlangıç noktasına geri dönmektir.

### 2.6. Yapay Sinir Ağlarının Tarihçesi

Bilgisayarlar yalnızca aritmetik işlemleri gerçekleştirirken insan gibi öğrenmeleri ve çevre şartlarına göre karar verebilen sistemler olarak çalışmaları istenilmiştir. Yapay sinir ağlarının tarihçesi nörobiyoloji konusuna ilgi duymaları ve bu bilgileri bilgisayar ortamına taşıma istekleri ile başlamıştır. Yalnız 1950 yıllının sonlarında büyük ölçekli işlemcilerin gelişmesiyle beynin yaptığı işlemleri gerçekleştirebilecek yapay sinir ağların oluşturulabilmesi mümkün olmuştur. Bilgisayarların yaygın bir şekilde kullanılmaya başlamasıyla araştırmalar hız kazanmış fakat 1970'li yıllarda yaşanan olumsuz gelişmelerle araştırmalar durmuştur. Yapay sinir ağı literatüründe bu olumsuz gelişme XOR problemi olarak bilinmektedir. 1970 yılları YSA için bir dönüm noktasını teşkil etmiş daha önce aşılması imkânsız görünen pek çok problem bu dönemlerde aşılmıştır.

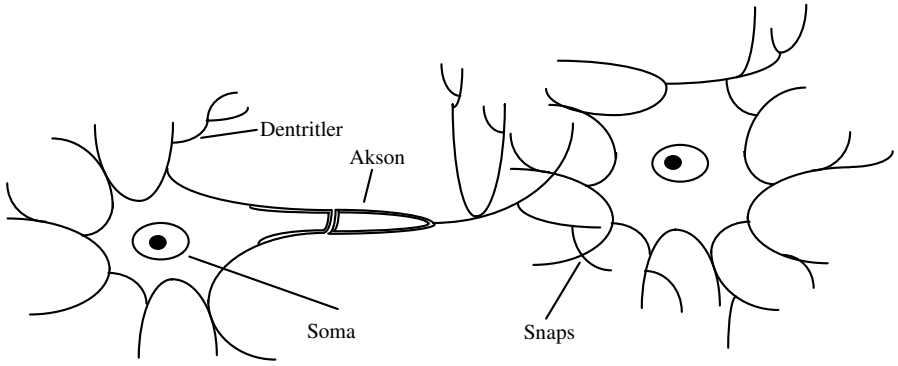
Yapay sinir ağları ile ilgili çalışmalar 20. yüzyılın ilk yarısında başlamış ve günümüze kadar büyük bir hızla devam etmiştir. İlk yapay sinir ağı modeli 1943 yılında, bir sinir hekimi olan Warren McCulloch ile bir matematikçi olan Walter Pitts tarafından gerçekleştirilmiştir. McCulloch ve Pitts, insan beyninin hesaplama yeteneğinden esinlenerek, elektrik devreleriyle basit bir sinir ağı modellemişlerdir. Böylelikle bir biyolojik nöronun temel fonksiyonlarının basit bir eşik cihazı olarak modelleneceğini göstermişlerdir (Warren McCulloch, Walter Pitts, 1943, S.115). 1948 yılında "Wiener "Cybernetics" isimli kitabında, sinirlerin çalışması ve davranış özelliklerine değinmiş, 1949 yılında ise Hebb "Organization of Behavior" adlı eserinde öğrenme ile ilgili temel teoriyi ele almıştır. Hebb hücresel seviyede beynin öğrenme mekanizmasını ortaya koymuştur. Bu mekanizmaya göre bir nörondan dentrit yoluyla gelen ve bir akson yoluyla alınan giriş onun bir darbe üretmesine sebep olur. Sonraki

aksonal girişlerin darbe üretmesi olasılığı artar. Böylelikle yapılan davranışın mükâfatının ortaya çıkacağını açıklamıştır. Hebb kitabında öğrenebilen ve uyum sağlayabilen sinir ağları modeli için temel oluşturacak "Hebb kuralı"nı ortaya koymuştur. Hebb kuralı; sinir ağının bağlantı sayısı değiştirilirse, öğrenebileceğini ön görmekteydi (Hebb,1949).

Hızlı çalışmaya yönelik ilk yapay sinir ağı çalışmaları 1950'li yıllarda başlamıştır. Basit nöron modellerine dayalı bir hesaplama modeli, 1950'lerde Frank tarafından önerilmiş ve ardından perceptron olarak tanımlanan tek katmanlı yapay sinir ağı modeli ortaya çıkmıştır (Rosenblatt,1959). 1957 yılında Frank Rosenblatt'ın Perceptron'u geliştirmesinden sonra, YSA'lar ile ilgili çalışmalar hız kazanmıştır. Perceptron; beyin işlevlerini modelleyebilmek amacıyla yapılan çalışmalar neticesinde ortaya çıkan tek katmanlı eğitilebilen ve tek çıkışa sahip bir ağ modelidir (Rosenblatt,1962). 1959 yılında Bernand Widrow ve Marcian Hoff (Stanford Üniversitesi) ADALINE (Adaptive Linear Neuron) modelini geliştirmişler ve bu model YSA'ların mühendislik uygulamaları için başlangıç kabul edilmiştir. Bu model Rosenblatt'ın Perceptron'una benzemekle birlikte, öğrenme algoritması daha gelişmiştir. Bu model uzun mesafelerdeki telefon hatlarındaki yankıları ve gürültüleri yok eden bir adaptif filtre olarak kullanılmış, ve gerçek dünya problemlerine uygulanan ilk YSA olma özelliğini kazanmıştır. Bu yöntem günümüzde de aynı amaçla kullanılmaktadır. 1960' ların sonlarına doğru YSA çalışmaları durma noktasına gelmiştir. Buna en önemli etki; Minsky ve Pappert tarafından yazılan Perceptrons adlı kitaptır. Burada YSA'ların doğrusal olmayan problemleri çözemediği meshur XOR problemi ile ispatlanmış ve YSA çalışmaları bıçak gibi kesilmiştir (Minsky, Papert, 1969). Tüm bunlara rağmen Anderson, Amari, Cooper, Fukushima, Grossberg, Kohonen, Hopfield gibi bilim adamları çalışmalarını sürdürmüşler, 1972 de Kohonen ve Anderson Associative memory konusunda benzer çalışmalar yayınlamışlardır. Kohonen daha sonra 1982 yılında Kendi Kendine Öğrenme Nitelik Haritaları (Self Organizing Feature Maps SOM) konusundaki çalışmasını yayınlamıştır. 1960'ların sonlarına doğru Grosberg Carpenter ile birlikte Adaptif Rezonans Teorisini (ART) geliştirmiştir.

1969-1982 yılları arasındaki çalışmalarda ise teori artık oturmuş ve 1982'de J.J. Hopfield tarafından yayınlanan "Neural Networks and Physical Systems" adlı çalışma ile çağdaş YSA devri başlamıştır. Bu çalışmada Hopfield, nöronların karşılıklı etkileşimlerine dayanan bir nöral hesaplama modeli önermiştir. Bu model, bir enerji

fonksiyonunu, alabileceği en az değerine indiren birinci mertebe lineer olmayan diferansiyel denklemlerden oluşmuştur. Hopfield; ağ seviyesinde, tek tek nöron seviyesinde varolmayan hesaplama kapasitesinin bulunduğunu öne sürmüştür. Bu tür YSA ya, "Hopfield Ağı" denilmektedir. Hopfield'in geri beslemeli YSA modelini ortaya atması ve bunun pratik optimizasyon problemlerinde kullanılabilirliğini göstermesi (Hopfield, 1985), YSA konusundaki çalışmaları yeniden hızlandırmıştır.




Şekil 2.7.1 Biyolojik nöron/sinir hücresinin şematik yapısı

1970'lerin sonlarına doğru Fukushima, NEOCOGNITRON modelini tanıtmıştır. Bu model şekil ve örüntü tanıma amaçlı geliştirilmiştir. 1982 1984 yıllarında Hopfield tarafından yayınlanan çalışmalar ile YSA'ların genelleştirilebileceği ve çözümü zor problemlere çözüm üretebileceğini göstermiştir. Geleneksel gezgin satıcı problemini çözmüştür. Bu çalışmaların neticesi Hinton ve arkadaşları'nın geliştirdiği Boltzman Makinası'nın doğmasına yol açmıştır.
1988 yılında, Broomhead ve Lowe radyal tabanlı fonksiyonlar modelini (Radial Basis Functions RBF) geliştirmişler ve özellikle filtreleme konusunda başarılı sonuçlar elde etmişlerdir. Daha sonra Spect, bu ağların daha gelişmiş şekli olan Probabilistik ağlar (PNN) ve Genel Regresyon Ağlarını (GRNN) geliştirmiştir.

Günümüze kadar yapılan sayısız çalışma ve uygulamaların bazılarını kronolojik olarak aşağıdaki gibi sıralanabilir.

1890- İnsan beyninin yapısı ve fonksiyonları ile ilgili ilk yayının yazılması
1911- İnsan beyninin sinir hücrelerinden oluştuğu fikrinin benimsenmesi

1943- Yapay sinir hücrelerine dayanan hesaplama teorisinin ortaya atılması ve eşik değerli mantıksal devrelerin geliştirilmesi

1949- Öğrenme prosedürünün bilgisayarlar tarafından gerçekleştirilecek şekilde geliştirilmesi

1956 - 1962 ADALINE ve Widrow - Hoff öğrenme algoritmasının geliştirilmesi

1957 - 1962 Perceptron'un geliştirilmesi

1965- İlk makina öğrenmesi kitabının yayınlanması

1967 - 1969 bazı gelişmiş öğrenme algoritmalarının geliştirilmesi

1969- Tek katmanlı algılayıcıların yetersizliklerinin ispatlanması

1969 - 1972 doğrusal ilişkilendiricilerin geliştirilmesi

1972- Korelasyon matris belleğinin geliştirilmesi

1974- Geriye yayılım modelinin geliştirilmesi

1978- ART modelinin geliştirilmesi

1982- Çok katmanlı algılayıcıların geliştirilmesi

1984- Boltzman Makinası'nın geliştirilmesi

1988- RBF modelinin geliştirilmesi

1991- GRNN modelinin geliştirilmesi (Öztemel,1993)

### 2.7. Yapay Sinir Ağlarının Geleceği

Yapay sinir ağının eğitimi gerçekleşmiş olay kümelerini kullanarak gerçekleştirilir. Fakat yapay sinir ağının eğitimi uzun zaman aldığından geleceğin çalışma alanı olarak belirlenmiştir. Zaman problemini çözüme ulaştırmak amacıyla daha etkin ve iyi performansla çalışabilen yeni öğrenme algoritmaları üzerine çalışmalar yapılmaktadır. Ayrıca performans sadece öğrenme algoritmalarındaki iyileştirme ile artmayacağından değişen modellere karşılık verebilecek ağlar ve silikon ağları geliştirmek amacıyla da çalışmalar yapılmaktadır.

Yapay sinir ağlarını kullanıma sunmak için özel yongalar gerekmektedir. Özel şirketlerde ve üniversitelerde sinir ağı yongası geliştirmek üzere çalışmalar yapılmaktadır. Bu çalışmalar da gelinen nokta ise sayısal, analog, ve optik olmak üzere üç tip sinir yongası üzerine çalışmalar yoğunlaştırılmıştır. Bazı özel şirketler özel derlenmiş devreler ile sinir ağı yongasını yaratmak için silikon malzemeler üzerine çalışmaktadırlar. Optik okuyucu yongalar ile sinire benzer sayısal yongalar ümit verici çalışmalar arasındadır(Elmas,2003, S.29).

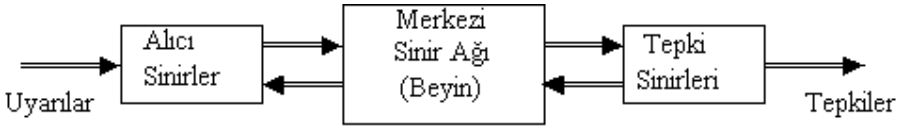
Yapay sinir ağı çalışmalarında günümüze kadar iyi bir gelişme alınmasına rağmen kaynağı insan beyni olan ağların çalışma performansı insan beyni ile henüz kıyas edilebilecek düzeye ulaşmamıştır. Gelecekte yapılacak çalışmaların en büyük amacı mümkün olduğunca klasik bilgisayarları eğitilebilen ve insan gibi hızlı bir şekilde karar alabilen mekanizmalara çevirmek isteği olacaktır.

# 3.YAPAY SİNİR AĞLARININ YAPISI VE TEMEL ELEMANLARI

## 3.1. Biolojik Sinir Hücreleri

Yapay sinir ağları insan beyninden esinlenerek ortaya çıkmış ve hala yoğun araştırma alanı olan yapılardır. Klasik bilgisayarlara insana özgü eğitilebilme ve karar alabilme yapılarını kazandırabilme isteği ile başlamıştır. Dolayısıyla yapay sinir ağlarını daha iyi kavrayabilmek için insan beyninin özelliklerini incelemek faydalı olacaktır. Henüz beynin tüm fonksiyonları tıp biliminde açıklanamamaktadır. Yapay sinir ağları kaynağı olan beynin tüm özelliklerini yansıtmamaktadır. Dolayısıyla beynin genel özellikleri ele alınacaktır.

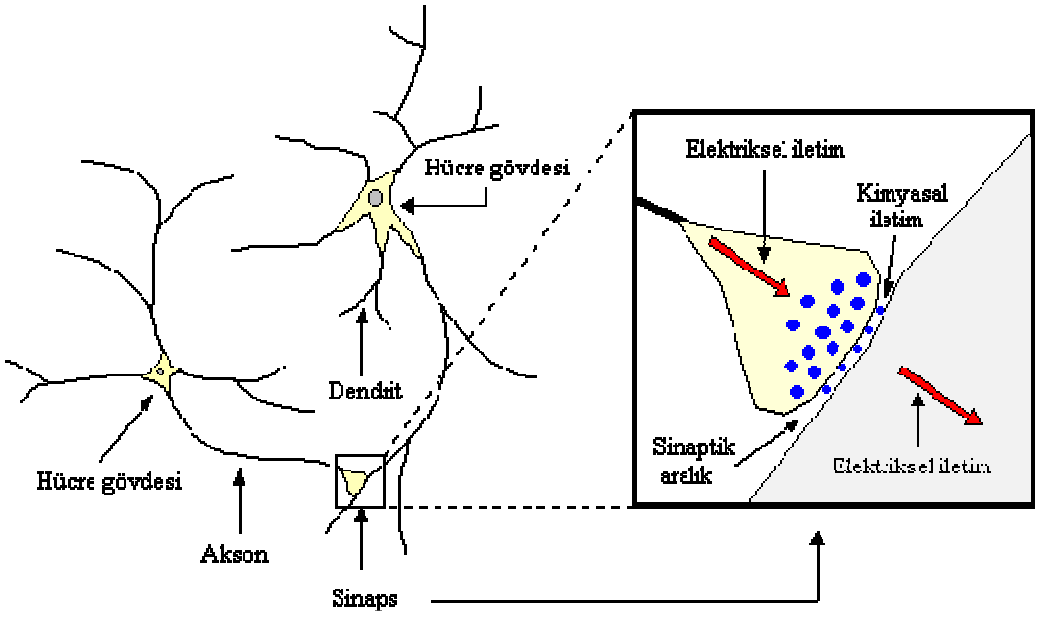
Beyin sinir sisteminin merkezi elemanıdır. Beyin sürekli iletilen bilgiyi alır, bilgiyi işler ve uygun kararları vererek ilgili bölgelere iletir. Çok basit olarak görülen beynin yapısı aslında çok karmaşıktır. Haykin (1999) tarafından sinir sistemi blok diagramının basit bir gösterimi şekilde sunulmaktadır.




Şekil: 3.1.1. Biyolojik Sinir Sisteminin Blok Diagramı

Biyolojik sinir sistemi, merkezinde sürekli olarak bilgiyi alan, yorumlayan ve uygun bir karar üreten beynin (merkezi sinir ağı) bulunduğu 3 katmanlı bir sistem olarak açıklanır. Alıcı sinirler (receptor) organizma içerisinden ya da dış ortamlardan algıladıkları uyarıları, beyine bilgi ileten elektriksel sinyallere dönüştürür. Tepki sinirleri (effector) ise, beyinin ürettiği elektriksel darbeleri organizma çıktısı olarak uygun tepkilere dönüştürür.

Merkezi sinir ağında bilgiler, alıcı ve tepki sinirleri arasında ileri ve geri besleme yönünde değerlendirilerek uygun tepkiler üretilir. Bu yönüyle biyolojik sinir sistemi, kapalı çevrim denetim sisteminin karakteristiklerini taşır. Merkezi sinir sisteminin temel işlem elemanı, sinir hücresidir (nöron) ve insan beyninde yaklaşık 10 milyar sinir hücresi olduğu tahmin edilmektedir. Sinir hücresi; hücre gövdesi, dendriteler ve axonlar olmak üzere 3 bileşenden meydana gelir. Dendriteler, diğer hücrelerden aldığı bilgileri hücre gövdesine bir ağaç yapısı şeklinde ince yollarla iletir. Axonlar ise elektriksel darbeler şeklindeki bilgiyi hücreden dışarı taşıyan daha uzun bir yoldur. Axonların bitimi, ince yollara ayrılabilir ve bu yollar, diğer hücreler için dendriteleri oluşturur.( Şekil: 3.1.2.)




Şekil: 3.1.2. Biolojik nöron yapısı

Beynin yapı taşlarını sinir hücreleri yani nöronlar oluşturur. Nöronların en belirgin özelliği vücudun diğer bölümlerinin tersine yeniden üretilmeyen belirli bir hücre türü olmasıdır. Beynin diğer temel yapısal ve fonksiyonel birimleri olan bağlantılar nöronlar arası etkileşimi sağlarlar.Beynin etkin çalışabilen organ olmasının en temel nedeni bu bağlantılı yapıdır. Bu yapı sayesinde beyin, bugünkü bilgisayar teknolojisinden kat kat daha etkin bir şekilde çalışabilmektedir.

Nöronların birbirleri ile elektrik sinyalleri kullanarak haberleşirler. Sinyaller bir nöronun aksonundan diğerinin dentritine gönderilir. Bir akson birden fazla dentrit ile ilişkiye girebilir. Bu bağlantının yapıldığı yere "snaps" denir. Nöronlar elektrik sinyalini hücre duvarındaki voltajı değiştirerek üretirler. Bu ise hücrenin içinde ve dışında dağılmış bulunan iyonlar vasıtasıyla olur. Bu iyonlar sodyum, potasyum, kalsiyum ve klor gibi iyonlardır. Potasyum yoğunluğu nöronun içinde sodyum yoğunluğu nöronun dışındadır.

Bir hücre diğer bir hücreye elektrik enerjisini bu kimyasal iyonlar vasıtasıyla transfer eder. Bazı iyonlar elektrik ve magnetik kutuplaşmaya sebep olurken bazıları kutuplaşmadan kurtulup hücre zarını açarak iyonların somaya geçmesini sağlar. Zaten sinyallerin bir hücreden diğerine akmasını sağlayan da bu kutuplaşmanın azalması olayıdır. Sinyaller hücrenin etkinliğini belirler. Bir hücrenin etkinliği, hücreye gelen snaps sayısı, snapslardaki iyonların konsantrasyonu, snapsın sahip olduğu güç olmak üzere üç faktöre bağlıdır. Bir nöron sahip olduğu dürtü miktarınca diğer hücreleri

etkiler. Bazı hücreler diğerlerinin dürtülerini pozitif yönde, bazı hücreler de negatif yönde etkiler. İnsan beyni bu şekilde çalışan milyonlarca nöronun bir araya gelmesinden oluşur. Beyinde korteks denilen bölgede her nöronun bir karşılığı vardır. Bir nöronun çıkışı ona bağlı olan bütün nöronlara iletilir. Fakat korteks, işin yapılabilmesi için hangi nöron harekete geçecekse sadece ona komut gönderir.

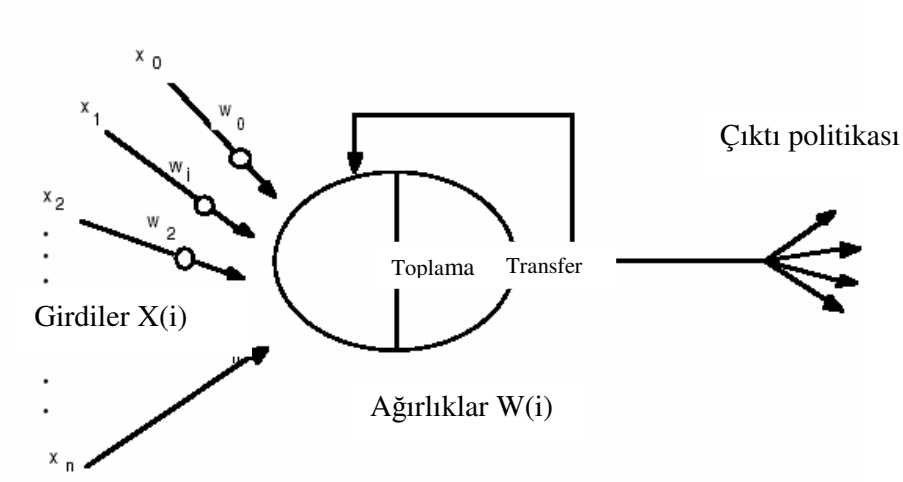
Biyolojik beynin en önemli özelliklerinden birisi de öğrenme olayıdır. İnsanlar ve hayvanlar sürekli olarak içlerinde bulunduğu çevre ile ilişkiler neticesinde bir öğrenme süreci içerisindedirler. Öğrenilen her yeni bilgi, hemen beynin fonksiyonlarını etkileyerek davranışlarda da kendini gösterir. Yapay sinir ağlarını geliştirilmesinde bu özellik esas teşkil eder.

Beynin yoğun bağlantılı yapısı beynin yenilenebilme (plastiklik) özelliğini de beraberinde getirmektedir. Yenileyebilme özelliği ile sinir sistemi kendisini kuşatan çevreye adapte olabilme imkânı bulur. Yenilenebilirlik, nöronlar arasında yeni bağlantıların oluşturulması ve mevcut bağlantıların modifiye edilmesi şeklinde ele alınabilir. Yenilenebilirlik öğrenme kavramı açısında da çok önemlidir. Yenilenebirlirlik nöronların işlem yapabilmesi için çok önemlidir. Çünkü öğrenme süresince yeni bağlantılar oluşturulur ya da bağlantı ağırlıkları değiştirilir. Hatta bazen bazı bağlantılar iptal edilir. Bu ilişki sayesinde yapay nöronlar kullanılarak yapay sinir ağları oluşturulmuştur. Yapay sinir ağları beyinden esinlendiği için benzer yapı gösterirler. Yapay sinir ağları oluşturmak için kullanılan yapay nöronlar beyin nöronlarına kıyasla oldukça ilkel kalırlar. Fakat genel yapı olarak tutarlıdırlar. Yani yapay sinir ağları beynin sadece en temel elemanlarını kopyalamaya çalışmıştır.

### 3.1.1. Sinir Sistemi ile Yapay Sinir Ağlarının Benzerlikleri

Bir nörondan yüzlerce, bazen de binlerce dendrit çıkabilir. Bunların uzunluğu genellikle bir milimetreden daha kısadır. Bazıları ise birkaç milimetre uzunluğa ulaşabilir. Dendritler çevre hücrelerden gelen sinyalleri (impulse) alıp, gövdeye ulaştırırlar. Her nöron, dendritler vasıtasıyla diğer bir çok nöronlardan gelen işaretleri alan ve birleştiren basit bir mikroişleme birimidir. Sinir sisteminde nöronun karşılığı yapay sinir ağlarında işlem elemanına denk gelir. Aksonlar eleman çıkışı olarak, sinapslar da ağırlıklar olarak adlandırılır. Sinir sisteminde dentritlerin işlevi yapay sinir ağlarında toplama fonksiyonu tarafından gerçekleştirilir. Hücre gövdesinin görevi de transfer fonksiyonu tarafından yapılır.

Beyin, sıkışık olarak arabağlaşımlı milyarlarca (yaklaşık olarak 1011) nörondan oluşmaktadır. Her eleman kendi aralarında oldukça çok sayıda nörona (eleman başına yaklaşık olarak 104 bağlantı) bağlanmıştır. Bir nöronun aksonu (çıkış yolu) ayrıştırılmıştır ve bir sinaps olarak adlandırılan bir jonksiyon vasıtasıyla diğer nöronların dendritlerine bağlanmıştır. Bu jonksiyon uçlarındaki iletim doğal olarak kimyasaldır ve işaretin miktarı, akson tarafından serbest bırakılan kimyasalların büyüklüğüne bağlı olarak transfer edilir ve dendritler vasıtasıyla alınır. Bu sinaptik büyüklük, beyin öğrenirken neyin modifiye edildiğini belirtir. Bu sinaps, beynin temel hafıza mekanizmasına dayanarak nöron içerisindeki bilginin işlenmesi ile birleştirilir.




Şekil: 3.1.3. Yapay Nöronun Genel Yapısı

Yapay sinir ağlarında ise yapay nöronlar doğal nöronların dört temel fonksiyonunu simule ederler. Şekil: girdi değerler X(i) matematiksel sembol ile gösterilir. Her bir girdi değeri belirli ağırlıklarla çarpılır. Bu ağırlıklar ise W(i) ile gösterilmektedir. En basit yapı için bu çarpımlar toplanarak transfer fonksiyonuna gönderilir. Transfer fonksiyonundan elde edilen sonuç çıktıya dönüştürülür. Bu elektronik uygulama değişik toplama ve transfer fonksiyonları ile gerçekleştirilir. Kullanılan transfer fonksiyonu ve toplama fonksiyonu farklı ağ yapılarına uygulanabilir. Şekil McCullogh ve Pitts (1943) tarafından tanımlanan biyolojik nöronun matematiksel modelinin gösterimidir. Burada toplama fonksiyonu olarak doğrusal bir fonksiyon ve transfer fonksiyonu olarak da birim adım fonksiyonu kullanılmıştır.

Tablo: 3.1.1. Sinir sistemi ile yapay sinir ağlarının benzerlikleri

| SİNİR SİSTEMİ | YSA SİSTEMİ |
|---|---|
| Neuron | İşlem elemanı |
| Dendrit | Toplama fonksiyonu |
| Hücre gövdesi | Transfer fonksiyonu |
| Aksonlar | Eleman çıkışı |
| Sinapslar | Ağırlıklar |

### 3.2. Yapay Sinir Hücresinin Genel Yapısı ve Elemanları

Nöronlar çok sayıda dentritleri vasıtasıyla giriş uyarısını alır. Dentritlerce alınan bir giriş, harekete geçirici (tetikleyici) veya yasaklayıcı olabilir. Girişler toplanır ve nöron gövdesine yerleştirilir. Bu giriş, belirli bir eşik değerini aştığı zaman, hücre diğer hücrelere aksonu vasıtasıyla bir etki iletir. Bir nöron/işlem elemanı kendine gelen girişleri toplayan ve sadece girişlerin toplamı iç eşik değerini aştığında bir çıkış üreten bir işlem elemanıdır. Bir eşik birimi olarak nöron snapslarındaki işaretleri alır ve hepsini toplar. Eğer toplanan işaret gücü, eşiği geçecek kadar güçlü ise, diğer nöronları ve dentritleri uyarmak üzere akson boyunca bir işaret gönderilir. Kesişen dentritlerde oluşan snapslarca tutulan bütün işaretleri soma toplar. Toplam işaret daha sonra nöronun iç eşik değeri ile karşılaştırılır ve eşik değeri aşmışsa aksona bir işaret yayar. Yapay sinir ağları bu basit nöronların (düğümlerin ya da ünitelerin) bağlanarak bir ağa dönüştürülmesiyle meydana getirilir.

Bir yapay sinir ağı modelinin temel birimi işlem elemanıdır. Girişler dış kaynaklardan veya diğer işlem elemanlarından gelen işaretlerdir. Bu işaretler, kaynağına göre kuvvetli veya zayıf olabileceğinden ağırlıkları da farklıdır. Yapay sinir ağlarında girilen giriş değerlerine önce toplama fonksiyonları uygulanır ve her bir işlem elemanının çıkış (İEÇ) değeri bulunur.

$$İEÇ = \sum_{i=1}^{N} X_i W_{ij} - \theta_i$$

(3.2.1)

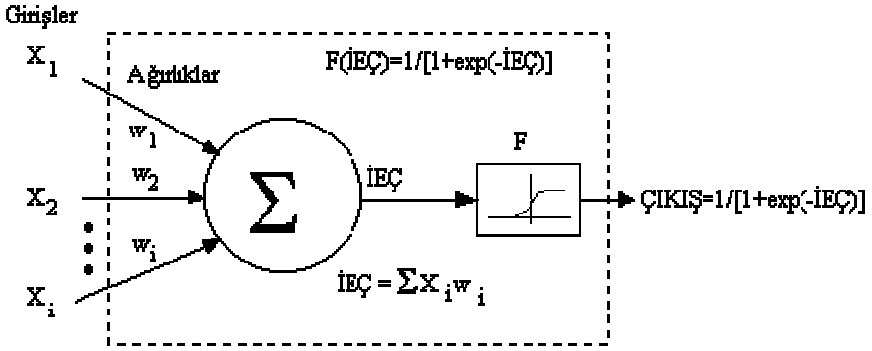
Burada Xi i'inci girişi, Wij j'inci elemandan i'inci elemana bağlantı ağırlığını ve Θi eşik (threshold) değerini göstermektedir. Daha sonra bu çıkış değerleri sigmoidal

aktivasyon fonksiyonuna yani öğrenme eğrisine uygulanır. Sonuçta çıkış değeri aşağıdaki şekilde bulunur.

$$ÇIKIŞ = \frac{1}{1 + e^{-İEÇ}}$$

(3.2.2)

Uygulamalarda, en çok *hiperbolik tanjant* veya *sigmoid* fonksiyonu kullanılmaktadır. Şekil: işlemci eleman çıkışında kullanılan sigmoid fonksiyona göre çıkış değerinin hesaplanması gösterilmiştir. Bu işlemci elemanın çıkış değeri diğer işlemci elemanlarına giriş veya ağın çıkış değeri olabilir.




Şekil: 3.2.1. İşlemci elemanı

Biyolojik sinir ağlarının sinir hücreleri olduğu gibi yapay sinir ağlarının da yapay sinir hücreleri vardır. Yapay sinir hücreleri mühendislik biliminde "proses elemanları" olarakta adlandırılır. Her proses elemanının beş temel elemanı vardır. Bunlar ağın girdileri, ağırlıkları, toplama fonksiyonu, aktivasyon fonksiyonu ve çıktısıdır.

### 3.2.1. Girdiler

Girişler yapay sinir hücresine dışarıdan girilen veriler ve diğer hücrelerden veya kendi kendisinden gelen veriler oluşturur. Girişler ($x_{1,X2,}$ $x_{3,}$ $x_{4,}$ $x_{5,}$ ….$x_{n}$) çevreden aldığı bilgiyi sinire getirirler. Girişler kendinden önceki sinirlerden veya dış dünyadan sinir ağına gelebilir. Bir sinir genellikle gelişi güzel bir çok girdileri alır.

Yapılacak olan uygulamaya göre sonucu en çok etkileyen faktörler giriş olarak uygulamaya sunulmalıdır. Yazılım maliyet tahminlemesinde yazılımın maliyetini en çok etkileyen faktörler yazılan kod satır sayısı, projenin büyüklüğü, çalışanların yetkinliği, hatalı kod satır sayısı, müşterinin teknik bilgi yeterliliği gibi faktörler giriş değerleri olarak alınabilir. Yapay sinir ağına verilen girişlerin ön işlemden geçirilerek ağa sunulması ağın eğitimini olumlu yönde etkiler. Veriler ağa sunulmadan önce normalize edilmelidir. Ağın eğitilememesi durumun da ağa verilen giriş verileri tekrar gözden geçirilmelidir.

### 3.2.2. Ağırlıklar

Ağa sunulan giriş değerlerin bir de ağırlıkları vardır. Ağırlıklar ($w_1, w_2, w_3, w_4, w_5, \ldots. w_n$) yapay sinir tarafından alınan girişlerin sinir üzerindeki etkisini belirleyen uygun katsayılardır. Yapay sinir ağına verilen her bir girişin kendine ait bir ağırlık değeri vardır.Öğrenme işlemi ağın ağırlıklarının değişmesi ile gerçekleşir. Bir ağın giriş ağırlık değerinin büyük olması o girişin önemli olduğu anlamına gelmez. Aynı şekilde ağırlık değerinin küçük olması da o girişin daha az etkili olmasıyla ilgi bilgiyi temsil etmez. Ağırlıklar değişken veya sabit değerler olabilir. Ağa sunulan ağırlığın değeri pozitif veya negatif olabilir. İşareti ağırlığın uygulamaya olan negatif veya pozitif etkisini simgeler.

Çok katmanlı algılayıcı ağının proses elemanlarını birbirine bağlayan bağlantıların ağırlıklarının başlangıç değerinin atanması da ağın performansı ile yakından ilgilidir. Genel olarak ağırlıklar belirli aralıklarla atanmaktadır. Bu aralık eğer büyük tutulursa ağın yerel çözümler arasında sürekli dolaştığı küçük olması durumunda ise öğrenmenin geç gerçekleştiği görülmüştür. Bu değerlerin atanması hakkında henüz standart bir yöntem yoktur. Ağın ağırlıkları için sıfır ve bir arasında başlangıç değerlerinin atanması önerilmiştir fakat bu her problem türü için uygun sonuç vermemektedir. Bu tamamen öğrenilmesi istenen problemin niteliğine bağlıdır. Burada yine tecrübelerden yararlanılmaktadır. Öğrenmeyen bir ağın başlangıç değerlerinin değiştirilmesi ağın öğrenmesine neden olabilir. Bazı durumlarda ağın öğrenmesi zor bir olay üzerinde eğitildiğinde, ağın öğrenmesi başlangıç noktalarının değiştirilmesi ile mümkün olmayabilir. O nedenle her olumsuz eğitimde başlangıç değerlerinde aramak doğru olmayabilir. Olayı bütün içinde ele alarak diğer etkenleride gözden geçirmek gerekir.

### 3.2.3. Toplama Fonksiyonu

Bu fonksiyon bir hücreye gelen net girdiyi hesaplar. Toplama işlevi $v_i$, sinirde her bir ağırlığın ait olduğu girişlerle çarpımının toplamlarını eşik değeri ile toplayarak etkinlik işlevine gönderir. Bazı durumlarda toplama işlevi bu kadar basit bir işlem yerine en az (min), en çok (max) çoğunluk veya birkaç normalleştirme algoritması gibi çok daha karmaşık olabilir.

Bu net girdiyi hesaplamak için bir çok yöntem mevcuttur. Fakat yaygın olarak kullanılana yöntem ağırlıklı toplam yöntemidir. Burada her giriş değeri ağırlık değeri ile çarpılarak toplanır. Böylece ağa gelen net girdi değeri bulunmuş olur. Uygulama için ağın bütün işlem elemanlarının toplama fonksiyonunu kullanması gerekmemektedir. Ağın tasarımcısına ve probleme bağlı olarak belirli bir işlem elemanları grubu için toplama fonksiyonu kullanılırken geri kalanlar için farklı bir fonksiyon kullanılabilir.

### 3.2.4. Aktivasyon Fonksiyonu

Transfer veya işaret fonksiyonları olarak da adlandırılan aktivasyon (eşik) fonksiyonları, muhtemel sonsuz giriş kümesine sahip işlem elemanlarından önceden belirlenmiş sınırlar içinde çıkışlar üretirler. Bu fonfsiyon hücreye gelen net girdiyi işleyerek hücrenin bu girdiye karşılık üreteceği çıktıyı belirler. Toplama fonksiyonunda olduğu gibi aktivasyon fonksiyonu olarakta çıktıyı hesaplamak içinde değişik formüller kullanılabilir. Örneğin çok katmanlı algılayıcılar bu fonksiyonun türevinin alınabilmesini şart koşmaktadır. Toplama fonksiyonunda olduğu gibi aktivasyon fonksiyonunda da ağın bütün proses elemanlarının aynı aynı fonksiyonu kullanması gerekmez. Ağın hangi fonksiyonu kullanacağına dair somut bir kural mevcut değildir. Ağı tasarlayan kullanıcının deneyimlerine bırakılmıştır. Beş tane yaygın olarak kullanılan eşik fonksiyonu vardır. Bunlar liner, rampa, basamak, sigmoid ve tanh(x) fonksiyonlarıdır. Şekil: de bu fonksiyonlar gösterilmiştir.

*Liner Fonksiyonun Denklemi:*

$$f(x) = \alpha . x$$

$\alpha$ işlem elemanının x aktivitesini ayarlayan reel değerli bir sabittir. Lineer fonksiyon $(-\tau, +\tau)$ sınırları arasında kısıtlandığında rampa eşik fonksiyonu olur .

*Rampa Fonksiyonun Denklemi:*

$$f(x)=\begin{cases}+\tau & :\text{Eğer x} \geq \tau \text{ ise} \\ x & :\text{Eğer } |\text{x}| < \tau \text{ ise (yani} -\tau < x < \tau) \\ -\tau & :\text{Eğer x} \geq -\tau \text{ ise}\end{cases} \qquad (2.3)$$

+τ /-τ işlem elemanının maksimumu (minimumu) çoğu zaman doyma seviyesi olarak adlandırılan çıkış değeridir. Eğer eşik fonksiyonu bir giriş işaretine bağlı ise yaydığı +τ giriş toplamı pozitif, bağlı değilse eşik basamak fonksiyonu |-τ | olarak adlandırılır.

*Basamak Fonksiyonun Denklemi:*

$$f(x)=\begin{cases}+\tau & : \text{Eğer x} > 0 \text{ ise} \\ -\delta & : \text{Diğer durumlar}\end{cases}$$

şeklindedir.

*Sigmoid Fonksiyonun Denklemi:*

Diğer bir eşik fonksiyonu ise sigmoid fonksiyonudur ve bu tez çalışmasında eşik fonksiyonu olarak kullanılmıştır. Yatık S biçimindeki sigmoid fonksiyonu; seviyeli, doğrusal olmayan(non-lineer) çıkış veren, sınırlı, monoton artan bir fonksiyondur ve denklemi;

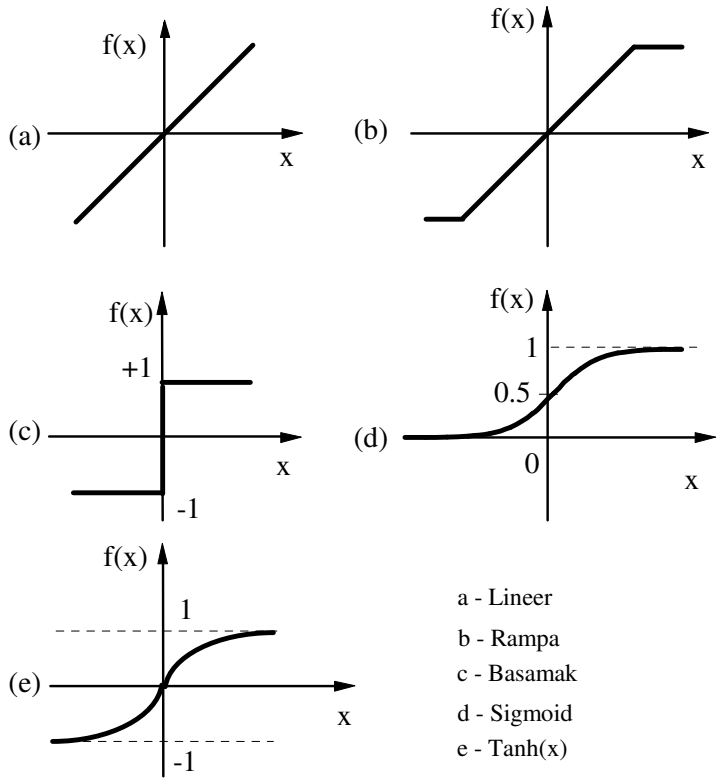
$$\text{f(x)}=\frac{1}{1+e^{-x}}$$

biçimindedir.

*Tanh(x) Fonksiyonun Denklemi:*

$$\tanh(\text{x})=\frac{e^{\text{x}}-e^{-\text{x}}}{e^{\text{x}}+e^{-\text{x}}}=\frac{e^{2\text{x}}-1}{e^{2\text{x}}+1}$$

şeklindedir. Her işlem elemanı kendisine gelen bir yerel veriye göre, kendisini ayarlayarak bütün YSA nın bilgi bölgesini öğrenmesini sağlar. Yukarıdaki transfer fonksiyonlarını kullanabilmek için, giriş verilerinin gerçek değerlerinin “0” ile “1” arasındaki bir reel sayıya dönüştürülmesi (normalizasyon) gerekir.(Bekir Karlık Ders Notları)




Şekil: 3.2.2. Kullanılan Eşik Fonksiyonları

### 3.2.5.Hücrenin Çıktısı

Çıkış işlevi etkinlik işlevi sonucunun dış dünyaya veya diğer sinirlere gönderildiği yerdir. Biri sinirin bir tek çıkışı vardır. Bu çıkış kendinden sonra gelen herhangi bir sayıdaki diğer sinirlere giriş olabilir. Yalnız her bir düğümde bir çıkış işaretine izin verilir. Bu işaret yüzlerce sinir hücresinin girişi olabilir. Bu durum biyolojik sinirde de aynı şekilde gerçekleşir. Düğüm çıkışı etkinlik işlevinin sonucuna eşdeğerdir. Fakat bazı ağ yapıları komşu düğümler arasında yarışma oluşturmak için etkinlik sonuçlarını düzenleyebilir. Böylece yarışmacı girişler hangi düğümün öğrenme ya da uyma işlemine katılacağına karar verilmesinde yardımcı olur.

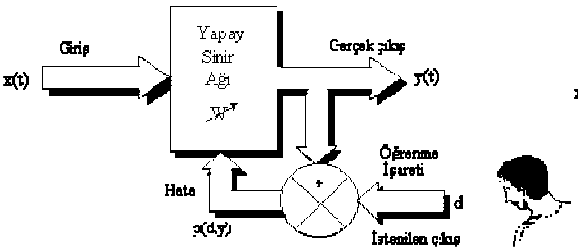
Ağ şeklinde gösterildiğinde bir proses elemanının birden fazla çıktısı varmış gibi görünür. Aslında bir proses elemanından çıkan tek bir çıktı değeri vardır. Aynı değer bir çok proses elemanına girdi değeri olarak gitmektedir.Bu elde edilen çıktı değerlerinin nümerik gösterimi gerçekleştirilmezse çıktı değerleri ile beklenen değerler arasındaki hatayı bulmak mümkün olmaz. Girdilerde olduğu gibi çıktılarda da nümerik

gösterim problemden probleme değişmektedir. Bir problem için birden fazla yöntem kullanılarak nümerik gösterim sağlanır. Bunlardan hangisinin en iyisi olduğunu belirleyen bir standart mevcut değildir. Önemli olan probleme uygun olan metodu belirleyebilmektir.

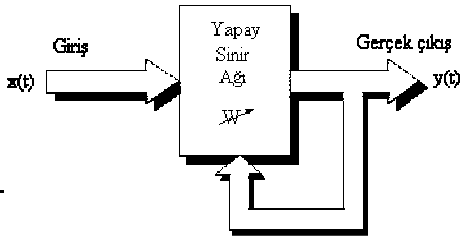
# 4. YAPAY SİNİR AĞLARINDA ÖĞRENME ve EĞİTME ALGORİTMALARI

## 4.1. Yapay Sinir Ağlarında Öğrenme

Öğrenme kuralı Hebbian öğrenme kuralı denilen basit bir modele dayanır. Hebbian öğrenme kuralı temel olarak eğer iki düğüm aynı zamanda etkin ise aralarındaki bağ gücü artar kuramına dayanır(Elmas,2003). Öğrenmenin amacı her bir düğümün girişlerindeki değişken bağlantı ağırlıklarını derlemektir. İstenen bazı sonuçları elde etmek için giriş bağlantılarının ağırlıklarını değiştirme işlemi uyma işlevi olarak adlandırılabildiği gibi öğrenme kipi olarakta adlandırılabilir. Yapay sinir ağları danışmalı öğrenme, danışmasız öğrenme ve takviyeli öğreme olarak üç sınıf altında toplanabilir. Danışmalı öğrenmede bir "öğretmene" veya "danışmana" ihtiyaç vardır. Widrow-Hoff tarafından geliştirilen delta kuralı ve Rumelhart ve McClelland tarafından geliştirilen genelleştirilmiş delta kuralı veya geri besleme (back propagation) algoritması danışmanlı öğrenme algoritmalarına örnek olarak verilebilir. Danışmalı öğrenmede, her bir örnekleme zamanında giriş uygulandığında sistemin arzu edilen cevabı y eğitici tarafından sağlanır. Arzu edilen çıkış y ile sinir ağı çıkışı o arasındaki fark hata ölçüsüdür ve ağ parametrelerini güncellemekte kullanılır. Ağırlıkların güncellenmesi süresince eğitici ödüllendirme-cezalandırma şemasını ağa uygulayarak hatayı azaltır. Bu öğrenme modelinde giriş ve çıkış örnekleri kümesi eğitim kümesi olarak adlandırılır




Şekil4.1.1. Danışmalı öğrenme




Şekil 4.1.2.Danışmasız öğrenme

Danışmasız öğrenmede ise girişe verilen örnekten elde edilen çıkış bilgisine göre ağ sınıflandırma kurallarını kendi kendine geliştirmektedir. Bu öğrenme algoritmalarında, istenilen çıkış değerinin bilinmesine gerek yoktur. Öğrenme süresince sadece giriş bilgileri verilir. Ağ daha sonra bağlantı ağırlıklarını aynı özellikleri gösteren desenler

(patterns) oluşturmak üzere ayarlar. Grossberg tarafından geliştirilen ART (Adaptive Resonance Theory) veya Kohonen tarafından geliştirilen SOM (Self Organizing Map) öğrenme kuralı danışmansız öğrenmeye örnek olarak verilebilir. Eğiticisiz öğrenmede, eğiticili öğrenmedeki gibi arzu edilen y çıkışları bilinmemektedir. Bu yüzden kesin bir hata bilgisini ağın davranışını değiştirmekte kullanmak mümkün değildir. Cevabın doğruluğu veya yanlışlığı hakkında bilgi sahibi olunmadığı için öğrenme, girişlerin verdiği cevaplar gözlenerek başarıya ulaşılır. Aslında eğiticisiz öğrenme demek doğru değildir, çünkü eğiticisiz öğrenme gerçekte mümkün değildir. Eğiticinin her öğrenme adımında dahil olmamasına rağmen, amaçları ayarlamaktadır.

Takviyeli öğrenme kuralı danışmanlı öğrenmeye yakın bir metottur. Denetimsiz öğrenme algoritması, istenilen çıkışın bilinmesine gerek duymaz. Hedef çıktıyı vermek için bir "öğretmen" yerine, burada YSA'ya bir çıkış verilmemekte fakat elde edilen çıkışın verilen girişe karşılık iyiliğini değerlendiren bir kriter kullanılmaktadır. Optimizasyon problemlerini çözmek için Hinton ve Sejnowski'nin geliştirdiği Boltzmann kuralı takviyeli öğrenmeye örnek olarak verilebilir. Takviyeli öğrenmede, ağın davranışının uygun olup olmadığını belirten bir öz yetenek bilgisine ihtiyaç duyulur. Bu bilgiye göre ağırlıklar ayarlanır. Gerçek zamanda öğrenme yöntemi olup deneme-yanılma esasına göre sinir ağı eğitilmektedir.

Bilginin kurallar şeklinde açıklandığı klasik uzman sistemlerin tersine, YSA gösterilen örnekten öğrenerek kendi kurallarını oluşturur. Öğrenme; giriş örneklerine veya bu girişlerin çıkışlarına bağlı olarak ağın bağlantı ağırlıklarını değiştiren veya ayarlayan öğrenme kuralı ile gerçekleştirilir. Öğreticisiz öğrenmede her giriş işareti için istenen çıkış sisteme tanıtılır ve YSA giriş/çıkış ilişkisini gerçekleştirene kadar kademe kademe kendini ayarlar. Günümüzde kullanılan birçok öğrenme kuralı vardır. Bilinen ve en çok kullanılan öğrenme kuralları şunlardır:

- Hebb Kuralı
- Hobfield Kuralı
- Delta Kuralı
- Kohonen Kuralı
- Filtreleme

### 4.1.1. Hebb Kuralı

Bilinen en eski öğrenme kuralıdır. 1949 yılında geliştirilen bu kural diğer öğrenme kurallarının temelini oluşturmaktadır. Bu kurala göre bir hücre diğer bir

hücreden bilgi alırsa ve her iki hücrede aktif ise her iki hücrenin arasındaki bağlantı kuvvetlendirilmelidir. Bir hücre kendisi aktif ise bağlı olduğu hücreyi aktif yapmaya pasif ise bağlı olduğu hücreyi pasif hale getirmeye çalışır. Diğer öğrenme kurallarının hepsi bu felsefeyi baz alarak geliştirmişlerdir.

### 4.1.2. Hobfield Kuralı

Bu kural kuvvetlendirme veya zayıflamanın genliğini belirleyebilmesi istisnası haricinde Hebb kuralıyla benzerdir. Buna göre eğer istenilen çıkış ve girişin her ikiside aktif vaya her ikiside durgun ise bağlantı boyutlarını öğrenme oranı kadar artırır, aksi halde boyutu öğrenme oranı kadar azaltır. Kısaca yapay sinir ağı elemanlarının bağlantılarının ne kadar zayıflatılması yada kuvetlendirilmesi gerektiğini belirler. Eğer beklenen çıktı ve girdiler ikiside aktif ise öğrenme katsayısı kadar ağırlık değerleri kuvvetlendirilir. Eğer beklenen çıktı ve girdiler ikiside pasif ise öğrenme katsayısı kadar ağırlık değerleri zayıflattırılır.

### 4.1.3. Delta Kuralı

En çok kullanılan kurallardan biri olan delta kuralı Hebb kuralının daha gelişmişidir. Bu kural bir sinirin gerçek çıkışı ile istenilen çıkış değeri arasındaki farkı azaltmak için giriş bağlantı güçlerini sürekli olarak geliştirme fikrine dayanır. Bu kural ağ hatasının karesini minimize etmek için bağlantı boyutlarını değiştirir. Hata bir önceki katmana geri çoğaltılır. Her bir zaman dilimi için bir hata şeklinde bu geri çoğaltma işlemi ilk katmana ulaşılıncaya kadar devam eder. Bu tip ağ ileri beslemeli ağ olarak isimlendirilir. Geri yayılım adını bu hata terimlerinin toplama yönteminden türetir. Bu kural aynı zamanda en küçük ortalamalar karesi(LSM Least Square Mean) olarak da adlandırılır.

Delta-Bar-Delta (DBD) çok katmanlı perseptronlarda bağlantı ağırlıklarının yakınsama hızını arttırmak için kullanılan bir sezgisel yaklaşımdır. Deneysel çalışmalar, ağırlık uzayının her boyutunun tüm hata yüzeyi açısından tamamen farklı olabileceğini göstermiştir. Hata yüzeyindeki değişimleri açıklamak için, özellikle, ağın her bağlantısı kendi öğrenme katsayısına sahip olmalıdır. Bu düşünce, tek ağırlık boyutu için uygun adım büyüklüğü, tüm ağırlık boyutları için uygun olmayabilir. Bununla birlikte, her bir bağlantıya bir öğrenme oranı atanarak ve bu öğrenme oranının zamanla değişmesine izin verirsek, yakınsama zamanını azaltmak için daha çok serbestlik derecesi sağlanmış olur.

İleri beslemeli YSA yapıları çoğu zaman karmaşıktırlar. Ağdaki her bağlantı için en uygun öğrenme katsayıları kümesini belirlemek oldukça zaman alıcı olabilir. Eğimin geçmişteki değerlerini kullanarak, yerel hata yüzeyinin eğriliğini çıkarmak için sezgisellik uygulanabilir. Bir bağlantı için ağırlık değişimlerinin işareti, birkaç ardışık zaman adımları sırayla değiştiği zaman, bağlantı için öğrenme oranı azalmalıdır. Bağlantı ağırlık boyutu hata yüzeyi ile ilgili büyük bir eğriliğe sahiptir. Bağlantı ağırlık değişimleri bir kaç ardışık zaman adımları için aynı işarete sahip olduğundan bağlantı için öğrenme oranı arttırılmalıdır. Ağırlık boyutu hata yüzeyi ile ilgili küçük bir eğriliğe sahiptir ve yüzey önemli bir mesafe için aynı doğrultudaki eğime göre devam eder.

### 4.1.4. Kohonen Kuralı

Bu kurala göre ağın elemanları ağırlıklarını değiştirmek için birbirleri ile yarışırlar. En büyük çıktıyı üreten hücre olmakta ve bağlantı ağırlıkları değiştirilmektedir. Bu o hücrenin yakındaki hücrelere karşı daha kuvvetli hale gelmesi demektir. Hem kazanan elemanların hem de komşuları sayılan elemanların ağırlıklarını değiştirmesine izin verilmektedir(Öztemel, 2003).

## 4.2. Yapay Sinir Ağlarında Eğitme Algoritmaları

Eğitme algoritmaları YSA nın ayrılmaz bir parçasıdır. Eğitme algoritması, eldeki problemin özelliğine göre öğrenme kuralının YSA ya nasıl adapte edileceğini belirtir. Kullanımı yaygın olan üç çeşit eğitme algoritması vardır.

### 4.2.1. Öğreticili Eğitme

Öğreticili eğitmede, elde doğru örnekler vardır. Yani eğitme örneklerinin tamamı için $(x_1, x_2, ..., x_n)$ şeklindeki giriş vektörünün; $(y_1, y_2, ..., y_n)$ şeklindeki çıkış vektörü, tam ve doğru olarak bilinmektedir. Herbir $(x_1, y_1)$, $(x_2, y_2)$, ...., $(x_n, y_n)$ çifti için, ağ doğru sonuçları verecek şekilde, seçilen bir öğrenme kuralı yardımıyla eğitilir.

### 4.2.2. Skor ile Eğitme

Skor ile eğitmede giriş işaretlerine karşılık gelen çıkış işaretleri tam olarak bilinmemektedir. Çıkış işareti yerine skor verilir ve ağın değerlendirmesi yapılır. Özellikle kontrol uygulamaları için idealdir. Çeşitli maliyet fonksiyonları kullanılır.

### 4.2.3. Kendini Düzenleme ile Eğitme

Probleme ait veri girişlerine karşın çıkışlar mevcut değildir. Bu yüzden bu tür bir eğitme, giriş verilerini gruplandırarak eğittikten sonra verilen herhangi bir girişin eğitme sınıflarından hangi sınıfa ait olduğunu gösterebilir. Kendini düzenleyen ağ, giriş işaretine göre kendini düzenleyerek organize eder. Bu eğitme tipi olasılık yoğunluk fonksiyonlarına, sınıflandırma ve şekil tanıma problemlerine uygulanabilir.

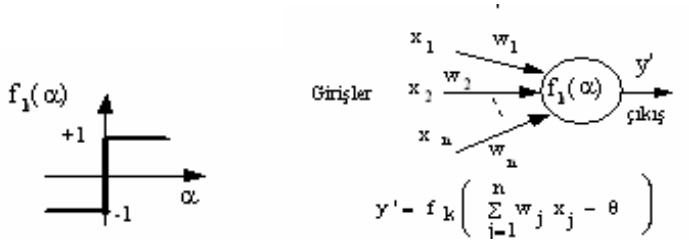
Ne tür eğitme yöntemi kullanılırsa kullanılsın, herhangi bir ağ için gerekli karakteristik özellik, ağırlıkların verilen eğitme örneğine nasıl ayarlanacağının belirtilerek öğrenme kuralının oluşturulmasıdır. Öğrenme kuralının oluşturulması için bir örneğin ağa defalarca tanıtılması gerekebilir. Öğrenme kuralı ile ilişkili parametreler (öğrenme katsayısı vb.) ağın zaman içinde gelişme kaydetmesiyle değişebilir. (Bekir Karlık, Ders Notları)

# 5. PERCEPTRON ve ÇOK KATMANLI PERCEPTRONLARIN YAPISI

## 5.1. Perceptron

Perceptron ağı ilk olarak 1943 yılında Mc Culloch ve Pitts tarafından ortaya atılmış ve 1960'larda sinir ağları olarak F.Rosentblatt tarafından önerilmiştir. Perceptronlar son derece sınırlı olmalarına rağmen en eski sinir ağlarından biridir. Mevcut algoritmalarla (örneğin geri yayılım algoritması gibi) yakın ilişki içinde olduklarından çok ilgi çekicidirler. En basit yapay sinir ağı örneği tek katmanlı ve tek sinirli perceptrondur. Bu yapay sinir ağlarının birden fazla girişi ve tek bir çıkışı mevcuttur. Çıkış değeri artı bir veye eksi bir olarak değerlendirilir. Perceptronlar genellikle nesneleri iki ayrı sınıfa ayırmak için kullanılmıştır.

Perceptron girişlerin ağırlıklı toplamını eşik değeri ile karşılaştırır. Ağırlıklı toplam giriş değerinden büyükse sonuç artı bir olarak alınır. Aksi durumda sonuç eksi bir olarak alınır. Karar bölgemiz iki nesneden oluşuyorsa artı bir değeri birinci bölgeyi eksi bir değeri ise ikinci bölgeyi ifade eder. Perceptronun en etkileyici yanı verilen girdi çıktı ilişkisini öğrenebiliyor olmasıdır. Fakat yapılan araştırmalar sonucu tek katmanlı sinir ağlarının çözemediği problemleri çok katmanlı ağların çözebildiği ortaya koyulmuştur. Örneğin bir XOR işlemi tek katlı perceptron ile çözülemezken çok katmalı yapay sinie ağı ile çözüme ulaşılabilmektedir. Giriş katmanının doğrudan çıkış katmanına bağlanmasıyla çözülemeyen XOR problemi arada gizli katman kullanımıyla çözülebilir bir problem haline gelmiştir.(Elmas,2003)




Şekil: 5.1.1 Tek katmanlı perceptron

Tek katmanlı perceptron, bir düzlemi ikiye ayıran karar bölgeleri oluşturur. İki katmanlı bir perceptron, açık ve kapalı konveks bölgeler oluşturabilir. Üç katmanlı perceptron ise kompleks karar bölgelerini oluşturur. Perceptron gibi ileri yayılımlı ağlarda, iki adet gizli katmandan fazlasına gerek yoktur. Üç katmanlı perceptronlarda karar bölgeleri arasında bağlantı yoksa veya bu bölgeler bir konveks alan ile

oluşturulamıyorlarsa, ikinci katmandaki düğüm sayısı bir'den büyük olmalıdır. En kompleks durum için, ikinci katmandaki düğümlerin sayısı, giriş dağılımlarındaki ayrı bölgelerin sayısı kadar olmalıdır. Benzer şekilde birinci katmandaki düğüm sayıları da herbir ikinci katman düğümünce oluşturulan herbir konveks bölge için üç veya daha fazla kenar sağlanacak şekilde yeterli sayıda olmalıdır.

### 5.2. Çok Katmanlı Perceptron

Tek katmanlı perceptronlarda problemin çözümü bir düzlemi ikiye ayıran karar bölgelerinden meydana gelmekteydi. Yani çıktıların arasına bir doğru veya doğrular çizilerek iki veya daha fazla sınıfa ayrılabiliyordu. Fakat problemin çözümü lineer değilse tek katmanlı perceptronlar yetersiz kalmaktadır. Fakat günlük olayların özellikle günümüzün dinamik sektörü göz önüne alınırsa problemlerin çözümü genellikle lineer değildir. XOR probleminin tek katmanlı perceptronla çözülememesi yapılan çalışmaları bir süre duraklatmıştır. Daha sonra Rumelhart tarafından XOR problemi çok katmanlı perceptron modeli ile çözüme ulaştırıldıktan sonra gelişmeler tekrar ivme kazanmıştır. Çok katmanlı perceptron modeline backpropogation network adıda verilir (Raul Rojas, 1996). Çok katmanlı perceptronlar günümüz problemlerinin bir çoğunun çözümünü kolaylaştırmıştır. Özellikle sınıflandırma, tanımlama, genelleme yapmayı gerektiren problemlere çözüm üretebilmiştir. Bu model delta öğrenme kuralı denilen bir yöntemi kullanmaktadır. Temel amacı ağın beklenen çıktısı ile ürettiği çıktı arasındaki hatayı en aza indirmektir. Bunu hatayı ağa yayarak gerçekleştirdiği için bu ağa hata yayma ağı da denilmektedir.

#### 5.2.1. Çok Katmanlı Perceptron Yapısı

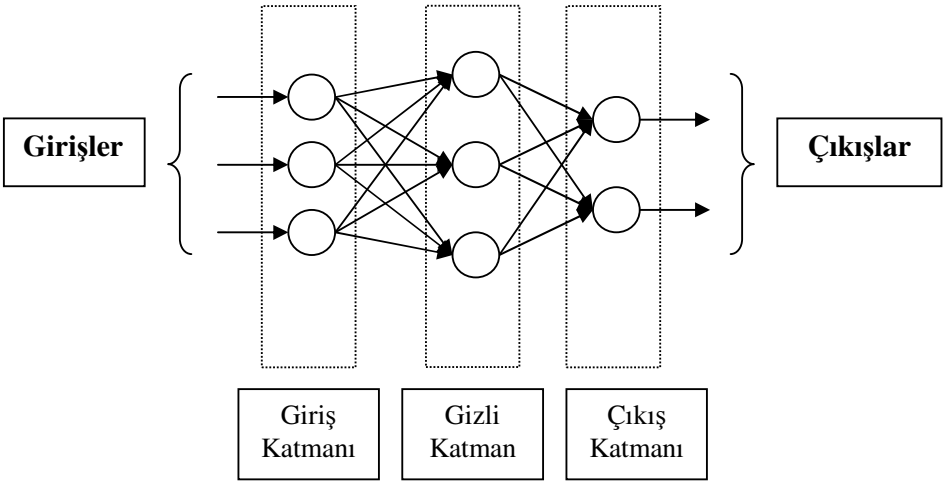
Çok katmanlı perceptronlar aşağıdaki şekilde görüldüğü gibi girdi katmanı arakatman ve çıktı katmanından meydana gelir. Girdi katmanı dış dünyadan gelen girdileri alarak ara katmana gönderir. Bu katmanda bilgi işleme olmaz. Gelen her bilgi geldiği gibi bir sonraki katmana gider. Birden fazla girdi gelebilir. Her proses elemanının bir tane girdisi ve bir tane çıktı elemanı vardır. Bu çıktı bir sonraki katmanda bütün proses elemanlarına gönderilir. Yani girdi katmanındaki her proses elemanı bir sonraki katmanda bulunan proses elemanlarının hepsine bağlıdır.

Ara katmanlar girdi katmanından gelen bilgileri işleyerek bir sonraki katmana gönderir. Çok katmanlı perceptron yapısında birden fazla ara katman ve her katmanda

birden fazla proses elemanı olabilir. Ara katmandaki her proses elemanı bir sonraki katmandaki bütün proses elemanlarına bağlıdır.

Çıktı katmanı ara katmandan gelen bilgileri işleyerek ağa girdi katmanından verilern girdilere karşılık ağın ürettiği çıktıları belirleyerek dış ortama gönderir. Bir çıktı katmanında birden fazla proses elemanı olabilir. Her proses elemanı bir önceki katmanda bulunan bütün proses elemanlarına bağlıdır. Her proses elemanın sadece bir çıktısı vardır.

Çok katmanlı perceptronda bilgiler girdi katmanından ağa sunulur. Bu girdi verileri ara katmanlardan geçerek çıktı katmanına gider ve ağa sunulan girdilere karşılık ağın cevabı dış ortama verilir. Çok katmanlı perceptronda öğretmenli öğrenme straejisi kullanılır. Ağa probleme uygun şekilde belirlenen giriş verileri ile bu örneklerden elde edilen yani beklenen çıkış değerleri ağa verilir. Ağ kendisine sunulan örneklerden genellemeler yaparak uygulama için bir sonuç kümesi oluşturur. Daha sonra eğitilen ağa gönderilen benzer örnekler için bu çözüm uzayı sonuçlar ve çözümler üretebilmektedir.




Şekil5.2.1. Çok katmanlı perceptron yapısı

### 5.2.2. Çok Katmanlı Perceptronun Öğrenme Kuralı

Çok katmanlı ağlarda öğretmenli öğrenme yöntemine göre çalışırlar. Çok katmanlı perceptronla için ağa uygulamayı temsil eden örneklerin girişleri ve bu girişler karşısında elde edilen sonuçlar ağa verilir. Bu ağların öğrenme kuralı en küçük kareler yöntemine dayalı olan Delta Öğrenme kuralının genelleştirilmiş halidir. Dolayısıyla öğrenme kuralına genelleştirilmiş delta kuralı da denilmektedir. Ağın

öğrenebilmesi için örneklerden oluşan bir eğitim setine ihtiyaç vardır. Bu set içinde var olan her örnek için hem girdiler hem de bu girdilere karşılık ağın üretmesi gereken çıktılar vardır. Delta kuralı iki safhadan oluşur(Öztemel, 2003).

1- İleri doğru hesaplama: Ağın çıktısını hesaplama safhasıdır
2- Geriye doğru hesaplama: Ağırlıkları değiştirme safhasıdır.

### 5.2.2.1. İleri doğru Hesaplama

Yapay sinir ağlarında ileri doğru hesaplamanın gerçekleştirilmesi için öncelikle uygulamayı sağlayacak örnek setinin oluşturulması gerekir. Eğitim oluşturulan örnek setinin birinin girdi katmanından ağa sunulması ile başlar. Giriş katmanında herhangi bir bilgi işlemi gerçekleştirilmez. Giriş katmanına gelen veriler ara katmana hiçbir değişiklik yapılmadan gönderilir. Girdi katmanındaki k. proses elemanının çıktısı:

$$Ç^{i}_{k} = G_k$$

ile belirlenir. Ara katmandaki her proses elemanı girdi katmanındaki bütün proses elemanlarından gelen bilgileri bağlantı ağırlıklarının (A1, A2,A3…) etkisi ile alır. Önce ara katmandaki proses elemanlarına gelen net girdi ($NET_j^a$) aşağıdaki formül ile hesaplanır.

$$NET_j^a = \sum_{k=1}^{n} A_{kj} \cdot Ç^{i}_{k}$$

Bu uygulama da sigmoid fonksiyonu kullanılmıştır. Fakat burada türevi alınabilen başka aktivasyon fonksiyonlarıda kullanılabilir. $A_{kj}$ k. Girdi katmanı elemanını j. ara katman elemanına bağlayan bağlantının çıktısı ise bu net girdinin aktivasyon fonksiyonundan geçirilmesi ile hesaplanır. Sigmoid fonksiyonunun kullanılması halinde burada ki çıktı:

$$Ç^{a}_{j} = 1/ (1+e^{-NET^{a}_{j} + ß^{a}_{j}})$$

şeklinde olacaktır. Buradaki $ß_j$ ara katmanda bulunan j. elemana bağlanan eşik değer elemanın ağırlığını göstermektedir. Bu eşik değeri ünitesinin çıktısı sabit olup bire eşittir. Ağırlık değeri ise sigmoid fonksiyonunun oryantasyonunu belirlemek üzere

konulmuştur. Eğitim sırasında ağ bu değeri kendisi belirlemektedir. Ara katmanın bütün proses elemanları ve çıktı katmanının proses elemanları ve çıktı katmanın proses elemanlarının çıktıları aynı şekilde kendilerine gelen NET girdinin hesaplanması ve sigmoid fonksiyonundan geçirilmesi sonucu belirlenir. Çıkış değerleri hesaplannca ağın ileri hesaplama işlemi tamamlanmış olur.(Öztemel, 2003)

#### 5.2.2.2 Geriye Doğru Hesaplama

Uygulamalarda yaygın olarak kullanılan öğretme algoritması geriye yayılım algoritmasıdır. Anlaşılması kolay ve matematiksel olarak ispatlanabilir olmasından tercih edilir. Bu algoritma, hataları geriye doğru çıkıştan girişe azaltmaya çalışmasından dolayı geri yayılım ismini almıştır. Tipik çok katlı geri yayılım ağı, daima; bir giriş tabakası, bir çıkış tabakası ve en az bir gizli tabakaya sahiptir. Gizli tabakaların sayısında teorik olarak bir sınırlama yoktur. Fakat genel olarak bir veya iki tane bulunur.

Ağa sunulan girdi için ağın ürettiği çıktı ağın beklenen çıktıları (B1,B2,B3…) ile karşılaştırılır. Bunların arasındaki fark hata olarak kabul edilir. Bu hatanın düşürülmesi asıl amaçların içerisindedir. Bu amaçla geriye doğru hesaplamada bu hata ağın ağırlık değerlerine dağıtılarak bir sonraki iterasyonda hatanın azaltılmasını sağlar. Çıktı katmanında m. proses elemanı için oluşan hata($E_m$)

$$E_{m=} B_m - Ç_m$$

olacaktır. Bu sadece bir proses elemanı içn olan hatadır. Çıktı katmanı için oluşan toplam hatayı (TH) bulmak için bütün hataların toplanması gerekir. Bazı hata değerleri negatif olduğundan toplamın sıfır olmasını önlemek için ağırlıkların karesi alınarak toplanır. Toplam değerin ise karekökü alınır.(Chen,1990). Elde edilen değerin minimum olması ağın eğitilmesindeki asıl amaçtır.

$$TH = 0{,}5 \sum_{k=1}^{n} E_m^{\,2}$$

Toplam hatayı en aza indirmek için bu hatanın kendisine neden olan proses elemanlarına dağıtılması gerekmektedir. Bu ise proses elemanlarının ağırlıklarını

değiştirmek demektir. Ağın ağırlıklarını değiştirmek için iki durum söz konusudur. Birinci durum ara katman ile çıktı katmanı arasındaki ağırlıkların değiştirilmesi ikinci durum ise ara katmanlar arası veya ara katman girdi katmanı arasındaki ağırlıkların değiştirilmesi.(Öztemel, 2003)

***Ara katman ile çıktı katmanı arasındaki ağırlıkların değiştirilmesi:***

Ara katmanındaki j. Proses elemanın çıktı katmandaki m. proses elemanına bağlayan bağlantının ağırlığındaki değişim miktarı $\Delta A^a$ denirse herhangi bir t. iterasyondaki ağırlığın değişim miktarının hesaplanması:

$$\Delta A^a_{jm}(t) = \lambda\, \delta m\, Ç^a_j\, Y_j + \alpha\, \Delta A^a_{jm}(t-1)$$

$\lambda$: öğrenme katsayısı

$\alpha$: momentum katsayısı

$\delta m$: m. çıktı ünitesinin hatası

Öğrenme katsayısı ağırlıkların değişim miktarını, momentum katsayısı ise ağın öğrenme esnasında yerel bir optimum noktaya takılıp kalmaması için ağırlık değişim değerinin belirli bir oranda bir sonraki değişime eklenmesini sağlar.

$$\delta m = f'(NET) \,.\, E_m$$

Buradaki f ' (NET) aktivasyon fonksiyonunun türevidir. Sigmoid fonksiyonunun kullanılması durumunda;

$$\delta m = Ç_m\,(1 - Ç_m)\,.\,E_m$$

olacaktır. Değişim miktarı hesaplandıktan sonra ağırlıkların t. iterasyondaki yeni değerleri ise:

$$\Delta A^a_{jm}(t) = \Delta A^a_{jm}(t-1) + \Delta A^a_{jm}(t)$$

Benzer şekilde eşik değer ünitesinin de ağırlıklarını değiştirmek gerekmektedir. Onun için öncelikle değişim miktarını hesaplamak gerekir. Eğer çıktı katmanında bulunan proses elemanlarının eşik değer ağırlıkları $\beta^ç$ ile gösterilirse bu ünitenin çıktısının sabit ve bir olması nedeni ile değişim miktarı:

$\Delta\beta^{ç}{}_{m}(t) = \lambda\, \delta m + \alpha\, \Delta\beta^{ç}{}_{m}(t-1)$

Eşik değerinin t. İterasyondaki ağırlığının yeni değeri ise:

$\beta^{ç}{}_{m}(t) = \beta^{ç}{}_{m}(t-1) + \Delta\beta^{ç}{}_{m}(t)$

***Ara katmanlar arası veya ara katman girdi katmanı arasındaki ağırlıkların değiştirilmesi:***

Ara katman arasındaki ağırlıkların değişimde her ağırlık için sadece çıktı katmanındaki bir proses elemanının hatası dikkate alınmıştır. Bu hataların oluşmasında girdi katmanı ve ara katman arasındaki ağırlıkların payı vardır. Çünkü en son ara katmana gelen bütün bilgiler girdi katmanı veya önceki ara katmandan gelmektedir. O nedenle girdi katmanı ile ara katman arasındaki ağırlıkların değiştirilmesinde çıktı katmanındaki proses elemanların hepsinin hatasından payını alması gerekir. Bu ağırlıklardaki değişimi $\Delta A^{a}{}_{j}$ ile gösterilirse değişim miktarı:

$\Delta A^{a}{}_{kj}(t) = \lambda\, \delta^{a}{}_{j}\, Ç^{i}{}_{k} + \alpha\, \Delta A^{i}{}_{kj}(t-1)$

Buradaki hata terimi $\delta^{a}$ ise:

$\delta^{a}{}_{j} = f'(NET) \,.\, \sum \delta_{m}\, A^{a}{}_{jm}$

Aktivasyon fonksiyonu olarak sigmoid fonksiyonu düşünülürse bu hata değeri

$\delta^{a}{}_{j} = Ç^{a}{}_{j}\,(1-Ç^{a}{}_{j}).\, \sum \delta_{m}\, A^{a}{}_{jm}$

şeklinde hesaplanır.Hata değeri hesaplandıktan sonra verilen eşitlik ile değişim miktarını bulmak mümkün olur. Ağırlıkların yeni değerleri:

$A^{i}{}_{kj}(t) = A^{i}{}_{kj}(t-1) + \Delta A^{i}{}_{kj}(t)$

şeklinde olacaktır. Benzer şekilde eşik değerlerinin yeni ağırlıkları da yukarıdaki gibi hesaplanır. Ara katman eşik değerleri $\beta^{a}$ ile gösterilirse değişim miktarı:

$\Delta\beta^{a}{}_{j}(t) = \lambda\, \delta^{a}{}_{j} + \alpha\, \Delta\beta^{a}{}_{j}(t-1)$

olacaktır. Ağırlıkların yeni değerleri ise t. iterasyonda aşağıdaki gibi hesaplanır. (Öztemel,2003)

$\beta^a_j(t) = \Delta\beta^a_j(t-1) + \Delta\beta^a_j(t)$

Böylelikle ağın bütün ağırlık değerleri değiştirilmiş olacaktır. Bir iterasyon hem geri hemde ileri hesaplamaları yapılarak tamamlanmış olur. İkinci örnek seti ağa verilerek ikinci iterasyon yapılır aynı işlemler öğrenme tamamlanıncaya kadar tekrarlanır.

### 5.2.2.3 Çok Katmanlı Perceptron da Geriye Yayılım Akış Şeması

Giriş katmanı, giriş veri gruplarının ağa sunulduğu terminallerdir. Bu katmandaki nöron sayısı, A, giriş veri sayısı kadardır ve her bir giriş nöronu bir veri alır. Burada veri işlenmeden bir sonraki katmana yani gizli katmana geçer.

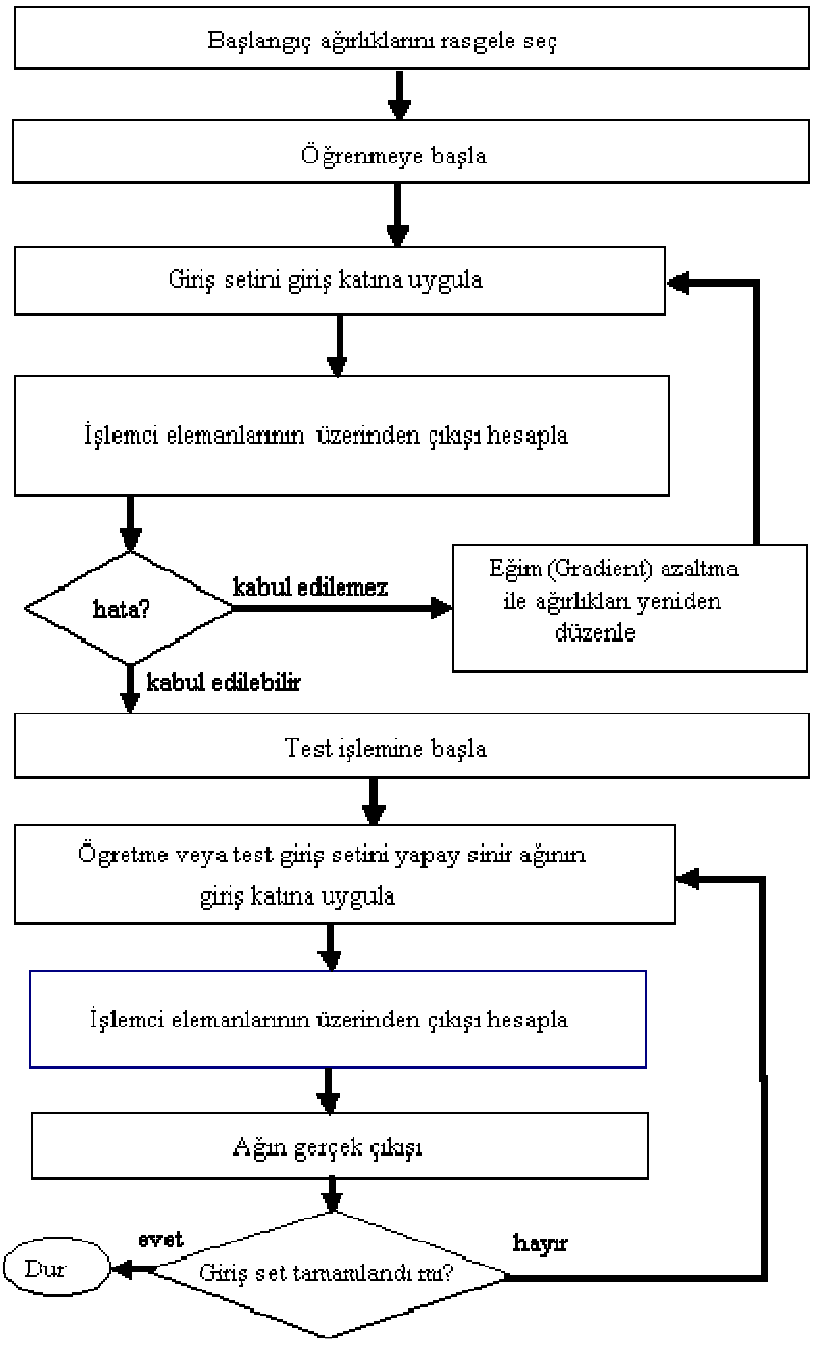
Gizli katman, ağın temel işlevini gören gizli katmandır. Bazı uygulamalarda ağda birden fazla saklı tabaka bulunabilir. Gizli katman sayısı ve katmanlardaki nöron sayısı, B, probleme göre değişir, tamamen ağ tasarımcısının kontrolü altındadır ve onun tecrübesine bağlıdır. Bu katman girdi katmanından aldığı ağırlıklandırılmış veriyi probleme uygun bir fonksiyonla işleyerek bir sonraki tabakaya iletir. Bu tabakada gereğinden az sayıda nöron kullanılması giriş verilerine göre daha az hassas çıkış elde edilmesine sebep olur. Aynı şekilde, gerektiğinden daha çok sayıda nöron kullanılması durumunda da aynı ağda yeni tip veri gruplarının işlenmesinde zorluklar ortaya çıkar.

Çıkış katmanı, ağın en uç tabakasıdır. Gizli katmandan aldığı veriyi ağın kullandığı fonksiyonla işleyerek çıktısını verir. Çıkış katmanındaki nöron sayısı, C, ağa sunulan her verinin çıkış sayısı kadardır. Bu katmandan elde edilen değerler yapay sinir ağının söz konusu problem için çıkış değerleridir.

İleri besleme safhasında, giriş katmanındaki nöronlar, veri değerlerini doğrudan gizli katmana iletirler. Gizli katmandaki bir nöron kendi giriş değerlerini ağırlandırarak toplam değer hesap ederler ve bunları bir taşıma fonksiyonu ile işleyerek bir ileri tabakaya veya doğrudan çıkış tabakasına iletirler. Tabakalar arasındaki ağırlıklar başlangıçta rasgele küçük rakamlardan seçilir.

Çıkış katmanındaki, her bir nöron ağırlıklandırılmış değeri hesaplandıktan sonra, bu değer yine taşıma fonksiyonu ile işlenerek sinir ağının ilk çıkış değeri hesaplanmış olur. Bu değer istenen çıkış değeri ile karşılaştırılarak mevcut hata hesaplanır ve hata minimize edilmeye çalışılır. Hata değeri belirli bir mertebeye

ininceye kadar iterasyon işlemine devam edilir ve böylece ağın eğitim aşaması tamamlanmış olur. Tabakalar arası bağlantılardaki ağırlık değerleri eğitimi tamamlanmış ağdan alınarak deneme safhasında kullanılmak üzere saklanır.




Şekil: 5.2.2. Çok katmanlı bir perceptron geri yayılım akış şeması

Genişletilmiş delta kuralına göre öğrenme aşağıdaki aşamalarda gerçekleşir.

1. Ağın yapısı belirlenir

   Giriş (nöron) sayısı

   Çıkış (nöron) sayısı

   Gizli katman sayısı ve gizli katmandaki nöron sayıları

2. Ağın başlangıç parametreleri belirlenir

   Başlangıç ağırlıkları

   Öğrenme katsayısı

   Momentum katsayısı

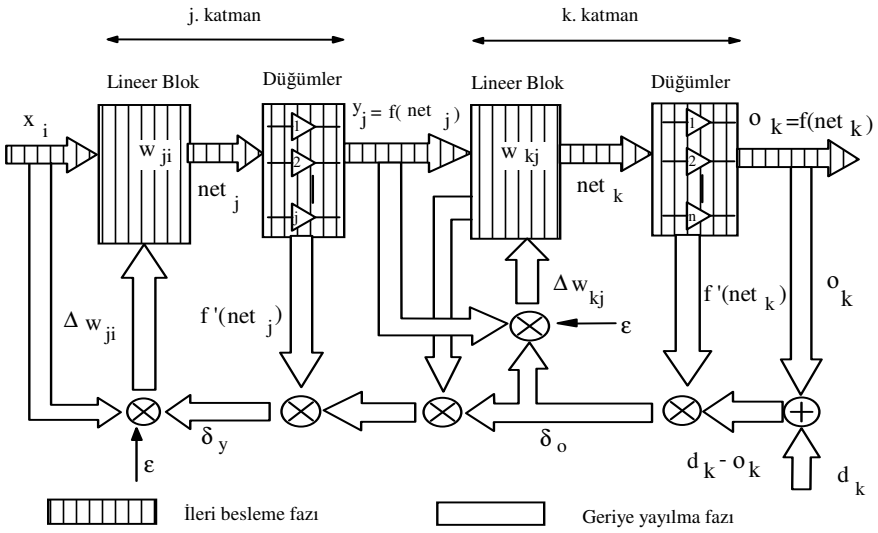
3. Giriş ve çıkış verileri ağın değerlendirebileceği şekilde düzenlenir.
4. İleri beslemeli ağ yapısına göre ağ çıktıları hesaplanır.
5. Ağ çıktısı ile gerçek çıktı arasındaki hata bulunur.
6. Hata minimum ise ağ problemi öğrendi. Öğrenmeyi durdurur.
7. Hata minimum değilse; geriye yayılım ağ yapısına göre hatayı minimize edecek şekilde ağırlıklar hesaplanır.
8. Adım üçe gidilerek işleme devam edilir.

### 5.3. Hatanın geriye yayılması algoritması ve genelleştirilmiş delta kuralı

Hatanın geriye yayılması eğitme algoritması, çok katmanlı, ileri yayılımlı bir perceptrondan elde edilen çıkışlar ile eldeki hedef çıkışlar arasındaki hataların karesinin ortalamasını minimum yapmak için geliştirilmiş iteratif bir gradyan algoritmadır ve eğitme işlemi için genelleştirilmiş delta kuralını (Generalized Delta Rule) kullanır.

Ağ mimarisi tanımlanır ve ağırlıklar bazı rasgele küçük sayılar ile başlatılarak, ağa ilk giriş sunulur. Burada m-boyutlu giriş örüntüleri girildiğinde; $x_i = [x_1, x_2, .... , x_m]^T$ dir. Benzer şekilde istenilen n-boyutlu çıkış örüntülerini ise; $d_k = [d_1, d_2, .... , d_n]^T$ belirtir. $x_i$ değerleri i katmanındaki nöronların çıkış değerleri ise, j katmanındaki bir nörona gelecek olan toplam giriş,

$$net_j = \sum_{i=1}^{m} w_{ji} . x_i \qquad (5.3.1)$$




Şekil5.3.1. Hatanın geriye yayılması algoritmasının blok diyagramı

(i. düğümden j. düğüme)olur. Gizli katmandaki j nöronunun çıkışı (transfer fonksiyonu çıkışı);

$$y_j = f_j(net_j) \quad , \quad j=1,2,...,J \tag{5.3.2}$$

dir ve burada $f_j$ transfer fonksiyonudur. Çıkış katmanındaki k nöronuna gelecek olan toplam giriş ise:

$$net_k = \sum_{j=1}^{J} w_{kj} . y_j \tag{5.3.3}$$

dir. Yine çıkış katmanındaki bir k nöronun lineer olmayan çıkışı;

$$o_k = f_k(net_k) \quad , \quad k = 1, 2 , ... , n \quad \text{olur.} \tag{5.3.4}$$

Ağdan elde edilen çıkış, asıl çıkış ile karşılaştırılır ve $e_k$ hatası aşağıdaki gibi hesaplanır:

$$e_k = (d_k - o_k) \tag{5.3.5}$$

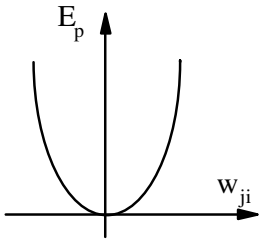
Burada $d_k$ ve $o_k$ sırasıyla çıkış katmanındaki herhangi bir k nöronunun istenen ve ağdan elde edilen asıl çıkışlardır. Dikkat edilmesi gereken husus, böyle bir karşılaştırmanın sadece ağın çıkış katmanı için mümkün olmasıdır. Böylece çıkış katmanıyla olan bağlar için ağırlık ayarlaması öncelikle gözönüne alınır. Herbir örnek set için toplam karesel hata:

$$E = \frac{1}{2}\sum_k (d_k - o_k)^2 \qquad (5.3.6)$$

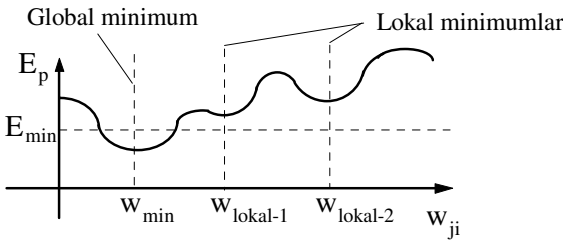
olur. Gizli katman olduğu zaman; hata düzeyi sadece bir minimumdan oluşmaz, çeşitli minimumlar oluşur. Bu minimumlar lokal ve global olarak sınıflandırılırlar. Lokal minimumlar birden fazla olabilir. Global minimum, hatanın en küçük olduğu minimumdur. Öğrenme sırasında global minimuma ulaşmak amaçlanır.(Bekir Karlık, 2007)

Ağırlıkların değişimi;

$$\Delta w_{kj} = -\varepsilon \frac{\partial E}{\partial w_{kj}} \qquad (5.3.7)$$




Şekil: 5.3.2. Gizli katman olamayan ağın hata fonksiyonu




Şekil: 5.3.3. Gizli katmana ait ağın hata fonksiyonu

Burada ε öğrenme oranı katsayısı adı verilen küçük değerde pozitif bir sayıdır . Genelikle 0.01 ile 0.9 arasında seçilir. Bu öğrenme oranı, bir sayısal optimizasyon algoritmasındaki adım boyutu parametresine benzemektedir. (5.3.7)'de eşitliğin sağ tarafı açılırsa (Rumelhart ,1986) şu sonuç elde edilir:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} \qquad (5.3.8)$$

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}}\sum_j w_{kj}.y_j = \sum_j \frac{\partial w_{kj}.y_j}{\partial w_{kj}} = y_j \qquad (5.3.9)$$

Amaç, uygun w seçimiyle, $E = \sum_p E_p$ toplam hatayı yeterince küçük yapmaktır. Bu amacı gerçekleştirmek için, bir $p \in P$ örüntüsü ardı ardına ve rastgele biçimde seçilir. k. nöronda oluşan ve "delta" adı verilen hata işareti,

$$\delta_o = -\frac{\partial E}{\partial net_k} \qquad (5.3.10)$$

(5.3.9) ve (5.3.10)'nu (5.3.8)'de yerine konulursa,

$$\frac{\partial E}{\partial w_{kj}} = -\delta_o.y_j \qquad (5.3.11)$$

ve, (5.3.11) ifadesi (5.3.7)'de yerine konulursa,

$$\Delta w_{kj} = \varepsilon \delta_o y_j \qquad (5.3.12)$$

$$\Delta\beta^a_j(t) = \lambda\,\delta^a_j + \alpha\,\Delta\beta^a_j(t-1)$$

elde edilir. $E_p$ değerini düşürmek demek, ağırlığı $\delta_o y_j$ 'ye bağlı olarak değiştirmek demektir. Buna "Delta kuralı" denir. Önceki gizli katmandaki tüm nöronlar ile çıkış

katmanındaki k. nöron arasındaki bağ mukavemetleri, (5.3.12)'deki $\Delta w_{kj}$ miktarı ile ayarlanır. (5.3.10) denklemi kısmi türevlerine ayrılırsa,

$$\delta_o = -\frac{\partial E}{\partial net_k} = -\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k} \tag{5.3.13}$$

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k) \tag{5.3.14}$$

olup; k. nöron çıkışının lokal hatasını verir.

$$\frac{\partial o_k}{\partial net_k} = f_k^{'}(net_k) \tag{5.3.15}$$

Son iki formül (5.3.13)'de yerine konulursa,

$$\delta_o = (d_k - o_k) f_k^{'}(net_k) \tag{5.3.16}$$

bulunur. Bu son terim (5.3.12)'de yerine konulduğunda k nöronu için;

$$\Delta w_{kj} = \varepsilon(d_k - o_k) f_k^{'}(net_k) y_j \tag{5.3.17}$$

olur. Eğer ağırlıklar çıkış nöronlarını doğrudan etkilemiyorsa (arada gizli katman varsa), (5.3.12)'e benzer biçimde delta kuralı bu gizli katman için de şu şekilde uygulanır.

$$\begin{aligned} \Delta w_{ji} &= -\varepsilon\frac{\partial E}{\partial w_{ji}} \\ &= -\varepsilon\frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -\varepsilon\frac{\partial E}{\partial net_j}x_i \\ &= -\varepsilon\left(-\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j}\right)x_i = \varepsilon\left(\frac{\partial E}{\partial y_j}\right)f_j^{'}(net_j)\,x_i \end{aligned} \tag{5.3.18}$$

$$\Delta w_{ji} = \varepsilon\delta_y x_i \tag{5.3.19}$$

Bununla birlikte, $\partial E / \partial y_j$ faktörü doğrudan geliştirilemez. Özel olarak çıkış katmanına etkisi,

$$-\frac{\partial E}{\partial y_j} = -\sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial y_j} = \sum_k \left(-\frac{\partial E}{\partial net_k}\right)\frac{\partial}{\partial y_j}\sum_m w_{km} y_j = \sum_k \left(-\frac{\partial E}{\partial net_k}\right) w_{kj} = \sum_k \delta_o w_{kj}$$

şeklinde bulunur. Bu durumda;

$$\delta_y = f_j^{'}(net_j)\sum_k \delta_o w_{kj} \qquad (5.3.20)$$

olur. Kısaca özetlenecek olursa; (5.3.18) formülü şayet j. nöron, çıkış katmanı nöronuysa (25.3.15)'e benzer biçimde,

$$\delta_y = (d_y - o_y) f_j^{'}(net_j) \qquad (5.3.21)$$

olur. Eğer j nöronları gizli katmana ait nöronlar ise, o zaman (5.3.17) denklemi kullanılır. Transfer (eşik) fonksiyonu olarak sigmoid fonksiyonu kullanıldığında,

$$f(net_j) = y_j = \frac{1}{1+e^{-net_j}} \qquad (5.3.22)$$

olur. Bu ifadenin türevi alınıp, gerekli sadeleştirme yapıldığında,

$$f^{'}(net_j) = \frac{1}{1+e^{-net_j}}\frac{1+e^{-net_j}-1}{1+e^{-net_j}} \qquad (5.3.23)$$

$$\frac{\partial y_j}{\partial net_j} = y_j(1-y_j) \qquad (5.3.24)$$

bulunur. Benzer işlem k katmanı için de yapılırsa;

$$\frac{\partial o_k}{\partial net_k} = f_k'(net_k) = o_k(1-o_k) \tag{5.3.25}$$

elde edilir. (5.3.25) denklemi (5.3.16)'da, (5.3.24) denklemi de (5.3.20)'de yerine konursa, (5.3.16) ve (5.3.20)'deki delta ifadeleri aşağıdaki gibi olur. ($\delta_o$ çıkış katmanı, δy: gizli katman elemanları içindir);

$$\delta_o = (d_k - o_k)o_k(1-o_k) \tag{5.3.26}$$

$$\delta_y = y_j(1-y_j)\sum_k \delta_o w_{kj} \tag{5.3.27}$$

$w_{ji}$ yi öğrenme durumunda her eğitme örüntüsünün seti için $\Delta w_{ij}$ yi hesaplamak gerekir. Öğrenme oranı ε hızlı öğrenmeyi sağlar; fakat dalgalanmalara sebep olabilir. Bu durumda;

$$\Delta w_{ji}(n+1) = \varepsilon \delta_y x_i + \alpha \Delta w_{ji}(n) \tag{5.3.28}$$

$$\Delta w_{kj}(n+1) = \varepsilon \delta_o y_j + \alpha \Delta w_{kj}(n) \tag{5.3.29}$$

olarak yazılabilir. Burada n, öğrenme döngülerinin (iterasyon) sayısını gösterir. Momentum terimi olan α, küçük değerde pozitif bir sayıdır.(Karlık, ders notları)

### 5.3.4. Öğrenme ve momentum katsayıları

YSA ile ilgili bir başka sorun da, düzgün bir öğrenme katsayısının (ε) ayarlanmasıdır. Ağırlıkları çok yüksek tutmak davranışın bozulmasına sebep olabilir. Öğrenme katsayısını böyle bir davranışı önlemek için küçük tutmak gereklidir. Öğrenme katsayısı, $0.01 < \varepsilon < 10$ aralığında seçilen sabit bir sayıdır. Öte yandan, çok küçük bir öğrenme oranında, öğrenme işleminin yavaşlamasına yol açar.

Momentum (α) fikri bu noktadan hareketle ortaya atılmıştır. Momentum mevcut delta ağırlığı üzerinden önceki delta ağırlığının belirli bir kısmını besler. Böylece daha düşük öğrenme katsayısı ile daha hızlı öğrenme elde edilir. Momentum katsayısı genellikle $0 < \alpha < 1$ aralığında değişen bir sayıdır.

Hatanın geriye yayılması algoritması genellikle iyi bir performans göstermesine rağmen; bu algoritma, bir gradyent arama tekniği olduğu için, global minimum yerine, en küçük kareler (Least Mean Square) fonksiyonunun bir lokal minimumunu bulabilmektedir. Performansı artırmak ve lokal minimum problemini azaltmak için; gizli düğümler ilave edilmesi, ağırlıkların ayarlanması için kullanılan öğrenme katsayısının azaltılması ve farklı rastgele ağırlık setlerinden başlayarak birçok defa eğitme işleminin tekrarlanması öneriler arasındadır (Karlik 1994).

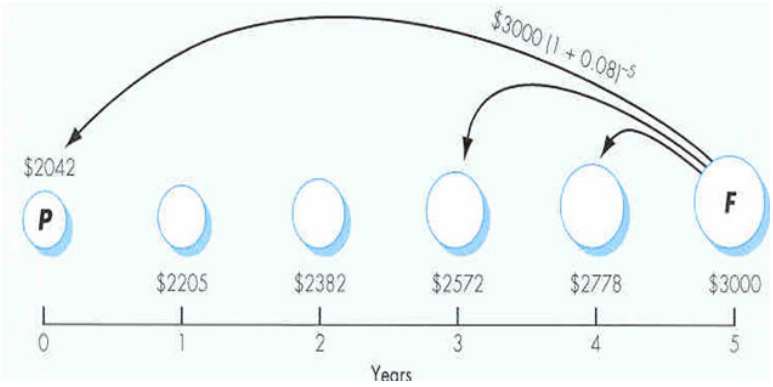
# 6.YAPAY SİNİR AĞI UYGULAMASI ve YAPILAN ÖN HAZIRLIKLAR

## 6.1. Yazılım Projelerinin Maliyetlerinin Yönetimi ve Zaman Değerlerinin Hesaplanması

Yazılım firmalarından yapay sinir ağı uygulamasının örnek setleri alınmıştır. Fakat yazılım firmaları gerçekleştirmiş olduğu yazılımları farklı tarihler içerisinde oluşturmuştur. Dolayısıyla elde edilen yazılım maliyetleri her bir proje için farklı dönemlere aittir. Paranın zaman değeri göz önüne alınması gerekmektedir. Yani elde edilen her bir proje için dönemin faiz oranları kullanılarak belirlenen günümüz tarihine göre projenin yeni maliyeti hesaplanmıştır. Bu hesaplamada faiz formülleri, ekonomik eşdeğerlik hesapları, firmanın nakit akış değerleri ve geri ödeme planları ele alınmıştır. Faiz oranları ise yıllara göre aylık olarak ele alınmıştır.

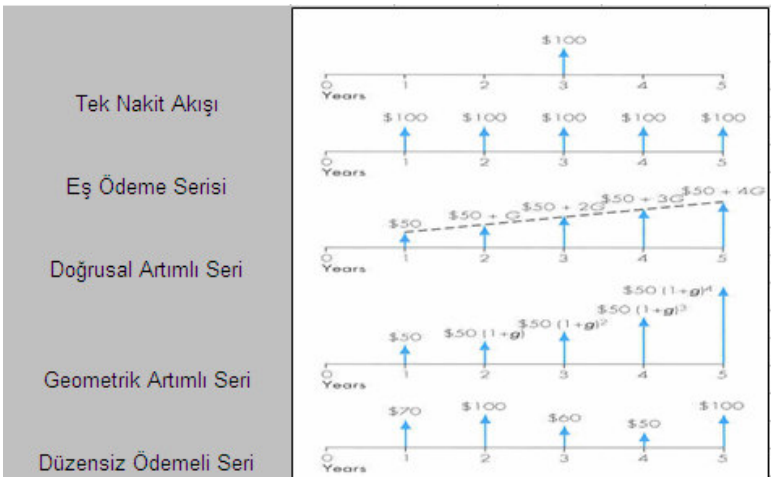
- Ekonomik eşdeğerlik, iki nakit akışının aynı ekonomik etkiye sahip olması ve bu yüzden birbiriyle değiştirilebilmesidir.

- Nakit akışındaki miktarlar ve zamanlar farklı olmasına rağmen, uygun bir faiz oranı iki nakit akışını birbirine eşit yapar.(Bakınız Şekil: 6.1) Farklı dönemlerdeki farklı miktarlardaki nakit akışları % 0.8 faiz oranı ile 5 yıl sonrasında $3000' lık meblağa eşittir.




Şekil 6.1.1. Nakit akış diyagramı

Şekil 6.1.2 de nakit akış türleri verilmiştir. Bu nakit akışlarının türlerine göre kullanılabilecek hesaplama yöntemleri ise:



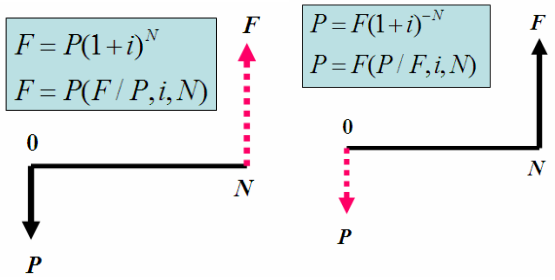

Şekil: 6.1.2. Nakit akış diyagramı türleri

***Tek nakit girişli/çıkışlı formül:*** Eğer *P* kadar para şimdi *N* dönem için *i* faizinden yatırılırsa, *N* dönem sonra *F* kadar para elimize geçecektir. *N* dönem sonraki *F* kadar para şimdiki *P* kadar paraya eşdeğer olmaktadır. Para kazanma gücümüz *i* faiz oranı ile ölçülmektedir.

P: Paranın Şimdiki değeri

F: Paranın Gelecek zamandaki değeri

i: faiz oranı




Şekil: 6.1.3. Tek nakit (girişli/çıkışlı) akış diyagramı

$$F = P \times [(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)]$$

$$F = P \times (1+i)^t$$

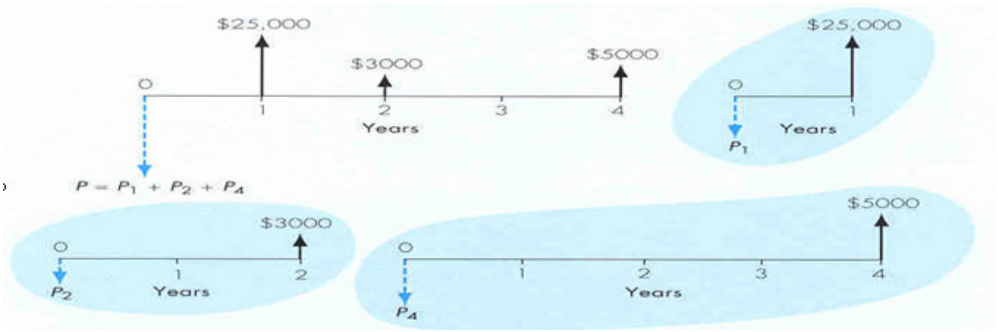
$$P = F \div [(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)\cdot(1+i)]$$

$$P = \frac{F}{(1+i)^t}$$

$$F = PV \times (1+i)^t \qquad (6.1.1.)$$

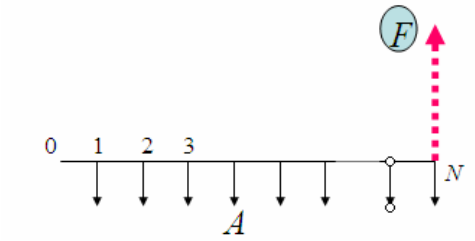
***Düzensiz Ödeme Serisi***: Farklı miktarlardaki birden fazla nakit girişi ve çıkışı bulunan nakit akış diyagramlarıdır. Her bir nakit akışı ayrı ayrı ele alınarak paranın şimdiki ve gelecek zaman değeri belirlenen faiz oranları ile hesaplanır. Şekil 6.4. de verilen düzensiz ödeme serisi üç ayrı tek nakit akışı haline getirilerek günümüz değeri hesaplanmıştır.

$$
\begin{aligned}
P_1 &= \$25{,}000(P/F,10\%,1)\\
P_2 &= \$3{,}000(P/F,10\%,2)\\
P_4 &= \$5{,}000(P/F,10\%,4)\\
P &= P_1 + P_2 + P_4\\
&= \$28{,}622
\end{aligned}
$$




Şekil: 6.1.4 Düzensiz Ödeme Serisi

***Eşit Ödemeli Seri:*** N dönem boyunca % i kazandıran (A) miktarındaki dönemsel ödemelerin gelecekteki değerinin F ve şimdiki zaman değerinin P hesaplanması aşağıdaki formüller aracılığıyla gerçekleştirilir.




Şekil: 6.1.4 Eşit ödemeli seri

$$F = A\frac{(1+i)^N - 1}{i} = A(F/A,i,N) \qquad (6.1.2.)$$

$$P = A\frac{(1+i)^N - 1}{i(1+i)^N} = A(P/A,i,N) \qquad (6.1.3.)$$

Eşit ödemeli serilerde günümüzdeki ve gelecek değerler verildiğinde eşit ödeme miktarlarının hesabı:

$$A = F\frac{i}{(1+i)^N - 1} = F(A/F, i, N) \qquad (6.1.4.)$$

$$A = P\frac{i(1+i)^N}{(1+i)^N - 1} = P(A/P, i, N)$$

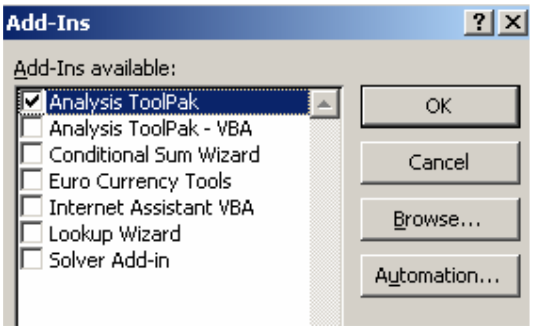
(6.1.5.)

Paranın zaman değeri Excell'de rahatlıkla hesaplanabilmektedir. Excell'de hesabı yapılması istenen spreadsheet'e ekle menüsünden "Function" ve "Financial" fonksiyonları eklenmeli

| | |
|---|---|
| PV | PMT |
| FV | RATE |
| FVSCHEDULE | IPMT |
| PPMT | NPER |
| NPV | XNPV |
| IRR | XIRR |

Şekil: 6.1.5. Eklenmesi gereken fonksiyonlar

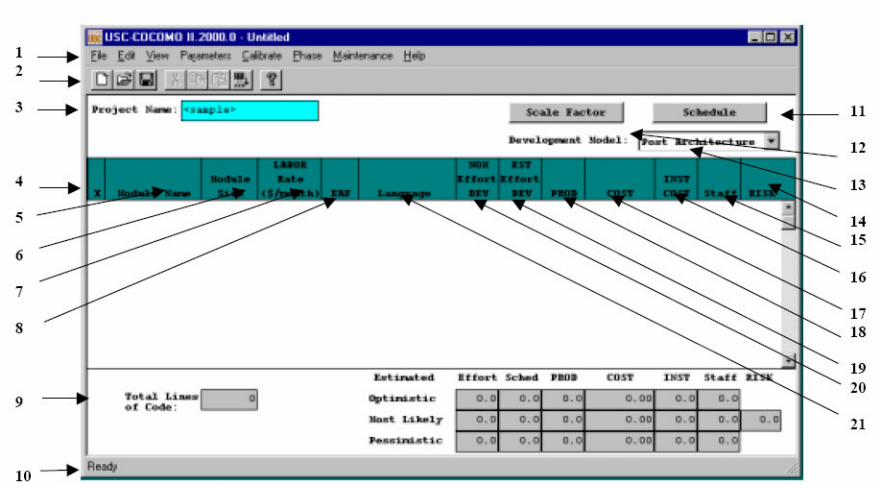
Fonksiyonlar eklendikten sonra "Tools menu" den Check Analysis Toolpak yüklenmelidir.




Şekil: 6.1.6 Analysis ToolPak

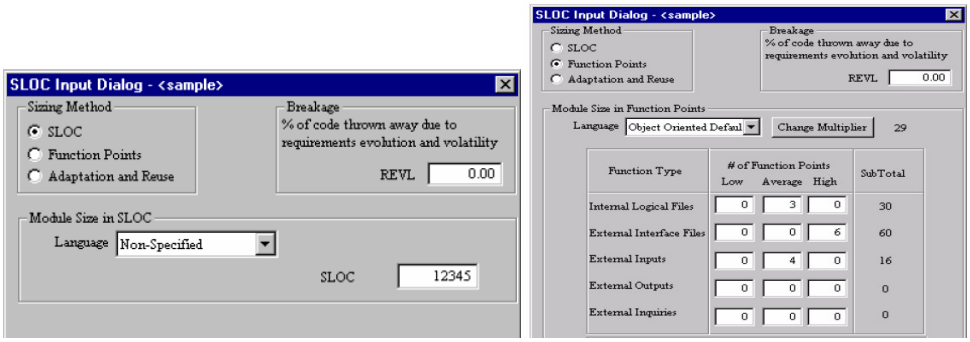
## 6.2. COCOMO II 2000 Kullanımı ile Yazılım Maliyet Tahmini Uygulaması

COCOMO II B. Boehm tarafından yayımlandıktan sonra oldukça ilgi gören bir maliyet kestirim modeli olmuştur. Uygulamada kullanılacak ayrıntı düzeyine göre üç ayrı model biçiminde yapılabilir. Bütün COCOMO modelleri temel girdi olarak satır sayısı kestirimini alır ve çıktı olarak iş gücü ve zaman çıktılarını verir. İş gücü değerinin zaman değerine bölünmesiyle yaklaşık kişi sayısı kestirimi de elde edilmiş olur. Tüm COCOMO modelleri gerek iş gücü gerekse zaman için doğrusal olmayan üstel formüller kullanır.



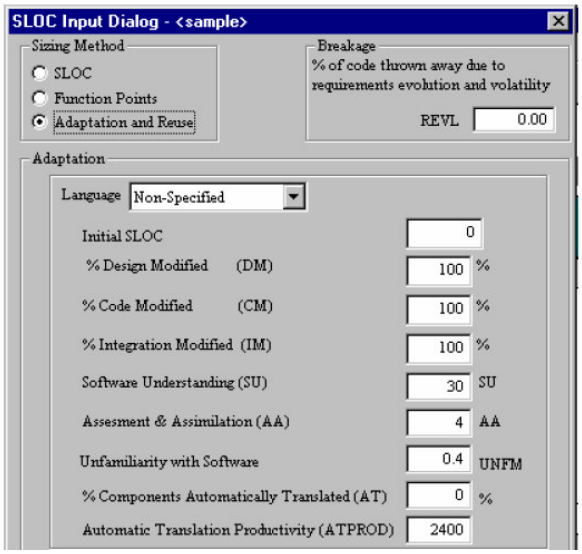

Şekil: 6.2.1. COCOMO II arayüzü

Şekil 6.2.1. 'de numara 1. ile gösterilen COCOMO II' nin Menu Bar'ıdır. File, Edit, View, Calibrate… gibi fonksiyonların bulunduğu araç çubuğudur. 2. numara ile gösterilen bölge Tool Bar'dır. Windows'daki gibi görüntü butonları vardır. Modül ekleme, silme, kopyalama gibi işlemlerde kullanılır. 3 numaralı bölge ise proje isminin görüntülendiği bölgedir. Proje ismini değiştirmek için iki kere üzerine tıklayarak yeni proje ismi yazılır. 4 numaralı bölgeye X satırının bulunduğu yere her seferinde yalnızca bir modül koyulur. Her bir modülün ismi 5 numaralı bölgeye yazılır. 6 numaralı bölgede ise modülün kaç satır koddan oluştuğu belirtilir. Bir yazılım projesinin kaç satır koddan oluştuğunu üç farklı yoldan girilebilir.



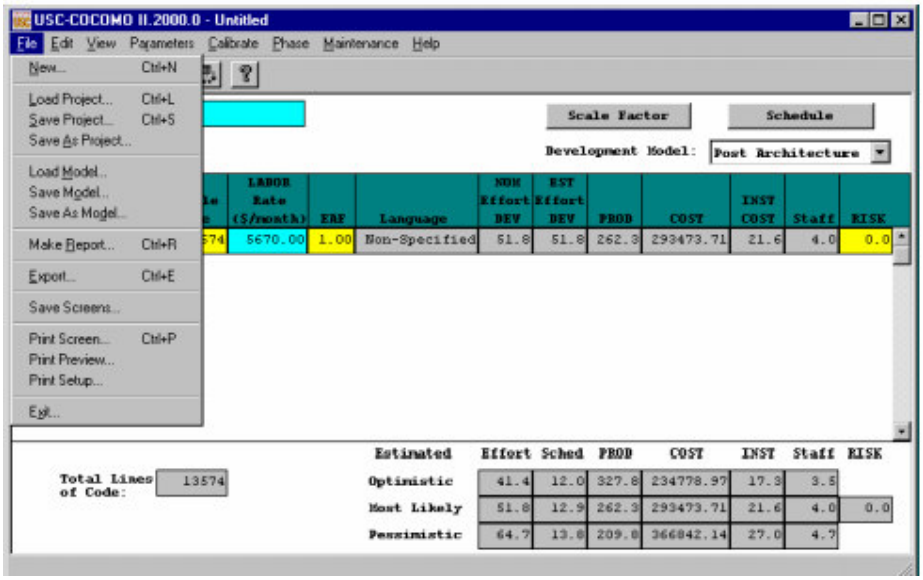

Şekil: 6.2.2. Satır Sayısı Giriş Yöntemleri 1 ve 2

Şekil 6.2.2. ‘de gösterildiği gibi istenirse direk olarak kaç satır koddan oluştuğu direk olarak SLOC butonu işaretlenerek girilebilir. Ya da Function Points seçeneği seçilerek veriler girilir. Diğer alternatif ise Adaptation And Reuse seçeneğinden Şekil 6.2.3. ‘deki gibi girilir.




Şekil: 6.2.3. Satır sayısı giriş yöntemleri 3

Numara 7 ile gösterilen bölmeye her bir çalışan için modül bazında ödenmesi gereken aylık miktarlar girilir. Numara 8 ile gösterilen bölge her bir modül için “cost driver” ların belirlendiği alandır. 17 numaralı bölge her bir modülün gerçekleşmesi için gereken maliyeti gösterir.

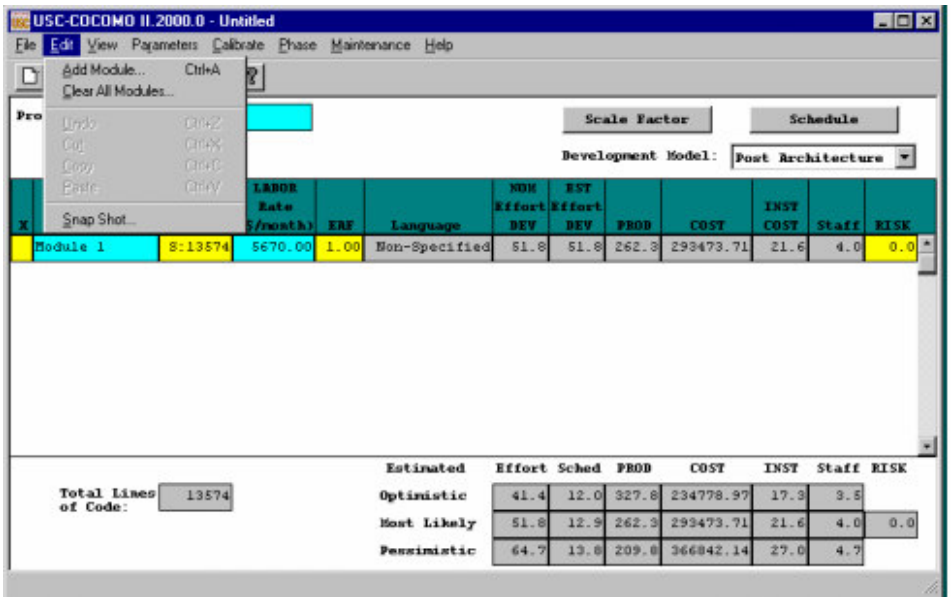



Şekil: 6.2.4. COCOMO II ile yeni proje oluşturma

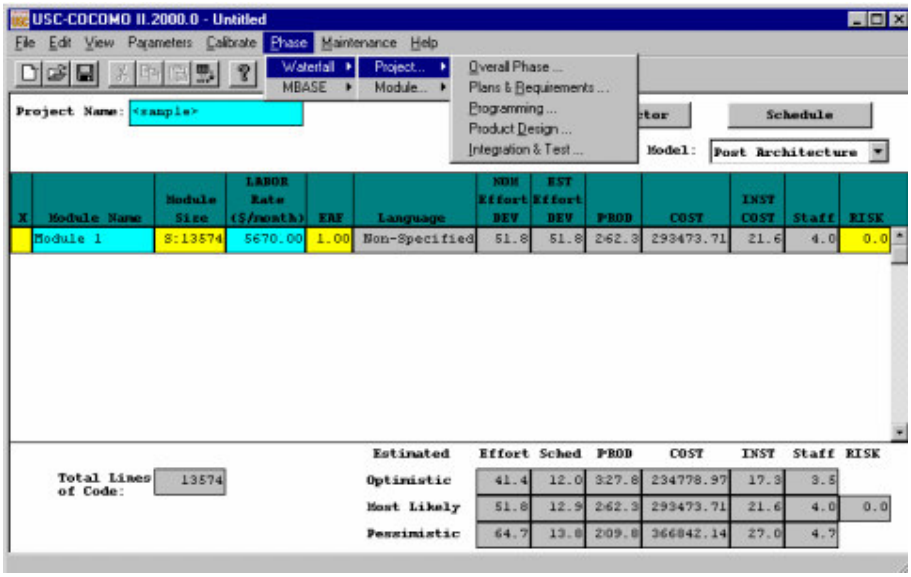
COCOMO II kullanılarak yeni bir proje oluşturmak için “File” menüsünden” Load Project” seçeneği işaretlenir. (Bakınız şekil: 6.2.4.) Projenin ismi verildikten sonra “est “ uzantısı ile kaydedilir.(Şekil:6.2.5.) Yükleme işlemi tamamlandıktan sonra projenin ismi “Project File”bölümünde ve ilgili modüllerde “CLEF” kısmında bulunur.


Şekil: 6.2.5. Projenin kaydedilmesi

COCOCMO II ‘ye yüklenen projeye yeni bir modül eklenmek istenirse “Edit” menüsünden “Add Module” seçeneğinden yeni modül projeye dahil edilir.

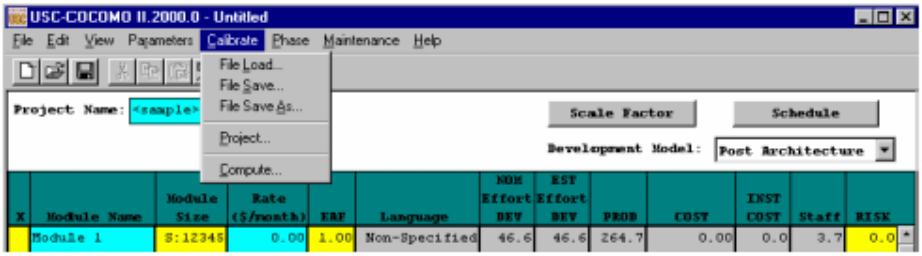



Şekil: 6.2.6. Projeye yeni modül eklenmesi



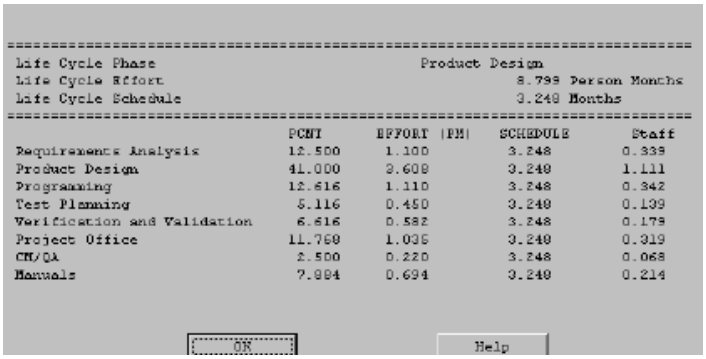

Şekil: 6.2.7 Projenin Phase'nin belirlenmesi

Projenin hangi Phase de yapıldığını belirlemek için "Phase " menüsünden Waterfall ->Project -> overall phase, Plans and Requirements, programming, product Design seçeneklerinden biri seçilir. (Şekil: 6.2.7)

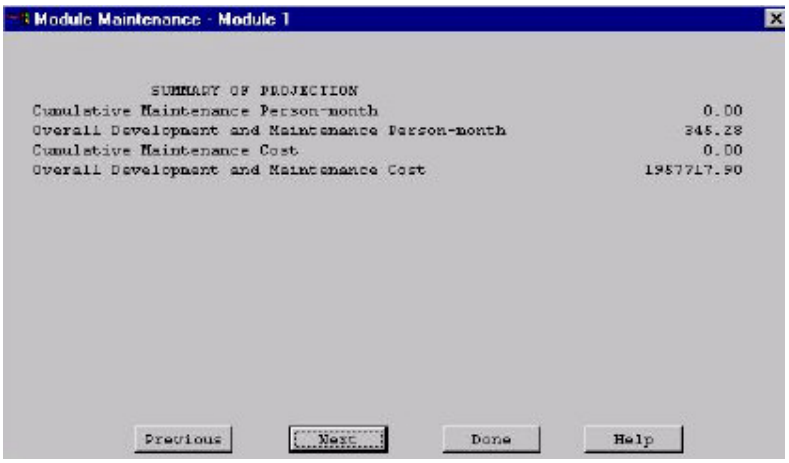



Şekil: 6.2.8 Projenin kalibrasyonu

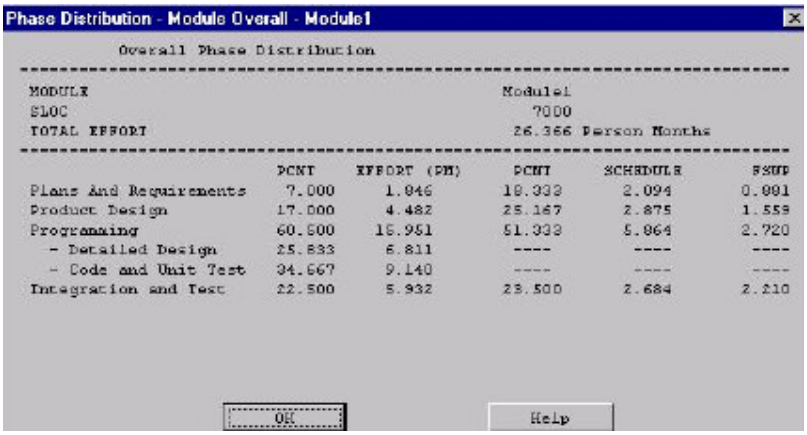
COCOMO II proje kestirimlerini verilen verilere göre hesaplayabilecek durumdadır. "Calibrate " seçeneğinden COCOMO II'nin hesaplamasını daha güvenilir hale getirilebilir.




Şekil: 6.2.9. COCOMO II çıktısı




Şekil: 6.2.10. COCOMO II ye göre proje maliyeti



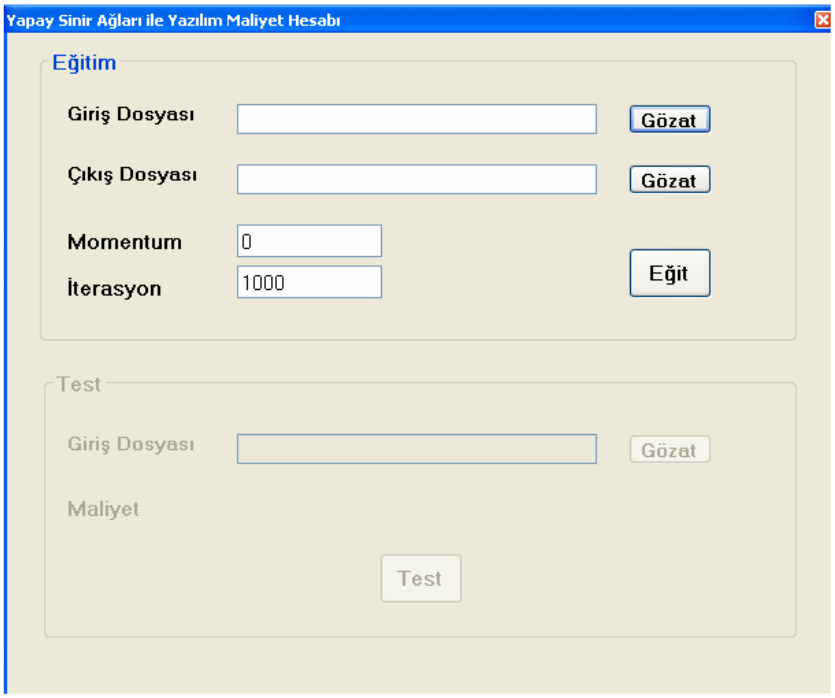

Şekil: 6.2.11. COCOMO II'ye göre gerekli iş gücü ve satır sayısı çıktısı

### 6.3. Yapay Sinir Ağı Uygulaması ile Yazılım Maliyet Tahmini Uygulaması

Yapay sinir ağı uygulaması için öncelikle yazılım firmalarından yazılım projelerine ait veri toplama işlemi gerçekleştirilmiştir. Yazılım maliyetini en çok etkileyen faktörler olarak belirlenen çalışan kişi sayısı, yazılım büyüklüğü, yazılım kod satır sayısı, bozunma oranı ve yazılım maliyeti her bir geliştirilen proje için firmalardan ilgili veriler sağlanıldı. Elde edilen yazılım maliyetleri, projenin gerçekleştirilme tarihine göre günümüz tarihindeki proje maliyet değerleri nakit akış diagramları ve Microsoft Excel yardımı ile oluşturuldu. Elde edilen verilerin normalizasyonu sağlandıktan sonra veriler eğitim ve test için iki ayrı gruba ayrıldı. Eğitim için 40 tane yazılım projesi kullanıldı. Eğitim için giriş dosyası olarak "TrainInput.txt" adlı dosyada yazılan kod satır sayısı, projedeki mevcut bozunma oranı ve çalışan kişi sayısı tutuldu. Eğitim çıkış dosyası olan "TrainOutput" dosyasında ise her bir projeye denk gelen yazılım maliyeti eklendi. Test verisi ise "TestInput" dosyasına aktarıldı.

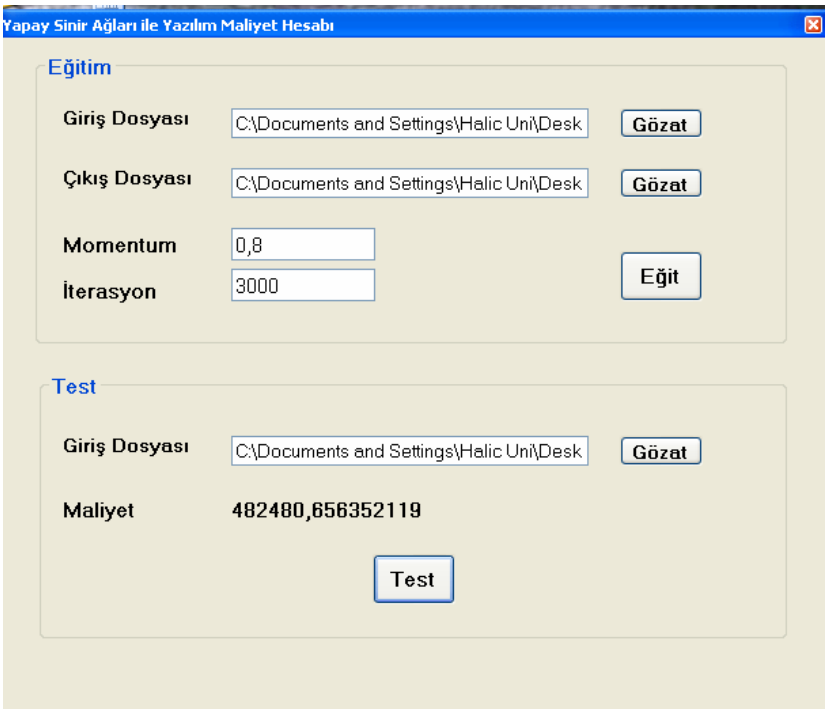
Yapay sinir ağı olarak çok katmanlı ileri beslemeli yapay sinir ağı seçildi. Eğitim algoritması olarak Delta Algoritmasına karar verildi. YSA uygulaması için hazırlanan program Microsoft Visual Stdio 2005 ortamında hazırlanmıştır. İleri beslemeli YSA'da eğitim, oluşturulan örnek setinin birinin girdi katmanından ağa sunulması ile başlar. Giriş katmanında herhangi bir bilgi işlemi gerçekleştirilmez. Giriş katmanına gelen veriler ara katmana hiçbir değişiklik yapılmadan gönderilir. Ara katmandaki her proses elemanı girdi katmanındaki bütün proses elemanlarından gelen bilgileri bağlantı ağırlıklarının etkisi ile alır. Önce ara katmandaki proses elemanlarına gelen net girdi hesaplanır. Girdi katmanı elemanını, ara katman elemanına bağlayan bağlantının çıktısı ise bu net girdinin sigmoid fonksiyonundan geçirilmesi ile hesaplanmıştır. Örnek veri setleri kullanan program iterasyon sayısı kadar döngüde kalarak ağırlıkları hesaplamıştır. Burada önemli olan ağın başlangıç parametrelerini doğru olarak belirlemektir. Yani başlangıç ağırlıklarını, öğrenme katsayısını ve momentum sayısını doğru olarak belirlenmelidir. Bu uygulamada denemeler sonucu en uygun momentum sayısı 0.8, iterasyon sayısı olarak 3000 ve öğrenme katsayısı olarak 0.1 değerleri belirlenmiştir. Bu değerler eğitim veri setleri ile ağa sunularak ağ çıktıları hesaplanır. Gerçek çıktı ile ağ çıktısı arasındaki fark minimum değere ulaşıncaya kadar işlem devam eder. Hata minimum değerdeyse ağ öğrenmeyi gerçekleştirmiştir. Burada dikkat edilmesi gereken en önemli husus iterasyon sayısını doğru belirlemektir. İterasyon sayısı yüksek tutulduğunda ağda bozulmalara neden olur ve ağ çıktıları gerçek değerden sapma gösterebilir.

Microsoft Visual Stdio' da hazırlanan program çalıştırıldığında şekil: 6.3.1. ekrana gelir. Gözat tuşundan giriş ve çıkış dosyaları bulundukları bölümden eklenir. Ekleme işlemi istenildiği takdirde elle direk olarak yazılabilir.



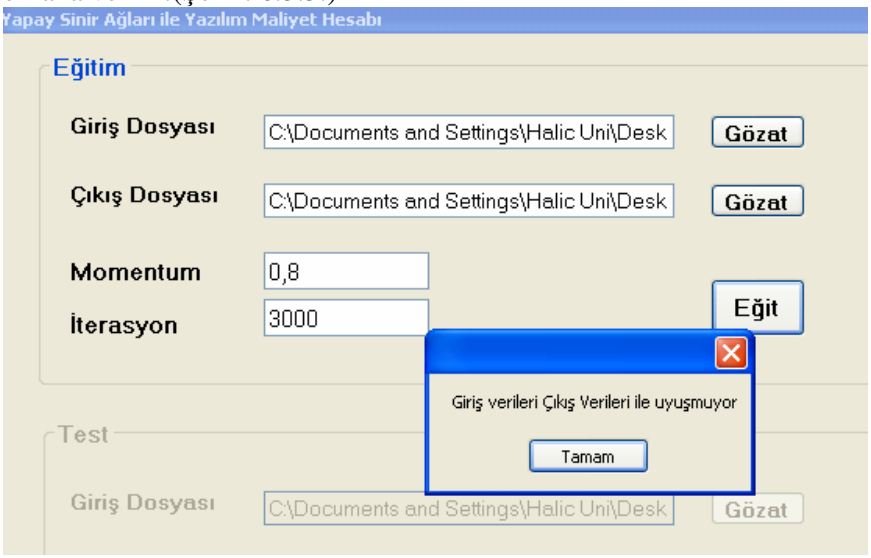

Şekil: 6.3.1 Yazılım maliyet tahmini uygulaması arayüzü 1

Şekil 6.3.2' de olduğu gibi giriş ve çıkış dosyalarından sonra momentum sayısı 0.8 ve iterasyon sayısı olarak 3000 verildiğinde ağ artık eğitime hazırdır. "Eğit" düğmesine basılarak ağ eğitilir. Eğitimini tamalayan program "Ağ Eğitildi " mesajını verir. İkinci bölümde yer alan test kısmı için maliyet tahmini yapılması istenilen yazılımın giriş dosyası ağa sunulur. "Test" düğmesine basılınca ağ test işlemini gerçekleştirir.



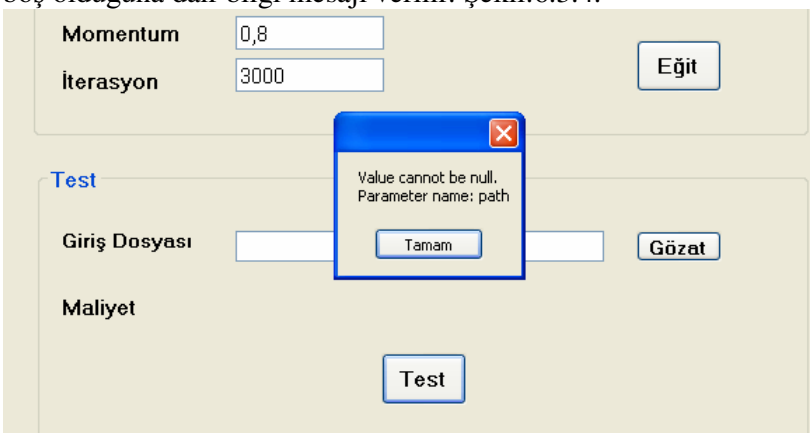

Şekil: 6.3.2. Yazılım maliyet tahmini uygulaması arayüzü 2

Yazılımda kontrollerde mevcuttur. Örneğin eğitim için oluşturulan giriş veri seti sayısı çıkış veri sayısı ile uyuşmadığında "Giriş verileri Çıkış verileri ile uyuşmuyor" mesajı ekrana verilir.(Şekil: 6.3.3.)



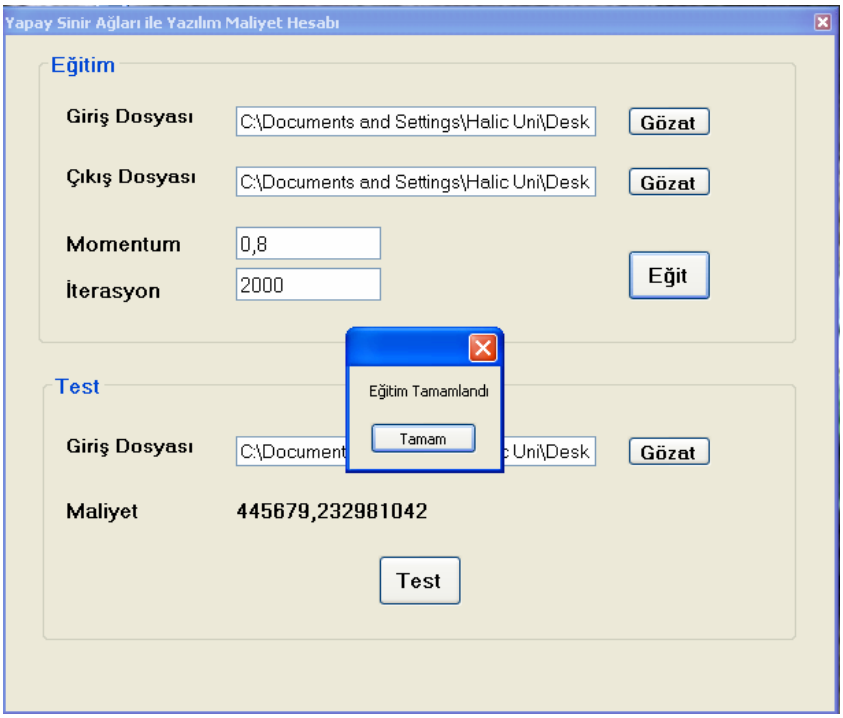

Şekil: 6.3.3 Yazılım maliyet tahmini uygulaması arayüzü 3

Ayrıca giriş dosyaları boş olduğunda yada ulaşılamadığında ekrana dosyanın içeriğinin boş olduğuna dair bilgi mesajı verilir. Şekil:6.3.4.




Şekil:6.3.4. Yazılım maliyet tahmini uygulaması arayüzü 3

Yapılan uygulamada test için kullanılan yazılım toplam 24000 satır koddan oluşmuştur. Bu yazılımı 24 tane yazılımcı oluşturmuştur ve yazılımdaki bozunma oranı 12 %'dir. Yazılımın firmaya maliyeti 480000 YTL'dir. Yaptığımız uygulama ile yazılımın maliyeti 2000 iterasyon için 445679 YTL olarak hesaplanmıştır(Şekil 6.3.5.). 3000 iterasyon ile yazılım maliyeti 482480 YTL olarak bulunmuştur (Şekil: 6.3.2.). Yazılımın maliyet tahmini firmadan alınan bilgiye göre COCOMO II 2000 ile 465 000YTL olarak tahmin edilmiştir .




Şekil:6.3.5. Yazılım maliyet tahmini uygulaması arayüzü 5

# 7. SONUÇ ve ÖNERİLER

Günümüzde teknolojinin hızla gelişmesine bağlı olarak yazılımlar her geçen gün daha fazla önem kazanmaktadır. Yazılımların birçoğu yanlış maliyet ve zaman tahmininden dolayı başarısızlıkla sonuçlanmaktadır. Bu çalışmada yazılım projelerinin maliyetini tahmin etmek amacıyla yapay sinir ağı uygulaması geliştirilmiştir.

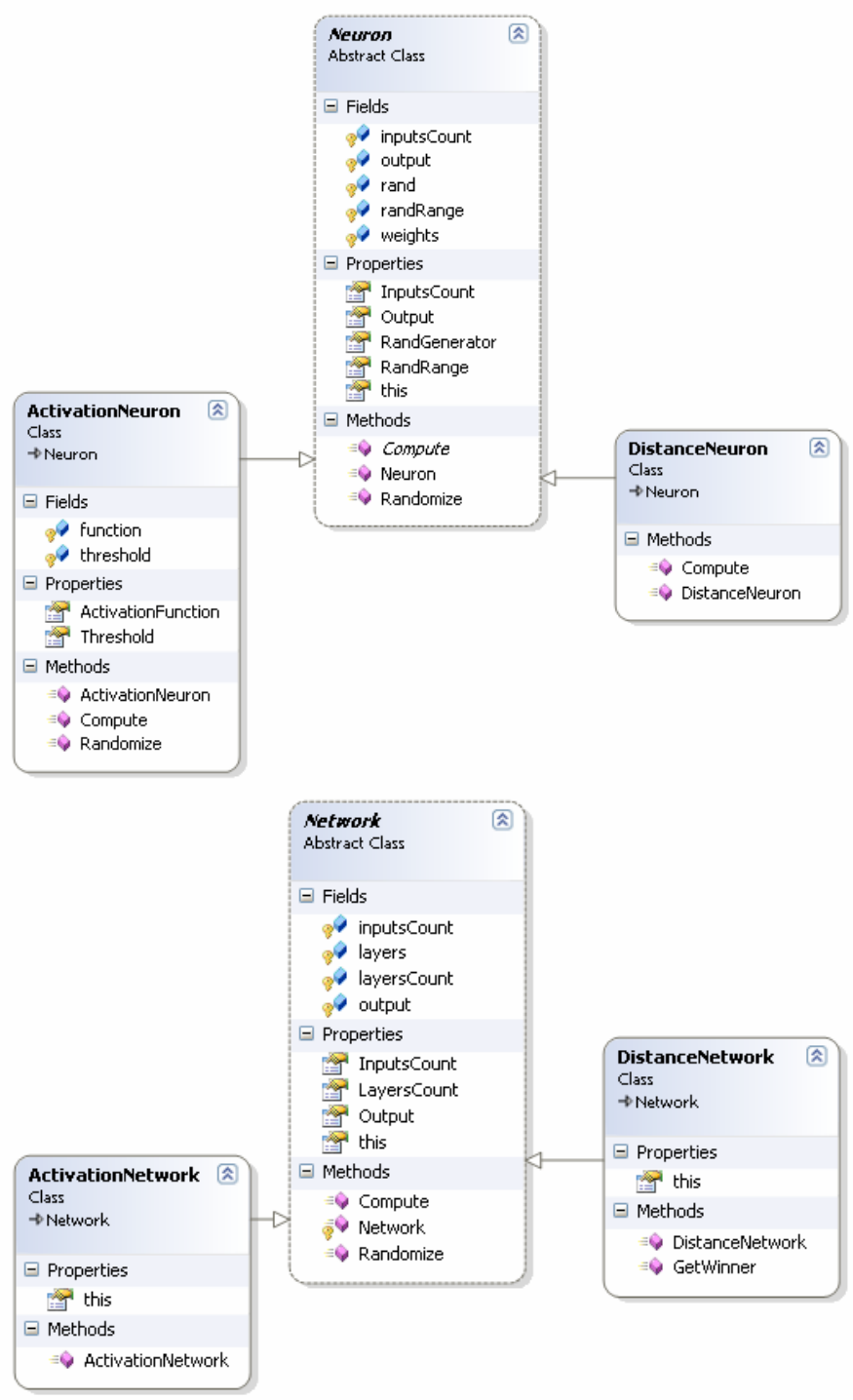
Yapay sinir ağı olarak çok katmanlı ileri beslemeli yapay sinir ağı seçildi. Eğitim algoritması olarak Delta Algoritması kullanıldı. Farklı yazılım firmalarından gerçekleşmiş yazılım projelerine ait veriler toplanıldı. Veriler önişlemden geçirildikten sonra eğitim ve test verileri olarak ayrıldı. YSA uygulamasının elde edilen veriler ile çalıştırılması sonucu gerçek yazılım maliyetine oldukça yakın bir değer elde edildi. Bu değer mevcut yazılım kestirim metodlarından biri olan COCOMO II 2000 ile karşılaştırıldı. Bu karşılaştırmanın doğru olabilmesi için yapay sinir ağında test için kullanılan yazılım projesi COCOMO II 2000'de de kullanıldı. Elde edilen sonuçlar karşılaştırıldığında yapay sinir ağı uygulaması ile daha iyi bir sonuç elde edildiği gözlendi.

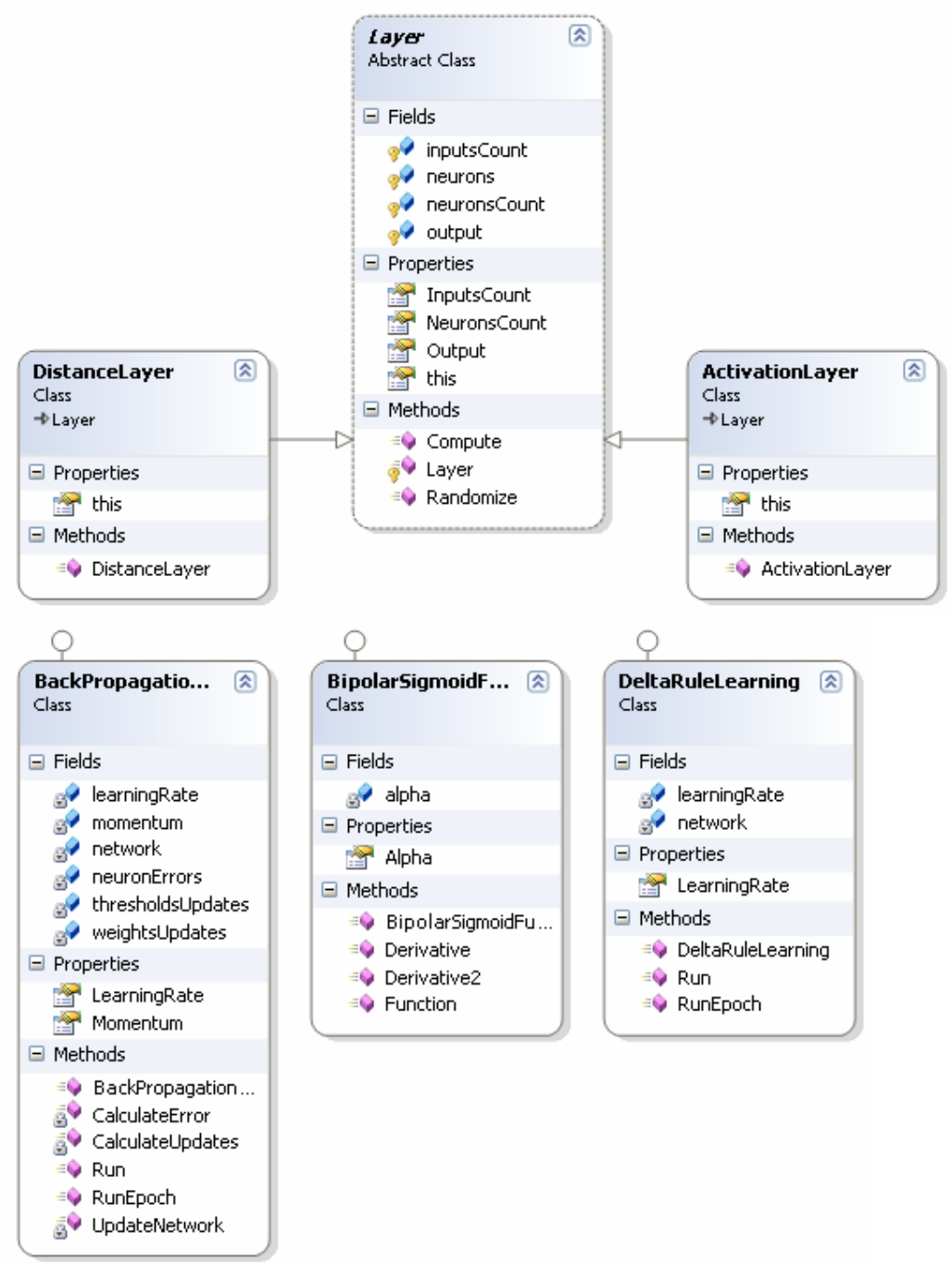
Yazılım projelerinin maliyet tahmininde yapay sinir ağı uygulaması ile daha başarılı sonuçlar elde etmek için yazılım maliyetini etkileyen faktörlerin sayısı artırılabilir. Bu çalışmada sadece yazılımın büyüklüğü, satır kod sayısı, çalışan kişi sayısı ve yazılımdaki bozunma oranı kullanılmıştır. Faktör sayısındaki artış miktarı doğru orantılı olarak yapay sinir ağı uygulamasındaki girdi katmanına verilen veri sayısını artırır. Buna bağlı olarak örnek veri seti sayısını da artırmak gerekmektedir. Dolayısıyla gerekli olan örnek veri seti için çok daha fazla yazılım projesi verilerine ihtiyaç duyulur. Yazılım firmaları gerçekleşmiş yazılımların ve tahminlerin verilerini veritabanında tutarak gerçekleşecek tahminler için saklamalıdır. Ayrıca yazılım maliyet tahmini yanında yazılım kod satır sayısı tahminide yaparak proje zaman yönetimini de daha sağlıklı yapabilirler.

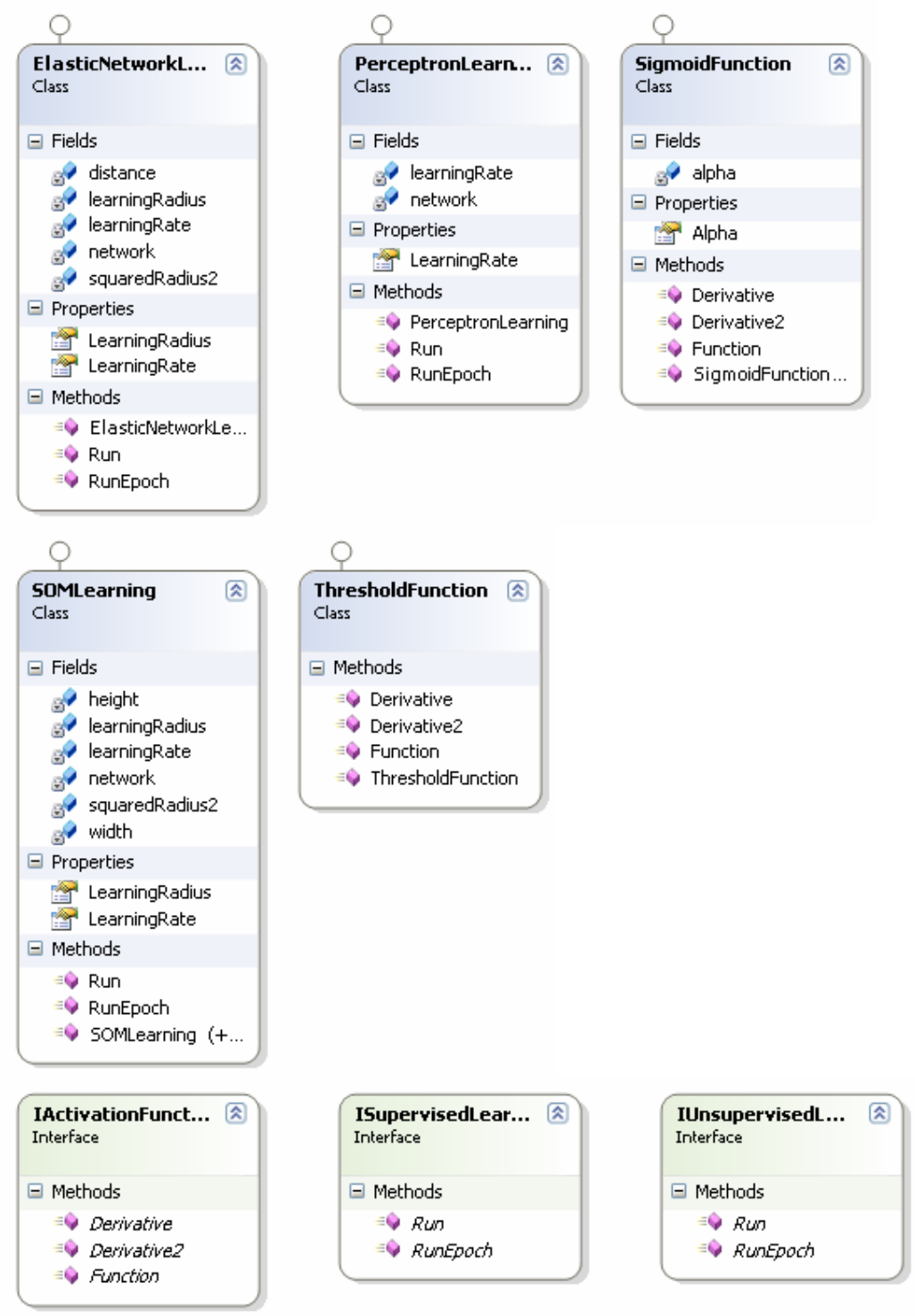
Yapılan başarılı maliyet ve zaman tahmini bir yazılım projesinin başarılı sayılabilmesi için yeterli değildir. Yapılan iyi tahminler doğru gerçekleştirilen gereksinim analizi ve iyi bir planlama ile müşteri beklentilerini sağlayarak hedefe ulaşabilir.

## EK-1

## Yapay Sinir Ağı Uygulamasının Class Diyagramı




Layer
Abstract Class
Fields
inputsCount
neurons
neuronsCount
output
Properties
InputsCount
NeuronsCount
Output
this
Methods
Compute
Layer
Randomize
DistanceLayer
Class
Layer
Properties
this
Methods
DistanceLayer
ActivationLayer
Class
Layer
Properties
this
Methods
ActivationLayer
BackPropagatio...
Class
Fields
learningRate
momentum
network
neuronErrors
thresholdsUpdates
weightsUpdates
Properties
LearningRate
Momentum
Methods
BackPropagation...
CalculateError
CalculateUpdates
Run
RunEpoch
UpdateNetwork
BipolarSigmoidF...
Class
Fields
alpha
Properties
Alpha
Methods
BipolarSigmoidFu...
Derivative
Derivative2
Function
DeltaRuleLearning
Class
Fields
learningRate
network
Properties
LearningRate
Methods
DeltaRuleLearning
Run
RunEpoch


ElasticNetworkL...
Class
Fields
distance
learningRadius
learningRate
network
squaredRadius2
Properties
LearningRadius
LearningRate
Methods
ElasticNetworkLe...
Run
RunEpoch
PerceptronLearn...
Class
Fields
learningRate
network
Properties
LearningRate
Methods
PerceptronLearning
Run
RunEpoch
SigmoidFunction
Class
Fields
alpha
Properties
Alpha
Methods
Derivative
Derivative2
Function
SigmoidFunction...
SOMLearning
Class
Fields
height
learningRadius
learningRate
network
squaredRadius2
width
Properties
LearningRadius
LearningRate
Methods
Run
RunEpoch
SOMLearning (+...
ThresholdFunction
Class
Methods
Derivative
Derivative2
Function
ThresholdFunction
IActivationFunct...
Interface
Methods
Derivative
Derivative2
Function
ISupervisedLear...
Interface
Methods
Run
RunEpoch
IUnsupervisedL...
Interface
Methods
Run
RunEpoch

**EK-2**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace SoftWareNeuralNetwork
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new SoftWareNeuralNetworkForm());
        }
    }
}


namespace SoftWareNeuralNetwork
{
    partial class SoftWareNeuralNetworkForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should
be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.m_textBoxMomentum = new
System.Windows.Forms.TextBox();
```

```
            this.m_textBoxIteration = new
System.Windows.Forms.TextBox();
            this.m_buttonTrain = new System.Windows.Forms.Button();
            this.m_groupBoxTrain = new
System.Windows.Forms.GroupBox();
            this.m_buttonOutputBrowse = new
System.Windows.Forms.Button();
            this.m_buttonInputBrowse = new
System.Windows.Forms.Button();
            this.m_textBoxOutputFilePath = new
System.Windows.Forms.TextBox();
            this.label4 = new System.Windows.Forms.Label();
            this.m_textBoxInputFilePath = new
System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.m_groupBoxTest = new
System.Windows.Forms.GroupBox();
            this.label6 = new System.Windows.Forms.Label();
            this.m_labelResult = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.m_buttonTestInputBrowse = new
System.Windows.Forms.Button();
            this.m_textBoxTestInputPath = new
System.Windows.Forms.TextBox();
            this.m_buttonTest = new System.Windows.Forms.Button();
            this.m_groupBoxTrain.SuspendLayout();
            this.m_groupBoxTest.SuspendLayout();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Microsoft
Sans Serif", 11.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(162)));
            this.label1.Location = new System.Drawing.Point(17, 131);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(92, 18);
            this.label1.TabIndex = 0;
            this.label1.Text = "Momentum";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Font = new System.Drawing.Font("Microsoft
Sans Serif", 11.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(162)));
            this.label2.Location = new System.Drawing.Point(17, 165);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(77, 18);
            this.label2.TabIndex = 0;
            this.label2.Text = "İterasyon";
            //
            // m_textBoxMomentum
            //
            this.m_textBoxMomentum.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_textBoxMomentum.Location = new
System.Drawing.Point(144, 128);
```

```
            this.m_textBoxMomentum.Name = "m_textBoxMomentum";
            this.m_textBoxMomentum.Size = new
System.Drawing.Size(106, 24);
            this.m_textBoxMomentum.TabIndex = 0;
            this.m_textBoxMomentum.Text = "0";
            this.m_textBoxMomentum.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.m_textBoxMomentum_KeyP
ress);
            //
            // m_textBoxIteration
            //
            this.m_textBoxIteration.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_textBoxIteration.Location = new
System.Drawing.Point(144, 158);
            this.m_textBoxIteration.Name = "m_textBoxIteration";
            this.m_textBoxIteration.Size = new
System.Drawing.Size(106, 24);
            this.m_textBoxIteration.TabIndex = 1;
            this.m_textBoxIteration.Text = "1000";
            this.m_textBoxIteration.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.m_textBoxIteration_Key
Press);
            //
            // m_buttonTrain
            //
            this.m_buttonTrain.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_buttonTrain.Location = new
System.Drawing.Point(430, 145);
            this.m_buttonTrain.Name = "m_buttonTrain";
            this.m_buttonTrain.Size = new System.Drawing.Size(61,
37);
            this.m_buttonTrain.TabIndex = 2;
            this.m_buttonTrain.Text = "Eğit";
            this.m_buttonTrain.UseVisualStyleBackColor = true;
            this.m_buttonTrain.Click += new
System.EventHandler(this.m_buttonTrain_Click);
            //
            // m_groupBoxTrain
            //

this.m_groupBoxTrain.Controls.Add(this.m_buttonOutputBrowse);

this.m_groupBoxTrain.Controls.Add(this.m_buttonInputBrowse);

this.m_groupBoxTrain.Controls.Add(this.m_textBoxOutputFilePath);
            this.m_groupBoxTrain.Controls.Add(this.label4);

this.m_groupBoxTrain.Controls.Add(this.m_textBoxInputFilePath);
            this.m_groupBoxTrain.Controls.Add(this.label3);
            this.m_groupBoxTrain.Controls.Add(this.m_buttonTrain);

this.m_groupBoxTrain.Controls.Add(this.m_textBoxIteration);

this.m_groupBoxTrain.Controls.Add(this.m_textBoxMomentum);
            this.m_groupBoxTrain.Controls.Add(this.label2);
```

```
            this.m_groupBoxTrain.Controls.Add(this.label1);
            this.m_groupBoxTrain.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_groupBoxTrain.Location = new
System.Drawing.Point(24, 12);
            this.m_groupBoxTrain.Name = "m_groupBoxTrain";
            this.m_groupBoxTrain.Size = new System.Drawing.Size(553,
213);
            this.m_groupBoxTrain.TabIndex = 4;
            this.m_groupBoxTrain.TabStop = false;
            this.m_groupBoxTrain.Text = "Eğitim";
            //
            // m_buttonOutputBrowse
            //
            this.m_buttonOutputBrowse.Font = new
System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_buttonOutputBrowse.Location = new
System.Drawing.Point(430, 84);
            this.m_buttonOutputBrowse.Name = "m_buttonOutputBrowse";
            this.m_buttonOutputBrowse.Size = new
System.Drawing.Size(61, 22);
            this.m_buttonOutputBrowse.TabIndex = 5;
            this.m_buttonOutputBrowse.Text = "Gözat";
            this.m_buttonOutputBrowse.UseVisualStyleBackColor = true;
            this.m_buttonOutputBrowse.Click += new
System.EventHandler(this.m_buttonOutputBrowse_Click);
            //
            // m_buttonInputBrowse
            //
            this.m_buttonInputBrowse.Font = new
System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_buttonInputBrowse.Location = new
System.Drawing.Point(430, 40);
            this.m_buttonInputBrowse.Name = "m_buttonInputBrowse";
            this.m_buttonInputBrowse.Size = new
System.Drawing.Size(61, 22);
            this.m_buttonInputBrowse.TabIndex = 5;
            this.m_buttonInputBrowse.Text = "Gözat";
            this.m_buttonInputBrowse.UseVisualStyleBackColor = true;
            this.m_buttonInputBrowse.Click += new
System.EventHandler(this.m_buttonInputBrowse_Click);
            //
            // m_textBoxOutputFilePath
            //
            this.m_textBoxOutputFilePath.Font = new
System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_textBoxOutputFilePath.Location = new
System.Drawing.Point(144, 84);
            this.m_textBoxOutputFilePath.Name =
"m_textBoxOutputFilePath";
            this.m_textBoxOutputFilePath.Size = new
System.Drawing.Size(263, 22);
            this.m_textBoxOutputFilePath.TabIndex = 4;
```

```
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Font = new System.Drawing.Font("Microsoft
Sans Serif", 9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(162)));
            this.label4.Location = new System.Drawing.Point(17, 83);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(103, 16);
            this.label4.TabIndex = 3;
            this.label4.Text = "Çıkış Dosyası";
            //
            // m_textBoxInputFilePath
            //
            this.m_textBoxInputFilePath.Font = new
System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_textBoxInputFilePath.Location = new
System.Drawing.Point(144, 40);
            this.m_textBoxInputFilePath.Name =
"m_textBoxInputFilePath";
            this.m_textBoxInputFilePath.Size = new
System.Drawing.Size(263, 22);
            this.m_textBoxInputFilePath.TabIndex = 4;
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Font = new System.Drawing.Font("Microsoft
Sans Serif", 9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(162)));
            this.label3.Location = new System.Drawing.Point(17, 39);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(101, 16);
            this.label3.TabIndex = 3;
            this.label3.Text = "Giriş Dosyası";
            //
            // m_groupBoxTest
            //
            this.m_groupBoxTest.Controls.Add(this.label6);
            this.m_groupBoxTest.Controls.Add(this.m_labelResult);
            this.m_groupBoxTest.Controls.Add(this.label5);

this.m_groupBoxTest.Controls.Add(this.m_buttonTestInputBrowse);

this.m_groupBoxTest.Controls.Add(this.m_textBoxTestInputPath);
            this.m_groupBoxTest.Controls.Add(this.m_buttonTest);
            this.m_groupBoxTest.Enabled = false;
            this.m_groupBoxTest.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_groupBoxTest.Location = new
System.Drawing.Point(27, 247);
            this.m_groupBoxTest.Name = "m_groupBoxTest";
            this.m_groupBoxTest.Size = new System.Drawing.Size(550,
195);
            this.m_groupBoxTest.TabIndex = 5;
            this.m_groupBoxTest.TabStop = false;
```

```
            this.m_groupBoxTest.Text = "Test";
            //
            // label6
            //
            this.label6.AutoSize = true;
            this.label6.Location = new System.Drawing.Point(14, 91);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(61, 18);
            this.label6.TabIndex = 6;
            this.label6.Text = "Maliyet";
            //
            // m_labelResult
            //
            this.m_labelResult.AutoSize = true;
            this.m_labelResult.Location = new
System.Drawing.Point(138, 91);
            this.m_labelResult.Name = "m_labelResult";
            this.m_labelResult.Size = new System.Drawing.Size(0, 18);
            this.m_labelResult.TabIndex = 6;
            //
            // label5
            //
            this.label5.AutoSize = true;
            this.label5.Font = new System.Drawing.Font("Microsoft
Sans Serif", 9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(162)));
            this.label5.Location = new System.Drawing.Point(14, 45);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(101, 16);
            this.label5.TabIndex = 3;
            this.label5.Text = "Giriş Dosyası";
            //
            // m_buttonTestInputBrowse
            //
            this.m_buttonTestInputBrowse.Font = new
System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_buttonTestInputBrowse.Location = new
System.Drawing.Point(427, 46);
            this.m_buttonTestInputBrowse.Name =
"m_buttonTestInputBrowse";
            this.m_buttonTestInputBrowse.Size = new
System.Drawing.Size(61, 22);
            this.m_buttonTestInputBrowse.TabIndex = 5;
            this.m_buttonTestInputBrowse.Text = "Gözat";
            this.m_buttonTestInputBrowse.UseVisualStyleBackColor =
true;
            this.m_buttonTestInputBrowse.Click += new
System.EventHandler(this.m_buttonTestInputBrowse_Click);
            //
            // m_textBoxTestInputPath
            //
            this.m_textBoxTestInputPath.Font = new
System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_textBoxTestInputPath.Location = new
System.Drawing.Point(141, 46);
            this.m_textBoxTestInputPath.Name =
"m_textBoxTestInputPath";
```

```
            this.m_textBoxTestInputPath.Size = new
System.Drawing.Size(263, 22);
            this.m_textBoxTestInputPath.TabIndex = 4;
            //
            // m_buttonTest
            //
            this.m_buttonTest.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(162)));
            this.m_buttonTest.Location = new
System.Drawing.Point(245, 133);
            this.m_buttonTest.Name = "m_buttonTest";
            this.m_buttonTest.Size = new System.Drawing.Size(61, 37);
            this.m_buttonTest.TabIndex = 2;
            this.m_buttonTest.Text = "Test";
            this.m_buttonTest.UseVisualStyleBackColor = true;
            this.m_buttonTest.Click += new
System.EventHandler(this.m_buttonTest_Click);
            //
            // SoftWareNeuralNetworkForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F,
13F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(605, 494);
            this.Controls.Add(this.m_groupBoxTest);
            this.Controls.Add(this.m_groupBoxTrain);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedToolWindow;
            this.Name = "SoftWareNeuralNetworkForm";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Yapay Sinir Ağları ile Yazılım Maliyet
Hesabı";
            this.m_groupBoxTrain.ResumeLayout(false);
            this.m_groupBoxTrain.PerformLayout();
            this.m_groupBoxTest.ResumeLayout(false);
            this.m_groupBoxTest.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox m_textBoxMomentum;
        private System.Windows.Forms.TextBox m_textBoxIteration;
        private System.Windows.Forms.Button m_buttonTrain;
        private System.Windows.Forms.GroupBox m_groupBoxTrain;
        private System.Windows.Forms.TextBox m_textBoxOutputFilePath;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.TextBox m_textBoxInputFilePath;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Button m_buttonOutputBrowse;
        private System.Windows.Forms.Button m_buttonInputBrowse;
        private System.Windows.Forms.GroupBox m_groupBoxTest;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Label m_labelResult;
```

```
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Button m_buttonTestInputBrowse;
        private System.Windows.Forms.TextBox m_textBoxTestInputPath;
        private System.Windows.Forms.Button m_buttonTest;
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO;
using AForge;
using AForge.Neuro;
using AForge.Neuro.Learning;

namespace SoftWareNeuralNetwork
{
    public partial class SoftWareNeuralNetworkForm : Form
    {
        private string m_currentDirectory;
        private string m_trainInputFilePath, m_trainOutputFilePath,
m_testInputFilePath;
        private ActivationNetwork m_network;

        public SoftWareNeuralNetworkForm()
        {
            InitializeComponent();
            m_currentDirectory = Environment.CurrentDirectory;
        }
        private static double getDoubleFromString(string str)
        {
            int index = 0;

            if ((index = str.IndexOf(',')) == -1)
                if ((index = str.IndexOf('.')) == -1)
                    return double.Parse(str);

            return index > 0 ? double.Parse(str.Substring(0, index))
+ int.Parse(str.Substring(index + 1)) * Math.Pow(10, -
str.Substring(index + 1).Length) : int.Parse(str.Substring(index +
1)) * Math.Pow(10, -str.Substring(index + 1).Length);
        }
        private static Thread creatThread(ThreadStart threadStart,
bool bIsBackground, bool bStart)
        {
            Thread result = new Thread(threadStart);

            result.IsBackground = bIsBackground;
            if (bStart)
                result.Start();

            return result;
        }
        private static Thread creatThread(ParameterizedThreadStart
threadStart, bool bIsBackground, bool bStart, object o)
        {
```

```
            Thread result = new Thread(threadStart);

            result.IsBackground = bIsBackground;
            if (bStart)
                result.Start(o);

            return result;
        }
        private void trainThreadProc()
        {
            StreamReader srInput = null, srOutput =  null;
            List<string> inputsList = new List<string>();
            List<string> outputsList = new List<string>();
            string line = null;

            try
            {
                srInput = new StreamReader(m_trainInputFilePath);
                srOutput = new StreamReader(m_trainOutputFilePath);

                while ((line = srInput.ReadLine()) != null)
                    inputsList.Add(line);

                while ((line = srOutput.ReadLine()) != null)
                    outputsList.Add(line);

                if (inputsList.Count != outputsList.Count)
                {
                    MessageBox.Show("Giriş verileri Çıkış Verileri
ile uyuşmuyor");
                    return;
                }

                double[][] inputVector = new
double[inputsList.Count][];
                double[][] outputVector = new
double[outputsList.Count][];

                for (int i = 0; i < inputsList.Count; ++i)
                {
                    inputVector[i] = new double[3];

                    string[] values = inputsList[i].Split(' ', '\t');
                    //3 den küçük hatası kontrolü

                    for (int j = 0; j < 3; ++j)
                    {
                        string tempStr = values[j].TrimStart(' ',
'\t');

                        inputVector[i][j] =
getDoubleFromString(tempStr.TrimEnd(' ', '\t'));
                    }
                }

                for (int i = 0; i < outputsList.Count; ++i)
                {
                    outputVector[i] = new double[1];

                    string[] values = inputsList[i].Split(' ', '\t');
```

```
                    string tempStr = values[0].TrimStart(' ', '\t');

                    outputVector[i][0] =
getDoubleFromString(tempStr.TrimEnd(' ', '\t'));
                }
                m_network = new ActivationNetwork(new
SigmoidFunction(), 3, 4, 1);
                BackPropagationLearning trainer = new
BackPropagationLearning(m_network);

                trainer.Momentum =
getDoubleFromString(m_textBoxMomentum.Text);
                int iterationCount =
int.Parse(m_textBoxIteration.Text);

                for (int i = 0; i < iterationCount; ++i)
                {
                    trainer.RunEpoch(inputVector, outputVector);
                }
                m_groupBoxTest.Enabled = true;
                MessageBox.Show("Eğitim Tamamlandı");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                if (srInput != null)
                    srInput.Close();
                if (srOutput != null)
                    srOutput.Close();
            }
        }
        private void testThreadProc()
        {
            StreamReader srInput = null;

            try
            {
                srInput = new StreamReader(m_testInputFilePath);

                string line = srInput.ReadLine();

                if (line == null)
                {
                    MessageBox.Show("Giriş Dosyasında veri yok");
                    return;
                }

                string[] values = line.Split(' ', '\t');
                //3 den küçük kontrolü

                double[] inputs = new double[3];

                for (int i = 0; i < 3; ++i)
                {
                    string tempStr = values[i].TrimStart(' ', '\t');

                    inputs[i] = getDoubleFromString(tempStr.TrimEnd('
', '\t'));
```

```
                }
                m_labelResult.Text = (m_network.Compute(inputs)[0] *
Math.Pow(10,7 )).ToString();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                if (srInput != null)
                    srInput.Close();
            }
        }
        private void m_textBoxMomentum_KeyPress(object sender,
KeyPressEventArgs e)
        {
            if (e.KeyChar < '0' || e.KeyChar > '9')
                if (e.KeyChar != (char)Keys.Back && e.KeyChar != ',')
                    e.Handled = true;

            if (e.KeyChar == ',' &&
m_textBoxMomentum.Text.IndexOf(',') != -1)
                e.Handled = true;
        }

        private void m_textBoxIteration_KeyPress(object sender,
KeyPressEventArgs e)
        {
            if (e.KeyChar < '0' || e.KeyChar > '9')
                if (e.KeyChar != (char)Keys.Back && e.KeyChar != ',')
                    e.Handled = true;

            if (e.KeyChar == ',' &&
m_textBoxIteration.Text.IndexOf(',') != -1)
                e.Handled = true;
        }

        private void m_buttonTrain_Click(object sender, EventArgs e)
        {
            m_groupBoxTest.Enabled = false;
            creatThread(new ThreadStart(trainThreadProc), true,
true);
        }

        private void m_buttonInputBrowse_Click(object sender,
EventArgs e)
        {
            OpenFileDialog dlg = new OpenFileDialog();

            if (dlg.ShowDialog() != DialogResult.OK)
                return;

            m_trainInputFilePath = m_textBoxInputFilePath.Text =
dlg.FileName;
        }

        private void m_buttonOutputBrowse_Click(object sender,
EventArgs e)
        {
```

```
            OpenFileDialog dlg = new OpenFileDialog();

            if (dlg.ShowDialog() != DialogResult.OK)
                return;

            m_trainOutputFilePath = m_textBoxOutputFilePath.Text =
dlg.FileName;
        }

        private void m_buttonTestInputBrowse_Click(object sender,
EventArgs e)
        {
            OpenFileDialog dlg = new OpenFileDialog();

            if (dlg.ShowDialog() != DialogResult.OK)
                return;

            m_testInputFilePath = m_textBoxTestInputPath.Text =
dlg.FileName;
        }

        private void m_buttonTest_Click(object sender, EventArgs e)
        {
            creatThread(new ThreadStart(testThreadProc), true, true);
        }
    }
}

// AForge Neural Net Library
//
// Copyright © Andrew Kirillov, 2005-2006
// andrew.kirillov@gmail.com
//

namespace AForge.Neuro
{
      using System;

      /// <summary>
      /// Bipolar sigmoid activation function
      /// </summary>
      ///
      /// <remarks>The class represents bipolar sigmoid activation
function with
      /// the next expression:<br />
      /// <code>
      ///                 2
      /// f(x) = ------------------ - 1
      ///        1 + exp(-alpha * x)
      ///
      ///           2 * alpha * exp(-alpha * x )
      /// f'(x) = ------------------------------- = alpha * (1 -
f(x)^2) / 2
      ///           (1 + exp(-alpha * x))^2
      /// </code>
      /// Output range of the function: <b>[-1, 1]</b><br /><br />
      /// Functions graph:<br />
      /// <img src="sigmoid_bipolar.bmp" width="242" height="172" />
      /// </remarks>
      public class BipolarSigmoidFunction : IActivationFunction
      {
```

```
            // sigmoid's alpha value
            private double alpha = 2;

            /// <summary>
            /// Sigmoid's alpha value
            /// </summary>
            ///
            /// <remarks>The value determines steepness of the
function. Default value: <b>2</b>.
            /// </remarks>
            public double Alpha
            {
                  get { return alpha; }
                  set { alpha = value; }
            }

            /// <summary>
            /// Initializes a new instance of the <see
cref="SigmoidFunction"/> class
            /// </summary>
            public BipolarSigmoidFunction( ) { }

            /// <summary>
            /// Initializes a new instance of the <see
cref="BipolarSigmoidFunction"/> class
            /// </summary>
            ///
            /// <param name="alpha">Sigmoid's alpha value</param>
            public BipolarSigmoidFunction( double alpha )
            {
                  this.alpha = alpha;
            }

            /// <summary>
            /// Calculates function value
            /// </summary>
            ///
            /// <param name="x">Function input value</param>
            ///
            /// <returns>Function output value, <i>f(x)</i></returns>
            ///
            /// <remarks>The method calculates function value at point
<b>x</b>.</remarks>
            ///
            public double Function( double x )
            {
                  return ( ( 2 / ( 1 + Math.Exp( -alpha * x ) ) ) - 1
);
            }

            /// <summary>
            /// Calculates function derivative
            /// </summary>
            ///
            /// <param name="x">Function input value</param>
            ///
            /// <returns>Function derivative, <i>f'(x)</i></returns>
            ///
            /// <remarks>The method calculates function derivative at
point <b>x</b>.</remarks>
            ///
```

```
            public double Derivative( double x )
            {
                  double y = Function( x );

                  return ( alpha * ( 1 - y * y ) / 2 );
            }

            /// <summary>
            /// Calculates function derivative
            /// </summary>
            ///
            /// <param name="y">Function output value - the value,
which was obtained
            /// with the help of <see cref="Function"/> method</param>
            ///
            /// <returns>Function derivative, <i>f'(x)</i></returns>
            ///
            /// <remarks>The method calculates the same derivative
value as the
            /// <see cref="Derivative"/> method, but it takes not the
input <b>x</b> value
            /// itself, but the function value, which was calculated
previously with
            /// the help of <see cref="Function"/> method. <i>(Some
applications require as
            /// function value, as derivative value, so they can seve
the amount of
            /// calculations using this method to calculate
derivative)</i></remarks>
            ///
            public double Derivative2( double y )
            {
                  return ( alpha * ( 1 - y * y ) / 2 );
            }
      }
}

// AForge Neural Net Library
//
// Copyright © Andrew Kirillov, 2005-2006
// andrew.kirillov@gmail.com
//

namespace AForge.Neuro.Learning
{
      using System;

      /// <summary>
      /// Back propagation learning algorithm
      /// </summary>
      ///
      /// <remarks>The class implements back propagation learning
algorithm,
      /// which is widely used for training multi-layer neural
networks with
      /// continuous activation functions.</remarks>
      ///
      public class BackPropagationLearning : ISupervisedLearning
      {
            // network to teach
            private ActivationNetwork network;
```

```
            // learning rate
            private double learningRate = 0.1;
            // momentum
            private double momentum = 0.0;

            // neuron's errors
            private double[][]           neuronErrors = null;
            // weight's updates
            private double[][][]    weightsUpdates = null;
            // threshold's updates
            private double[][]           thresholdsUpdates = null;


            /// <summary>
            /// Learning rate
            /// </summary>
            ///
            /// <remarks>The value determines speed of learning.
Default value equals to 0.1.</remarks>
            ///
            public double LearningRate
            {
                  get { return learningRate; }
                  set
                  {
                        learningRate = Math.Max( 0.0, Math.Min( 1.0,
value ) );
                  }
            }

            /// <summary>
            /// Momentum
            /// </summary>
            ///
            /// <remarks>The value determines the portion of previous
weight's update
            /// to use on current iteration. Weight's update values
are calculated on
            /// each iteration depending on neuron's error. The
momentum specifies the amount
            /// of update to use from previous iteration and the
amount of update
            /// to use from current iteration. If the value is equal
to 0.1, for example,
            /// then 0.1 portion of previous update and 0.9 portion of
current update are used
            /// to update weight's value.<br /><br />
            ///   Default value equals to 0.0.</remarks>
            ///
            public double Momentum
            {
                  get { return momentum; }
                  set
                  {
                        momentum = Math.Max( 0.0, Math.Min( 1.0, value
) );
                  }
            }

            /// <summary>
```

```
            /// Initializes a new instance of the <see
cref="BackPropagationLearning"/> class
            /// </summary>
            ///
            /// <param name="network">Network to teach</param>
            ///
            public BackPropagationLearning( ActivationNetwork network
)
            {
                  this.network = network;

                  // create error and deltas arrays
                  neuronErrors = new double[network.LayersCount][];
                  weightsUpdates = new
double[network.LayersCount][][];
                  thresholdsUpdates = new
double[network.LayersCount][];

                  // initialize errors and deltas arrays for each
layer
                  for ( int i = 0, n = network.LayersCount; i < n; i++
)
                  {
                        Layer layer = network[i];

                        neuronErrors[i] = new
double[layer.NeuronsCount];
                        weightsUpdates[i] = new
double[layer.NeuronsCount][];
                        thresholdsUpdates[i] = new
double[layer.NeuronsCount];

                        // for each neuron
                        for ( int j = 0; j < layer.NeuronsCount; j++ )
                        {
                              weightsUpdates[i][j] = new
double[layer.InputsCount];
                        }
                  }
            }

            /// <summary>
            /// Runs learning iteration
            /// </summary>
            ///
            /// <param name="input">input vector</param>
            /// <param name="output">desired output vector</param>
            ///
            /// <returns>Returns squared error of the last layer
divided by 2</returns>
            ///
            /// <remarks>Runs one learning iteration and updates
neuron's
            /// weights.</remarks>
            ///
            public double Run( double[] input, double[] output )
            {
                  // compute the network's output
                  network.Compute( input );

                  // calculate network error
```

```
                double error = CalculateError( output );

                // calculate weights updates
                CalculateUpdates( input );

                // update the network
                UpdateNetwork( );

                return error;
            }

            /// <summary>
            /// Runs learning epoch
            /// </summary>
            ///
            /// <param name="input">array of input vectors</param>
            /// <param name="output">array of output vectors</param>
            ///
            /// <returns>Returns sum of squared errors of the last
layer divided by 2</returns>
            ///
            /// <remarks>Runs series of learning iterations - one
iteration
            /// for each input sample. Updates neuron's weights after
each sample
            /// presented.</remarks>
            ///
            public double RunEpoch( double[][] input, double[][]
output )
            {
                double error = 0.0;

                // run learning procedure for all samples
                for ( int i = 0, n = input.Length; i < n; i++ )
                {
                    error += Run( input[i], output[i] );
                }

                // return summary error
                return error;
            }


            /// <summary>
            /// Calculates error values for all neurons of the network
            /// </summary>
            ///
            /// <param name="desiredOutput">Desired output
vector</param>
            ///
            /// <returns>Returns summary squared error of the last
layer divided by 2</returns>
            ///
            private double CalculateError( double[] desiredOutput )
            {
                // current and the next layers
                ActivationLayer   layer, layerNext;
                // current and the next errors arrays
                double[] errors, errorsNext;
                // error values
                double error = 0, e, sum;
```

```
                    // neuron's output value
                    double output;
                    // layers count
                    int layersCount = network.LayersCount;

                    // assume, that all neurons of the network have the
same activation function
                    IActivationFunction     function =
network[0][0].ActivationFunction;

                    // calculate error values for the last layer first
                    layer = network[layersCount - 1];
                    errors      = neuronErrors[layersCount - 1];

                    for ( int i = 0, n = layer.NeuronsCount; i < n; i++
)
                    {
                          output = layer[i].Output;
                          // error of the neuron
                          e = desiredOutput[i] - output;
                          // error multiplied with activation function's
derivative
                          errors[i] = e * function.Derivative2( output
);
                          // squre the error and sum it
                          error += ( e * e );
                    }

                    // calculate error values for other layers
                    for ( int j = layersCount - 2; j >= 0; j-- )
                    {
                          layer       = network[j];
                          layerNext   = network[j + 1];
                          errors            = neuronErrors[j];
                          errorsNext  = neuronErrors[j + 1];

                          // for all neurons of the layer
                          for ( int i = 0, n = layer.NeuronsCount; i <
n; i++ )
                          {
                                sum = 0.0;
                                // for all neurons of the next layer
                                for ( int k = 0, m =
layerNext.NeuronsCount; k < m; k++ )
                                {
                                      sum += errorsNext[k] *
layerNext[k][i];
                                }
                                errors[i] = sum * function.Derivative2(
layer[i].Output );
                          }
                    }

                    // return squared error of the last layer divided by
2
                    return error / 2.0;
              }

              /// <summary>
              /// Calculate weights updates
              /// </summary>
```

```
        ///
        /// <param name="input">Network's input vector</param>
        ///
        private void CalculateUpdates( double[] input )
        {
            // current neuron
            ActivationNeuron  neuron;
            // current and previous layers
            ActivationLayer         layer, layerPrev;
            // layer's weights updates
            double[][]  layerWeightsUpdates;
            // layer's thresholds updates
            double[]    layerThresholdUpdates;
            // layer's error
            double[]    errors;
            // neuron's weights updates
            double[]    neuronWeightUpdates;
            // error value
            double            error;

            // 1 - calculate updates for the last layer fisrt
            layer = network[0];
            errors = neuronErrors[0];
            layerWeightsUpdates = weightsUpdates[0];
            layerThresholdUpdates = thresholdsUpdates[0];

            // for each neuron of the layer
            for ( int i = 0, n = layer.NeuronsCount; i < n; i++
)
            {
                neuron      = layer[i];
                error = errors[i];
                neuronWeightUpdates     =
layerWeightsUpdates[i];

                // for each weight of the neuron
                for ( int j = 0, m = neuron.InputsCount; j <
m; j++ )
                {
                    // calculate weight update
                    neuronWeightUpdates[j] = learningRate *
(
                        momentum * neuronWeightUpdates[j]
+
                        ( 1.0 - momentum ) * error *
input[j]
                        );
                }

                // calculate treshold update
                layerThresholdUpdates[i] = learningRate * (
                    momentum * layerThresholdUpdates[i] +
                    ( 1.0 - momentum ) * error
                    );
            }

            // 2 - for all other layers
            for ( int k = 1, l = network.LayersCount; k < l; k++
)
            {
                layerPrev               = network[k - 1];
```

```
                        layer                   = network[k];
                        errors                  =
neuronErrors[k];
                        layerWeightsUpdates     = weightsUpdates[k];
                        layerThresholdUpdates = thresholdsUpdates[k];

                        // for each neuron of the layer
                        for ( int i = 0, n = layer.NeuronsCount; i <
n; i++ )
                        {
                              neuron      = layer[i];
                              error = errors[i];
                              neuronWeightUpdates     =
layerWeightsUpdates[i];

                              // for each synapse of the neuron
                              for ( int j = 0, m = neuron.InputsCount;
j < m; j++ )
                              {
                                    // calculate weight update
                                    neuronWeightUpdates[j] =
learningRate * (
                                          momentum *
neuronWeightUpdates[j] +
                                          ( 1.0 - momentum ) * error *
layerPrev[j].Output
                                          );
                              }

                              // calculate treshold update
                              layerThresholdUpdates[i] = learningRate
* (
                                    momentum *
layerThresholdUpdates[i] +
                                    ( 1.0 - momentum ) * error
                                    );
                        }
                  }
            }

            /// <summary>
            /// Update network'sweights
            /// </summary>
            ///
            private void UpdateNetwork( )
            {
                  // current neuron
                  ActivationNeuron  neuron;
                  // current layer
                  ActivationLayer         layer;
                  // layer's weights updates
                  double[][]  layerWeightsUpdates;
                  // layer's thresholds updates
                  double[]    layerThresholdUpdates;
                  // neuron's weights updates
                  double[]    neuronWeightUpdates;

                  // for each layer of the network
                  for ( int i = 0, n = network.LayersCount; i < n; i++
)
                  {
```

```
                        layer = network[i];
                        layerWeightsUpdates = weightsUpdates[i];
                        layerThresholdUpdates = thresholdsUpdates[i];

                        // for each neuron of the layer
                        for ( int j = 0, m = layer.NeuronsCount; j <
m; j++ )
                        {
                              neuron = layer[j];
                              neuronWeightUpdates =
layerWeightsUpdates[j];

                              // for each weight of the neuron
                              for ( int k = 0, s = neuron.InputsCount;
k < s; k++ )
                              {
                                    // update weight
                                    neuron[k] +=
neuronWeightUpdates[k];
                              }
                              // update treshold
                              neuron.Threshold +=
layerThresholdUpdates[j];
                        }
                  }
            }
      }
}

// AForge Neural Net Library
//
// Copyright © Andrew Kirillov, 2005-2006
// andrew.kirillov@gmail.com
//

namespace AForge.Neuro
{
      using System;

      /// <summary>
      /// Activation network
      /// </summary>
      ///
      /// <remarks>Activation network is a base for multi-layer neural
network
      /// with activation functions. It consists of <see
cref="ActivationLayer">activation
      /// layers</see>.</remarks>
      ///
      public class ActivationNetwork : Network
      {
            /// <summary>
            /// Network's layers accessor
            /// </summary>
            ///
            /// <param name="index">Layer index</param>
            ///
            /// <remarks>Allows to access network's layer.</remarks>
            ///
            public new ActivationLayer this[int index]
            {
```

```
                get { return ( (ActivationLayer) layers[index] ); }
            }

            /// <summary>
            /// Initializes a new instance of the <see
cref="ActivationNetwork"/> class
            /// </summary>
            /// <param name="function">Activation function of neurons
of the network</param>
            /// <param name="inputsCount">Network's inputs
count</param>
            /// <param name="neuronsCount">Array, which specifies the
amount of neurons in
            /// each layer of the neural network</param>
            ///
            /// <remarks>The new network will be randomized (see <see
cref="ActivationNeuron.Randomize"/>
            /// method) after it is created.</remarks>
            ///
            /// <example>The following sample illustrates the usage of
<c>ActivationNetwork</c> class:
            /// <code>
            ///         // create activation network
            ///         ActivationNetwork network = new
ActivationNetwork(
            ///             new SigmoidFunction( ), // sigmoid
activation function
            ///             3,                      // 3 inputs
            ///             4, 1 );                 // 2 layers:
            ///                                 // 4 neurons in the
firs layer
            ///                                 // 1 neuron in the
second layer
            ///   </code>
            /// </example>
            ///
            public ActivationNetwork( IActivationFunction function,
int inputsCount, params int[] neuronsCount )
                                        : base( inputsCount,
neuronsCount.Length )
            {
                // create each layer
                for ( int i = 0; i < layersCount; i++ )
                {
                    layers[i] = new ActivationLayer(
                        // neurons count in the layer
                        neuronsCount[i],
                        // inputs count of the layer
                        ( i == 0 ) ? inputsCount :
neuronsCount[i - 1],
                        // activation function of the layer
                        function );
                }
            }
```

**KAYNAKLAR**

[1] Albrecth and Gaffney,"Software function, source lines of code, and development effort prediction": a software science validation, IEEE Trans. on Softw.Eng., 9(6),pp639-648, 1983.

[2] B.Barry, C.Bradford, H.Ellis, W.Chris, M.Ray, S.Richard, "The Cocomo II Software Cost Estimation model", International Society of Parametric Analysts 1995

[3] Boehm B.," Software Engineering Economics", Prentice Hall, 1981

[4] Chen, F.-C, "Back-propagation neural networks for nonlinear self-tuning adaptive control," IEEE Trans. Software Eng ISSN: 0272-1708 3697367,1990

[5] Dreger, " J.B. Function Point Analysis. Englewood Cliffs", Prentice Hall.1989

[6] Elmas Çetin, "Yapay Sinir Ağları", Seçkin Yayıncılık, Ankara 2003

[7] Grosberg, S." The adaptive brain I. Cognition, Learning, Reinforcement, and Rhytm, and The adaptive brain 2": Visions, Speech, Language and Motor Control, Elsevier/North-Holland Amsterdam, 1986

[8] Hilyard Joyand Tellhet Stephon," C# Cookbook Second Edition "published by O'Reilly Media USA,2004

[9] Hebb, D.O. "The organisation of Behavior, John WileySons", Newyork 1949

[10] Hopfield, J.J." Neural Networks and Physical sistems with emergent collective computational ağabeylities." Proc.Nat.Acad.Sci 79:2554-2558, 1982

[11] Hopfield, J.J." Learning algorithms and probability distributions in feed-forward and feed back Networks". Proc.Nat.Acad.Sci. 88:8429-8433, 1987

[12] İrfan Macit ,Yazılım Mühendisliği ve Araçları (CASE)

[13] International Function Point Users Group (IFPUG), "Function Point Counting Practices Manual",2000

[14] Karlik B.," Myoelectric Control of Multifunction Prosthesis Using Neural Networks", Yildiz Technical University, PhD, Thesis, Istanbul, Turkiye, 1994

[15] Krilolov A. "Neural Networks on C#", CodeProject

[16] Karlık B. Yapay Sinir Ağları ders notları Turkiye, 2007

[17] Kohonen, T. "Self organisation and Associative memory, Springer-Verlag", Berlin, 1984

[18] Minsky, M. Ve Papert, S." Perceptrons: An introduction to computational Geometry," MIT Pres, Cambridge, MA. 1969

[19] M. Ayyıldız, O. Kalıpsız "A Metric-Set and Model Suggestion for Better Software Project Cost Estimation" Proceedings of World Academy of Science and Tecchnology Volume1307-6884, 2007

[20] Mc Culloch, W.S., W.H.," A logical calculus of the ideas immanent in neural nets." Bull. Math. Biophys. 5:115-133, 1943

[21] Nogel Christian, Bill Evjen, G. Joy ,"Wrox Proffesional C# 2008" published by Wiley Publishing, USA 2008

[22] Öztemel Ercan, "Yapay Sinir Ağları", Papatya Yayıncılık, İstanbul 2003

[23] Park, R.E." Software size Measurement: A framework for Counting Source Statements" (CMU/SEI-92-TR-20), Pittsburgh, PA 1992

[24] Raul Rojas," Neural Networks A Systematic Introduction," ISBN 3-540-60505-3 Berlin Hidelberg New York 1996

[25] Rockford Lhotka,"C# 2005 Business Objects",publisshed by Apress, 2006

[26] Rosenblatt, F," Principles of Neuradynamics", Boks, Newyork, 1959

[27] Rosenblatt, F. "Principles of Neuradynamics: Perceptrons and the theory of brain mechanisms", Spartan Books, Washington D.C. 1962

[28] Rumelhart, D.E. Mc Clelland, J.L. PDP research Group 1986

[29] Simon Robinson, “Professional C# ” published by Wrox Pres, USA 2001

[30] Windrow, B., Hoff, M.” Adaptive switching circuits. IRE WESCON Convention Record” 4:96-104. 1960

# ÖZGEÇMİŞ

**KIMLIK BILGILERI**

Dogum Tarihi :10-03-78
Dogum Yeri :Mut-ICEL
Uyrugu :Turk

**EĞİTİM BİLGİLERİ**

Yüksek Lisans: T.C. Haliç Üniversitesi (Bilgisayar Mühendisliği)
Lisans: T.C. İstanbul Kültür Üniversitesi (Bilgisayar Mühendisliği ve Endüstri Mühendisliği (Çap))

**DENEYİM**

**E.P.N. (Fransa)** : Muhasebe Bolumunde (Bilanco, Stok Servisi, Siparis Takibi) Part Time
**SOMEBAT (Fransa)**: Almanya ya siparişlerin verilmesi ve takibi. Opticut programının hazırlanması. Siparişleri verilen placardların optimum şekilde kesilmesini sağlayacak çizelgelerin hazırlanması. Haftalık üretim kapasitesini belirlemek.
**ARCELIK** Üretim Planlama ve Kalite Departmanlarında sataj
**ARENA BİLGİSAYAR A.Ş.** Strateji ve İş Geliştirme departmanında aylık beyin fırtınası toplantısı için proje önerilerinde bulunmak. SAP modüllerinde destek sağlamak. ABAP editörlüğü yapmak. Firma kapsamında yürütülen projelerin alt yapı çalışmalarını hazırlamak.
**T.C HALİÇ ÜNİVERSİTESİ**: Araştırma görevlisi olarak çalışmaktayım.