**EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ**

**(YÜKSEK LİSANS TEZİ)**

# GÖRSEL YAZILIM GELİŞTİRME ORTAMI İLE BERABER BİR YAPAY SİNİR AĞI KÜTÜPHANESİ TASARIMI VE GERÇEKLEŞTİRİMİ

**Ahmet Cumhur KINACI**

Bilgisayar Mühendisliği Anabilim Dalı

Bilim Dalı Kodu : 619.01.00

**Sunuş Tarihi : 10.08.2006**

**Tez Danışmanı : Yrd. Doç. Dr. Aybars UĞUR**

BORNOVA – İZMİR

Ahmet Cumhur KINACI tarafından YÜKSEK LİSANS TEZİ olarak sunulan “Görsel Yazılım Geliştirme Ortamı ile Beraber Bir Yapay Sinir Ağı Kütüphanesi Tasarımı ve Gerçekleştirimi” başlıklı bu çalışma E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 10.08.2006 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

| Jüri Üyeleri: | | İmza |
|---|---|---|
| Jüri Başkanı | : Yrd. Doç. Dr. Aybars UĞUR | ..... |
| Raportör Üye | : Doç. Dr. Mustafa M. İNCEOĞLU | ..... |
| Üye | : Yrd. Doç. Dr. Muhammet G. CİNSDİKİCİ | ..... |

# ÖZET

## GÖRSEL YAZILIM GELİŞTİRME ORTAMI İLE BERABER BİR YAPAY SİNİR AĞI KÜTÜPHANESİ TASARIMI VE GERÇEKLEŞTİRİMİ

KINACI, Ahmet Cumhur

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç Dr. Aybars UĞUR

Ağustos 2006, 90 sayfa

Yapay zeka konusu bilgisayar biliminin doğuşuyla birlikte üzerinde çalışılan bir konu olmuştur. İnsan gibi düşünen, problem çözebilen yazılım ve donanım geliştirmenin günlük hayatta karşılaşılan bir çok sorunun çözümünde kullanılabileceği düşünülmüştür. Yapay sinir ağları bu anlamda yapay zeka konusu içerisinde önemli bir yere sahiptir. İnsan beyninin kolaylıkla çözebildiği bir çok problemin çözümünde başarıyla uygulanmaktadır.

Bu tez çalışmasında Algılayıcı, ADALINE, geriye yayılma, SOM ve LVQ öğrenme algoritmalarını destekleyen bir yapay sinir ağı kütüphanesi geliştirilmiştir. Ayrıca bu kütüphanenin yazılım geliştirme sürecinde zahmetsizce kullanılmasını sağlayan bir görsel geliştirme aracı da gerçekleştirilmiştir. Platform bağımsızlığı ve geniş kullanıcı kitlesine sahip olması özelliklerinden dolayı, geliştirme ortamı olarak Java tercih edilmiştir.

**Anahtar sözcükler:** Yapay sinir ağları, yazılım kütüphanesi, yazılım mühendisliği, görsel yazılım geliştirme ortamı, Java

# ABSTRACT

## DESIGN AND IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK LIBRARY WITH VISUAL DEVELOPMENT ENVIRONMENT

KINACI, Ahmet Cumhur

MSc. in Computer Engineering

Supervisor: Assistant Prof. Dr. Aybars UĞUR

August 2006, 90 pages

Artificial intelligence has been a topic studied on since beginning of computer science. It's been thought that developing hardware and software that able to think and solve problems like human do, can be helpful in dealing most of the daily problems. In that sense, artificial neural networks is very important in artificial intelligence. It's applied successfully on the problems that the human mind can easily solve.

In that work, an artificial neural network software library, which has Perceptron, ADALINE, backpropagation, SOM and LVQ learning algorithms , has been developed. Also, a visual development environment, which enables to easily use the library in software development process, has been implemented. For development environment Java was chosen, because of it's platform independence and wide user community.

**Keywords:** Artificial neural networks, software library, software engineering, visual software development environment, Java

## TEŞEKKÜR

Tezin hazırlanma sürecindeki değerli katkılarından dolayı tez danışmanım Yrd. Doç. Dr. Aybars Uğur'a teşekkürü bir borç bilirim. Eğitimim konusunda bana her türlü desteği veren aileme de çok teşekkür ederim.

# İÇİNDEKİLER

# ŞEKİLLER DİZİNİ

# KISALTMALAR

| | | |
|---|---|---|
| ADALINE | : | Adaptive Linear Neuron |
| ART | : | Adaptive Resonance Theory |
| EMF | : | Eclipse Modeling Framework |
| FANN | : | Fast Artificial Neural Network Library |
| GEF | : | Graphical Editing Framework |
| GMF | : | Graphical Modeling Framework |
| JOONE | : | Java Object Oriented Neural Engine |
| LVQ | : | Learning Vector Quantization |
| SNNS | : | Stuttgart Neural Network Simulator |
| SOM | : | Self-Organizing Map |
| YSA | : | Yapay Sinir Ağları |

# 1. GİRİŞ

Zeka özellikleri taşıyan yazılımlar gerçekleştirme isteği bilgisayar biliminin gelişmesi süresince hedeflenen amaçlardan olmuştur. Bu amaç için bir çok yaklaşım ve teknik geliştirilmiştir. Bunların hepsi birlikte yapay zeka bilim dalını oluşturur. Yapay zeka konusu altındaki başlıca konular sayılacak olursa; uzman sistemler, bulanık mantık, genetik algoritmalar ve yapay sinir ağlarından söz edilebilir.

Geleneksel yazılımlarla zeki yazılımlar kıyaslandığında başlıca iki fark görülür. Birincisi, geleneksel yazılımlar programlanan koşullara göre hareket edip, bu sınırlar dışına çıkamazken zeki yazılımlar ise bilmedikleri durumlara da bir şekilde cevap üretebilirler. İkinci olarak ise, zeki yazılımların öğrenme yeteneğinin olması ve böylece sürekli olarak kullanıldığı problemin çözümü için kendini iyileştirmesidir.

Yapay zeka geliştirmede temelde iki farklı yaklaşım olduğu söylenebilir. İlk yaklaşım, bilinen zeki varlıkların zekasını taklit etmekle yapay zekanın gerçekleştirilebileceği şeklindedir. İkincisi ise makineleri zekileştirmek için makinelere has teknikler geliştirilmesi gerektiğini söyleyen yaklaşımdır.

Yapay sinir ağları, insan beyninin çalışma mantığının bilgisayar ortamına taşınmasını amaçlar. Biyolojik sinir ağlarının yetenekleri bu sayede, geleneksel programlama yöntemleriyle çözümü zor olan problemler için kullanılabilir.

Yapay sinir ağlarının kullanıldığı problemler genel olarak insanın çözmekte zorlanmadığı ancak bilgisayarlarla çözümü çok zor olan problemlerdir. Örnek verilecek olunursa; yüz tanıma, karakter tanıma, tahminleme, görüntü işleme gibi problemler sayılabilir.

Yapay sinir ağları tekniği uygulandığı problemlerdeki başarısı nedeniyle yapay zeka konusu altında önemli bir yere sahiptir. Ancak bu başarılı teknik, yazılım geliştirme sürecinde problem çözmede çok kullanılamamıştır. Bunun başlıca nedenleri ise, yapay sinir ağlarının öğrenilmesinin ve kullanılmasının bilgisayar bilimindeki diğer problem çözme tekniklerine kıyasla zor olması ve yazılım geliştirme sürecine doğrudan entegrasyonu sağlayacak araçların mevcut olmaması veya yetersiz olmasıdır.

Bu çalışmada, yapay sinir ağlarının yazılımlara eklenebilmesi için kolay kullanılabilir bir yazılım kütüphanesi, nesne yönelimli programlama yaklaşımıyla geliştirilmiştir. Ayrıca, geliştirilen kütüphaneyi kullanarak yazılımlara bu tekniğin eklenmesini sağlayan görsel geliştirme aracı da gerçekleştirilmiştir.

Bölüm 2'de yapay sinir ağları hakkında genel bilgi verilmiştir. Bölüm 3'de belli başlı sinir ağı modelleri ve öğrenme algoritmaları anlatılmıştır. Bölüm 4'de bu çalışmayla ilgili benzer çalışmalar değerlendirilmiştir. Geliştirilen yazılım hakkında detaylı bilgiler Bölüm 5'de verilmiştir.

# 2. YAPAY SİNİR AĞLARI

## 2.1 Yapay Sinir Ağları Nedir?

Yapay sinir ağları (YSA), örneklerden öğrenme, bilgiyi genelleştirebilme, paralel çalışma gibi özellikleri sayesinde veri işleme aracı olarak günümüzde bir çok alanda kullanılmaktadır. Yapay sinir ağları bir bilgi işleme teknolojisidir.

Yapay sinir ağları günümüzde kullanılan bilgi işleme yaklaşımlarından tamamen farklı bir yaklaşımdır. Bu hesaplama tekniğinin, desen tanımlama, robot kontrolü ve bilgi kazancı gibi çeşitli adresleme problemlerinde daha etkin olduğu kanıtlanmıştır (Kohonen, 1988).

Karmaşık bir yapıya sahip olan yapay sinir ağlarını analiz etmek ve içinde neler olduğunu anlamak oldukça zordur. Buna benzer bir başka sorun ise yapay sinir ağları konusunu öğrenme aşamasında yaşanılan zorluklardır. Çok boyutlu vektör uzayları arasındaki ilişkilerin, öğrenen tarafından akılda canlandırılması güçtür. Bu yüzden yapay sinir ağlarının görselleştirilmesi, konunun öğrenilmesinde yararlı olmaktadır**.**

Yapay sinir ağları, insan beyninin çalışma mantığının bilgisayar ortamına aktarılmasını ve böylece geleneksel programlama yöntemleriyle gerçekleştirimi oldukça zor veya mümkün olmayan biyolojik sinir ağlarının yeteneklerinin kullanılmasını sağlar (Fausett, 1994).

İnsan beyninin çalışma prensipleri ışığında ortaya çıkan yapay sinir ağları gücünü insan beyninin iki temel özelliği olan; paralel dağıtık yapısından ve öğrenme sonucunda genelleyebilme yeteneğinden alır.

Teknik olarak, bir yapay sinir ağının en temel görevi, kendisine gösterilen bir girdi setine karşılık gelebilecek bir çıktı seti belirlemektir.

Bunu yapabilmesi için ağ, ilgili olayın örnekleri ile eğitilerek (öğrenme) genelleme yapabilecek yeteneğe kavuşturulur. Bu genelleme ile benzer olaylara karşılık gelen çıktı setleri belirlenir. Ağı oluşturan temel elemanların bilgileri işleme yetenekleri ve birbirleri ile bağlantılarının şekilleri değişik modelleri oluşturmaktadır (Haykin, 1999).

## 2.2 Biyolojik Sinir Ağları

İnsan beyni yaklaşık olarak 10 milyar sinir hücresine (nöron) sahiptir. Ortalama olarak her sinir hücresi 10000 bağlantıyla diğer sinir hücrelerine bağlıdır. Beynin bu sinir hücrelerinden oluşan ağ yapısı muazzam bir bilgi işleme sistemi oluşturur. Beynin bu özelliği, tek işlemciye sahip ve aynı anda sadece tek işlem yapabilen geleneksel bilgisayarlar ile belirgin bir farklılık oluşturur. Buna karşın temel işlemleri geçekleştirme sürelerine bakacak olursak; sinir hücreleri en fazla 100 Hz hızında çalışırken, geleneksel bir Merkezi İşlem Ünitesi (CPU) saniyede yüz milyonlarca işlem yapabilme kapasitesine sahiptir. Donanımsal olarak düşük bir hıza sahip olarak görünse de, beynin çok önemli yetenekleri vardır:

- Bölgesel hasarlar sonrası çalışmaya devam edebilir. Buna karşın bir çok yazılım, rastgele bazı kısımları çıkarıldığında çoğunlukla işlevselliğini kaybeder.
- Tecrübeyle öğrenebilir, kendini yeniden şekillendirebilir.
- Çok sayıda hesaplamayı paralel ve oldukça etkin bir şekilde gerçekleştirir. Örneğin, karmaşık bir görsel algılamayı 100 mili saniyenin altında yapabilir.
- Hatalı verilere rağmen doğru sonuç üretebilir.

Beyin homojen yapıda değildir. En büyük anatomik ölçekte beyin 4 bölüme ayrılır; korteks, orta beyin, beyinsapı ve beyincik. Bu her bölüm de kendi içinde, gerçekleştirdiği işleme veya sahip olduğu sinir ağı yapısına göre farklı bölgelere ve alanlara ayrılabilir.

İnsan beynindeki bir sinir hücresi kabaca üç kısımdan (Şekil 2.1) oluşur :

- Dendrit
- Hücre gövdesi (Soma)
- Akson


Şekil 2.1 Biyolojik sinir hücresi

Sinir hücreleri birbirlerine sinaps denilen oluşumlarla bağlıdır. Sinaps; bir hücrenin akson sonuyla diğer hücrenin dendrit kısımlarının birleştikleri noktadır. Bir sinir hücresi dendritleri aracılığıyla kendisine bağlı diğer hücrelerden sinyalleri alır. Bu sinyaller veya bir diğer deyişle girdiler, hücre gövdesinde birleştirilir. Eğer bu girdiler toplamı belli bir kritik değeri aşarsa hücre gövdesinden akson boyunca ilerleyen ve nihayetinde bağlı olduğu diğer sinir hücrelerine ulaşan bir elektrik atımı oluşur. Bir hücreden gelen sinyalin önemi, yani sinyallerin birleştirilmesi işleminde ne derecede dikkate alınacağı sinapsta bulunan sinyal iletim

maddesinin miktarıyla doğru orantılı olarak saptanır (Russel-Norvig, 1995).

İnsan beyninin nasıl öğrendiği konusunda henüz bir çok nokta bilinmese de temel olarak; sinir hücreleri arasındaki bağlantıları değiştirerek ve hücreler arasında bağlantılar oluşturup veya mevcut bağlantıları kaldırarak öğrenmeyi sağladığı söylenebilir.

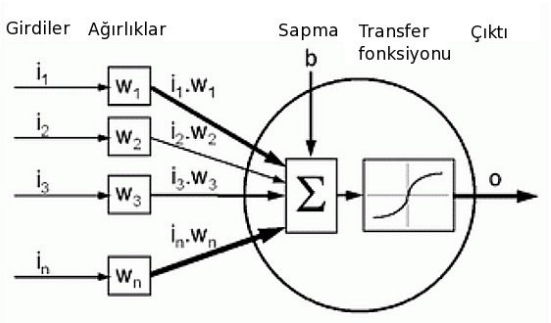
Yapay sinir ağları modellerinde, sinir sisteminin şu özellikleri kullanılmıştır:

- Paralel ve dağıtık bilgi işleme
- Temel elemanlar arasında yüksek dereceli bağlantı
- Bağlantıların tecrübeye bağlı olarak değişmesi
- Öğrenmenin genellikle eğiticisiz olması
- Sadece yerel bilgiye dayalı öğrenme

## 2.3 Yapay Sinir Hücresi Temel Elemanları

Bu bilgiler ışığında bir yapay sinir hücresinin (Şekil 2.2) biyolojik anlamdaki karşılığının kopyası olduğu söylenebilir. Bir yapay sinir hücresinin 5 temel elemanı vardır:

- Girdiler
- Ağırlıklar
- Toplam fonksiyonu
- Aktivasyon ya da transfer fonksiyonu
- Hücre çıktısı




Şekil 2.2 Yapay sinir hücresi

**Girdiler**: Dış sistemden gelen veriler olabileceği gibi başka yapay sinir ağı hücrelerinini çıktı değerleri de olabilir.

**Ağırlıklar** : Bir sinir hücresine genellikle bir çok girdi verisi ulaşır. Her girdinin kendine bağlı bir ağırlık değeri vardır. Ağırlık, girdi değerinin toplam fonksiyonu üzerindeki etkisini belirler. Bu sayede bazı girdi değerleri diğerlerinden daha önemli olabilir ve böylece sinir hücresi üzerinde daha fazla etkili olabilir. Ağın topolojilik yapısına veya öğrenme algoritmasına göre ağırlıkların değiştirilmesi yöntemleri farklılık gösterir.

**Toplam Fonksiyonu**: Bir yapay sinir hücresinde ilk işlem tüm girdilerin ağırlıklı toplamlarını bulmaktır. Matematiksel olarak bu girdiler ve karşılık gelen ağırlıklar $(i_1, i_2, \ldots, i_n)$ ve $(w_1, w_2, \ldots, w_n)$ şeklinde gösterilebilen vektörlerdir. Toplam girdi sinyali genellikle bu iki vektörün iç çarpımı $\sum_{j=1}^{n} i_j \cdot w_j$ olarak bulunur ve sonuç bir vektör değil tek bir sayıdır.

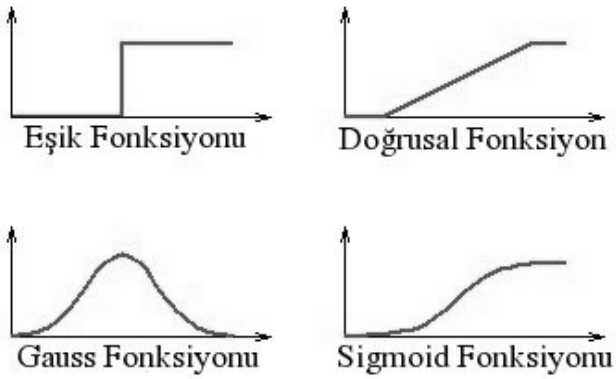
Geometrik olarak, iki vektörün iç çarpımı aralarındaki benzerliğin bir ölçütü olarak düşünülebilir. Eğer vektörler aynı yöndeyseler iç çarpım en fazla, eğer ters yöndeyseler iç çarpım en düşük değerini alır.

Toplam fonksiyonu, ağırlıklı toplamdan daha karmaşık olabilir. Ağırlıklar ve girdiler farklı şekillerde birleştirilebilir. Örneğin; en düşük, en büyük fonksiyonları kullanılabileceği gibi çeşitli normalleştirme fonksiyonları da kullanılabilir.

**Aktivasyon (Transfer) Fonksiyonu:** Toplam fonksiyonunun sonucu aktivasyon fonksiyonuna aktarılır ve hücrenin çıktı değeri üretilir. Aktivasyon fonksiyonları ağın kullanıldığı probleme, ağın topolojisine veya kullanılan öğrenme algoritmasına göre değişiklik gösterebilir. Örneğin, geriye yayılma (backpropagation) öğrenme algoritması uygulanacak ağdaki yapay sinir hücrelerinin aktivasyon fonksiyonlarının, türevleri alınabilir cinsten olması gerekir.

Aktivasyon fonksiyonları ağın doğrusal olmaması için gereklidir. Doğrusal olmayan aktivasyon fonksiyonlarının kullanılması ile yapay sinir ağı, sadece algılayıcıların (perceptron) kullanıldığı ağlardan daha kuvvetli olur. Çok katmanlı ağları güçlü yapan şey doğrusal olmamalarıdır. Her türlü doğrusal olmayan fonksiyon bu anlamda sinir ağını etkin kılmakta kullanılabilir. Sigmoid, hiperbolik tanjant ve Gauss fonksiyonları çok kullanılan doğrusal olmayan fonksiyonlardır.

Hem negatif hem pozitif değerler üreten doğrusal olmayan fonksiyonlar sadece pozitif veya negatif değer üreten fonksiyonlara kıyasla ağın daha hızlı öğrenmesini sağlayabilirler. Bazı aktivasyon fonksiyonları Şekil 2.3'de gösterilmiştir.




Şekil 2.3 Örnek aktivasyon fonksiyonları

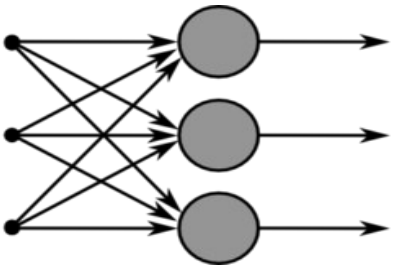
**Hücre çıktısı :** Aktivasyon fonksiyonun ürettiği sonuç hücre çıktısı olarak kabul edilir. Bu çıktı hücrenin ağdaki konumuna göre, bağlı olduğu diğer hücrelere girdi olarak iletilir veya ağın ürettiği sonuç olarak değerlendirilir.

Bir yapay sinir hücresinin çalışmasını özetleyecek olursak; Şekil 2.2'deki yapay sinir hücresine gelen girdiler $i_n$ ile gösterilmiştir. Bu girdiler $w_n$ ile gösterilen bağlantı ağırlılıklarıyla çarpılır. En basit haliyle bu çarpımlar toplanarak, aktivasyon fonksiyonuna aktarılır ve çıktı üretilmesi sağlanır. Toplam değerinde bazı durumlarda belirli bir sapma uygulanabilir. Bu basit sinir hücresi yapısı farklı toplam ve aktivasyon fonksiyonları kullanılarak çeşitlendirilebilinir.

## 2.4 Belli Başlı YSA Mimarileri

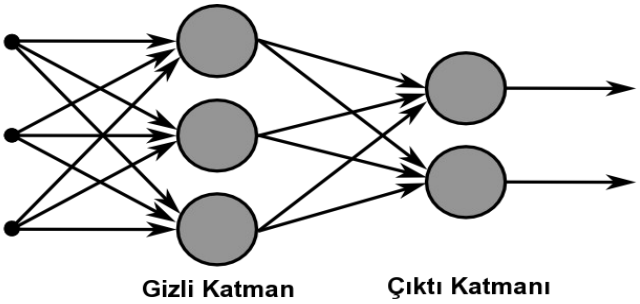
Yapay sinir ağlarının nasıl yapılandığı, öğrenme algoritmalarıyla çok yakından ilgilidir. Mimarilerine (yapılarına) bakarak genel olarak 3 temel yapay sinir ağı sınıfının olduğu söylenebilir.

**1. Tek katmanlı İleri Beslemeli Ağlar :** Tek bir girdi katmanı vardır ve bu da doğrudan çıktı katmanına bağlıdır. Bu ağ yapısında veri iletimi sadece ileri yöndedir, yani girdi katmanından çıktı katmanına doğru. Girdi katmanında hiçbir hesaplama yapılmadığı için sadece çıktı katmanı hesaba katılarak bu türdeki ağlara tek katmanlı ağ (Şekil 2.4) denir.



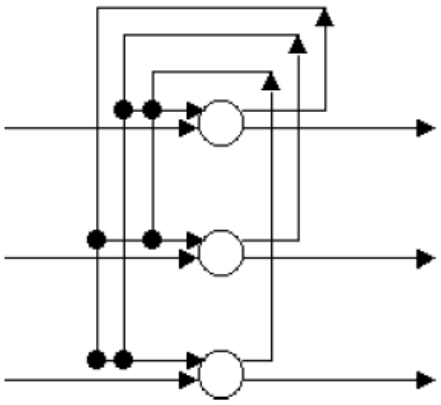
Şekil 2.4 Tek katmanlı ağ

**2. Çok Katmanlı İleri Beslemeli Ağlar :** Bir önceki modelden farklı olarak bu tür ağlarda bir veya birden fazla gizli katman bulunur. Gizli katman eklemekle ağ yüksek dereceli istatistik çıkarabilir hale gelir. Girdi verisi büyük olduğu zamanlarda, özellikle yüksek dereceli istatistik çıkarmak önem kazanır. Şekil 2.5'de *tümüyle bağlı* olan, yani her katmandaki hücrelerin bir sonraki katmandaki tüm hücrelerle bağlı olduğu türden, bir çok katmanlı ağ gösterilmiştir. Eğer bazı sinir hücresi bağlantıları ağda eksikse, o sinir ağına *kısmen bağlı ağ* denilir (Haykin, 1999).




Şekil 2.5 Çok katmanlı ileri beslemeli ağ

**3. Geri Dönüşümlü (Recurrent) Ağlar :** Bu tür ağları ileri beslemeli ağlardan ayıran özellik en az bir tane geri besleme döngüsüne sahip olmasıdır. Örneğin, bir geri dönüşümlü ağ tek bir katmandan oluşabilir ve bu katmandaki her sinir hücresinin çıktısı diğer hücrelere girdi olarak aktarılabilir. Bir sinir hücresi kendi çıktısını yine kendisine girdi olarak gönderebilir. Geri besleme döngülerinin olması ağın öğrenme yeteneklerini arttırır ve daha performanslı çalışmasını sağlar (Haykin, 1999). Şekil 2.6'da örnek bir geri dönüşümlü ağ gösterilmiştir.



Şekil 2.6 Geri dönüşümlü ağ

## 2.5 Yapay Sinir Ağları ve Geleneksel Hesaplama Yöntemleri

Yapay sinir ağları yöntemi, geleneksel hesaplama yöntemlerinden farklı şekilde veri inceleme ve desen tanıma imkanı getirmektedir. Ancak tüm hesaplama problemleri için uygun bir yöntem değildir. Geleneksel hesaplama yöntemleri iyi tanımlanabilen problemler için gayet iyi çalışır.

Yapay sinir ağları yöntemi, hem kendini programlayan hem de kendi kendine öğrenebilen bir araç sunar. Problemleri bir uzmanın veya programcının yardımına ihtiyaç duymadan çözmeyi sağlar. Veri kümesi

içindeki, kimsenin bilmediği desenleri tanımlayabilir. Avantajlarına rağmen bu yöntem her şey için çözüm değildir.

## 2.6 YSA'nın Avantajları ve Dezavantajları

Yapay sinir ağları çok iyi desen tanımlayabilir ve sınıflandırabilirler. Geleneksel yöntemlerin işe yaramadığı yerlerde başarılı olabilirler. Bu yöntem istisnaların ve normal olmayan girdi verilerinin olduğu sistemlerde uygulanabilir. Örneğin, radar veya sonar sistemleri. Yapay sinir ağlarının avantajları şu şekilde sıralanabilir:

- Veriden modele doğrudan geçişi sağlar.
- Tahmin edilecek değerler için sınırlandırmaya gerek duymaz.
- Geleneksel istatistik yöntemlerinden daha az çaba ister.

Dezavantaj olarak ilk akla gelebilecek durum; yapay sinir ağlarının, ürettiği sonuçları açıklayamamasıdır. Bu nedenle yapay sinir ağları *kara kutu* olarak anılır. Aslında bu durum bazı geleneksel hesaplama yöntemleri için de geçerlidir, örneğin regresyon analizi.

## 2.7 YSA'nın Kullanım Alanları

Yapay sinir ağları çok geniş uygulama alanlarında, verinin yoğun olduğu uygulamalarda başarıyla kullanılmıştır. Maddeler halinde sıralanacak olunursa:

- **Karakter tanıma :** El yazısı veya basılı metinlerin sayısal ortama aktarılması.

- **Süreç modelleme ve kontrol** : Bir tesis için sinir ağı modeli oluşturup bu modelle işletme için en uygun kontrol ayarlarının bulunması.
- **Portföy yönetimi :** Portföyün, karı en fazla riski ise en az olacak şekilde ayarlanması.
- **Hedef Tanımlama :** Askeri uygulamalarda, video veya kızılötesi verilerden düşman hedeflerinin tespit edilmesi.
- **Tıbbi Teşhis :** Girilen belirtilere göre ya da MRI veya röntgen verilerine göre doktorlara hastalık teşhisi koymasında yardımcı olunması.
- **Kredi Notlandırma :** Kişilerin veya kuruluşların mali durumlarına göre kredi notlarının tespiti.
- **Ses Tanıma :** Konuşulan kelimelerin metin haline getirilmesi.
- **Finansal Tahminleme :** Geçmiş verileri kullanarak gelecekteki finansal hareketleri tahmin etme.
- **Kalite Kontrol :** Üretim bandındaki hatalı ürünlerin tespiti.
- **Zeki Arama :** Kullanıcının geçmiş bilgilerine göre en uygun içeriği ve reklam afişlerini gösteren İnternet arama motoru.

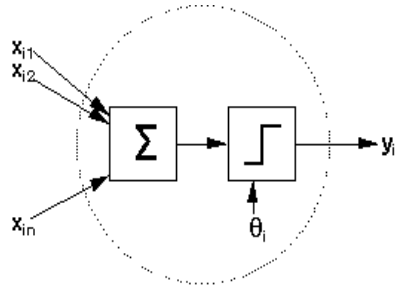
Yapay sinir ağlarının tüketiciye yönelik ürünlerde ilk uygulamaları 1990 yılının sonlarında görülmüştür. Bu dönemde bulanık mantık teknolojisinin ürünlerde yaygın olarak kullanılması nedeniyle bulanık mantığı ve sinir ağlarını birleştiren bir çok melez sistem ortaya çıkmıştır (Asakawa-Takagi, 1994).

# 3. YAPAY SİNİR AĞI MODELLERİ VE ÖĞRENME ALGORİTMALARI

## 3.1 McCulloch-Pitts Sinir Hücresi

McCulloch ve Pitts 1943 yılında, hesaplamaların ikili (binary) değerler alan basit sinir hücreleri ile yapılabileceğini öne sürmeleriyle birlikte yapay sinir ağları kavramı ortaya çıkmıştır.

McCulloch-Pitts hücre modeli iki tür girdi içerir, uyarıcı (excitory) girdi ve engelleyici (inhibitory) girdi. Hücre gelen girdileri toplar, eğer uyarıcı girdiler engelleyici girdilerden büyükse hücre ateşlenir yani çıktı oluşturur. Bu model mantık işlemlerinde çalışmakla beraber, bilginin nasıl saklandığını veya zeki davranışların nasıl öğrenildiğini göstermez (McCulloch-Pitts, 1943). McCulloch-Pitts hücresi basit olarak bir ikili eşik birimidir. Şekil 3.1'de şematik olarak gösterilmiştir.




Şekil 3.1 McCulloch-Pitts hücresi

## 3.2 Hebb Kuralı

Hebb, 1949'da bilginin sinir hücreleri arasındaki bağlantılarda saklandığı, öğrenmenin de bu bağlantıların ve çeşitli girdilerin uyarıcı ve engelleyici etkilerinin değiştirilmesi ile olduğu varsayımında bulunmuştur (Hebb, 1949).

Basit olarak Hebb kuralı şöyle özetlenebilir; eğer iki hücre eş zamanlı olarak aktif ise yani her ikisi de aynı anda pozitif veya negatif değere sahip ise aralarındaki bağ kuvvetlendirilmeli, eğer aynı anda zıt değerlere sahiplerse bu bağ zayıflatılmalıdır.

Matematiksel olarak ağırlık değişimi şu şekilde olur:

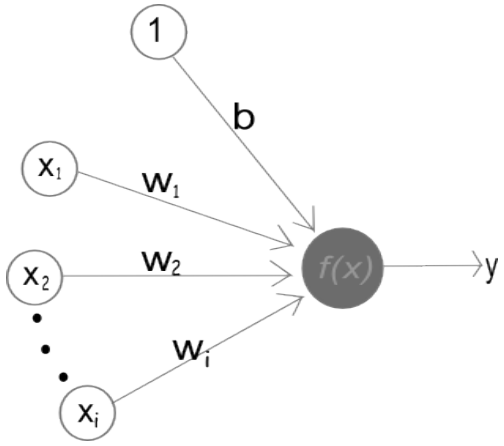
$w_{ij}(yeni) = w_{ij}(eski) + x_i x_j$ , $x_i ve x_j$ birbirlerine bağlı $i$ ve $j$ hücrelerinin çıktılarını gösterir.

## 3.3 Algılayıcı (Perceptron)

Frank Rosenblatt, sinir ağları konusuna, algılayıcıyı geliştirerek önemli bir katkı yapmıştır. Algılayıcılar ilk sinir ağları arasında en önemli ilerlemeyi gösterenlerdir.

Algılayıcı, sinir ağlarının kapsamlı incelenmesine olanak sağlayan basit bir model sağlamıştır. Rosenblatt ayrıca dijital bilgisayarlarda yapay sinir ağlarının simülasyonunu gerçekleştirmiştir (Rosenblatt, 1962). Marvin Minsky ve Seymour Papert algılayıcılar üzerinde yaptıkları derin matematiksel incelemeler sonucunda, algılayıcıların çok sınırlı alanlarda kullanılabileceğini ve algılayıcının çözemeyeceği çok fazla problem sınıfı olduğunu göstermişlerdir (Minsky-Papert, 1969). Algılayıcıların çözemediği problemlere örnek olarak XOR problemini gösterebiliriz.

Algılayıcı öğrenme algoritması Hebb kuralından çok daha etkilidir. Belli varsayımlar altında, yinelemeli öğrenme sürecinin doğru ağırlık değerlerine yakınsayacağı ispatlanabilir. Doğru ağırlık değerlerinden kasıt, her eğitim verisi için doğru çıktı değerini üretmesini sağlayan ağırlıklardır. Şekil 3.2'de sapma elemanı olan bir algılayıcı gösterilmiştir.




Şekil 3.2 Algılayıcı

Her algılayıcı için aktivasyon fonksiyonu olarak sabit eşik değerine sahip bir eşik fonksiyonu (Şekil 3.3) kullanılır .

$$f(y_{girdi}) = \begin{cases} 1 & y_{girdi} \geq 0 \\ 0 & y_{girdi} < 0 \end{cases}$$

Şekil 3.3 Eşik fonksiyonu

### 3.3.1 Algılayıcı Öğrenme Algoritması

*Adım 0* :Ağırlıkların ve sapmanın (bias) başlangıç değerlerini ata. Öğrenme oranını ( $\alpha$ ) belirle. ( $0 < \alpha \leq 1$ )

*Adım 1* :Durma koşulu sağlanana kadar 2-6 adımları arasında döngüde kal.

*Adım 2* : Her eğitim verisi çifti **s:t** için 3-5. adımları uygula.

*Adım 3* :Girdi değerlerini ata.

$$x_i = s_i$$

*Adım 4* :Çıktı değerini hesapla.

$$y_{girdi} = b + \sum x_i \cdot w_i$$

$$y = \begin{cases} 1 & y_{girdi} \geq 0 \\ 0 & y_{girdi} < 0 \end{cases}$$

*Adım 5* :Eğer hata oluşursa ağırlıkları ve sapma değerini güncelle.

*Eğer* $y \neq t$

$$w_i(yeni) = w_i(eski) + \alpha \cdot (t - y) \cdot x_i$$
$$b(yeni) = b(eski) + \alpha \cdot (t - y)$$

*değilse*

$$w_i(yeni) = w_i(eski)$$
$$b(yeni) = b(eski)$$

*Adım 6* :Durma koşulunu kontrol et; eğer 2.adımda hiç ağırlık değiştirme olmadıysa dur, değilse devam et.

### 3.3.2 Algılayıcı Öğrenme Kuralı Yakınsama Teoremi

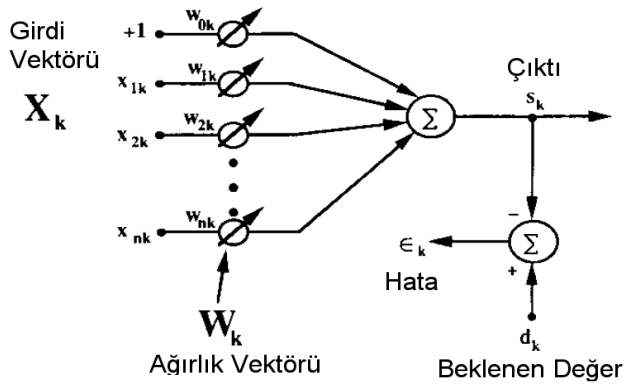
Eğer tüm eğitim verileri için beklenen sonuçları üreten bir $\boldsymbol{w}^*$ vektörü varsa, her $\boldsymbol{w}$ başlangıç ağırlık vektörü için algılayıcı öğrenme algoritması, tüm eğitim verileri için doğru sonucu üreten bir ağırlık vektörüne sonlu sayıda adımda yakınsar.

## 3.4 ADALINE ve Delta Kuralı

ADALINE, 1960 yılında Widrow ve Hoff tarafından ortaya konulmuştur. Açık adı '**ADA**ptive **LI**near **NE**uron' veya '**ADA**ptive

**LIN**ear **E**lement' olan bu sinir hücresi modeli yapısal olarak algılayıcıdan farklı değildir. Ancak, algılayıcı aktivasyon fonksiyonu olarak eşik fonksiyonu kullanırken ADALINE doğrusal fonksiyon kullanır. Algılayıcının çıktı değerleri 0 ve 1 ile sınırlıyken ADALINE her türlü değeri alabilir. Her iki model de sadece doğrusal olarak ayrılabilen problemleri çözebilirler.

Şekil 3.4'de ADALINE gösterilmiştir. ADALINE $k$ zamanında $\boldsymbol{X_k}=[x_0, x_{1k}, x_{1k}, \ldots, x_{mk}]^T$ girdi vektörü alırsa, girdi vektörü bileşenleri belli ağırlık katsayılarıyla çarpılırlar, ağırlık vektörü $\boldsymbol{W_k}=[w_{0k}, w_{1k}, w_{1k}, \ldots, w_{mk}]^T$ . Bu ağırlıklarla çarpılmış değerlerin toplamı hesaplanır, yani iç çarpım $s_k=\boldsymbol{X_k^T W_k}$ bulunur ve doğrusal bir çıktı hesaplanmış olur. Girdi vektörü $\boldsymbol{X_k}$ bileşenleri sürekli analog değerler olabileceği gibi ikili değerler de olabilir. Ağırlık değerleri sürekli değişkenlerdir ve eksi değerler yanında artı değerler de alabilirler (Widrow-Lehr, 1990).




Şekil 3.4 ADALINE

ADALINE, *delta kuralı* ya da *LMS(Least Mean Square)* denilen öğrenme algoritmasını kullanır. Delta kuralı; Adaline kuralı veya

Widrow-Hoff kuralı olarak da bilinir. Bu algoritma temel olarak ortalama hata kareleri toplamını en aza indirme mantığına dayanır.

Tek bir çıktı elemanı olan sistemde;

$\boldsymbol{x}$ : girdi elemanları vektörü

$y_{girdi}$ : çıktı elemanına gelen net girdi

$y_{girdi} = \sum x_i \cdot w_i$

$t$ : beklenen çıktı

Eğitim süresince çıktı elemanının ürettiği sonuç olarak $y_{girdi}$ kabul edilerek öğrenme işlemi gerçekleştirilir. Aktivasyon fonksiyonu ise doğrusal fonksiyondur.

Herhangi bir eğitim deseni için hata karesi :

$$E = \frac{1}{2}(t - y_{girdi})^2$$

*E,* tüm ağırlıkların bir fonksiyonudur. *E'*nin gradyanı *E*'nin tüm ağırlıklara göre kısmi türevlerinden oluşan bir vektördür. Gradyan, *E* için en hızlı artışın yönünü verir; bunun tersi yön ise hatadaki en hızlı düşüşün yönünü verir. Hata, $w_I$ ağırlığının $\frac{-(\partial E)}{(\partial w_I)}$ yönünde değiştirilmesiyle azaltılabilir.

$y_{girdi} = \sum x_i \cdot w_i$ ise , $\frac{(\partial E)}{(\partial w_I)} = -(t - y_{girdi}) \frac{(\partial y_{girdi})}{(\partial w_I)}$

$= -(t - y_{girdi}) x_I$

Böylece yerel hata en hızlı şekilde (verilen öğrenme oranına göre) ağırlıkların delta kuralına göre değiştirilmesiyle bulunur,

$$\Delta w_I = \alpha (t - y_{girdi}) x_I$$

Bu kural tek bir çıktı elemanı olan sistem için geçerlidir. Birden fazla çıktı elemanı olan sistemler için de bu kural genelleştirilebilir. Herhangi bir eğitim deseni için hata:

$$E=\frac{1}{2}\sum_{j=1}^{m}(t_j-y_{j_{girdi}})^2$$

ve yukarıdaki adımları takip ederek çok elemanlı sistemler için *I*'nci girdiden *J*'nci çıktı elemanına olan ağırlığı güncellemek için delta kuralı şu şekilde olur:

$$\Delta w_{IJ}=\alpha(t_J-y_{J_{girdi}})x_I$$

ADALINE öğrenme algoritmasının algılayıcı öğrenme algoritmasından farkı, girdi değerleri doğru sınıflandırılmış olsa da ağırlıklar güncellenir.

## 3.5 Çok katmanlı Algılayıcılar

Algılayıcılar yetenekleri sınırlı yapılardır. Örneğin XOR fonksiyonunu temsil edemezler. Bu sorunu inceleyecek olursak;

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 1 | 1 |
| 1 | -1 | 1 |
| 1 | 1 | -1 |

Şekil 3.5 XOR tablosu

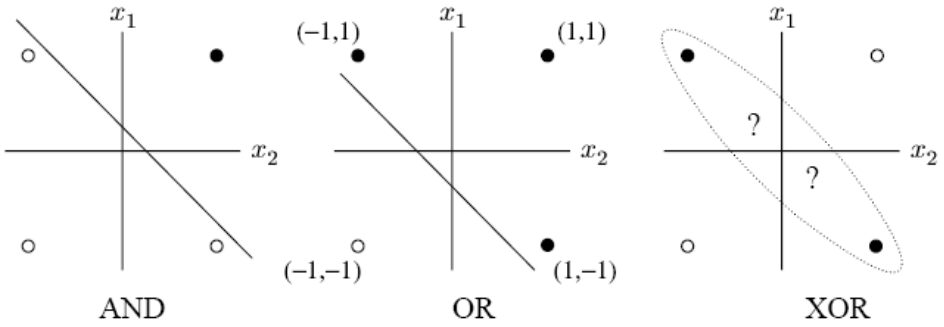
Şekil 3.5'deki iki girdi ve bir çıktı durumuna uygun bir algılayıcıda girdi şöyle olur;

$$y_{girdi} = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

Algılayıcının çıktısı, $y_{girdi}$ eksi değer olduğunda -1, artı değer olduğunda ise +1 olacaktır. Buna göre geometrik olarak düşünecek olursak;

$$w_1 \cdot x_1 + w_2 \cdot x_2 = -b$$

doğrusunun ayırdığı iki taraf algılayıcının çıktısına denktir. Ancak tek bir doğru ile XOR'daki iki sınıf tam olarak birbirinden ayrılamaz. Bu durum geometrik olarak Şekil 3.6'da gösterilmiştir.




Şekil 3.6 AND,OR ve XOR problemlerinin geometrik gösterimi

AND ve OR problemleri algılayıcıyla çözülebilir. Buna karşın XOR probleminde, tek bir doğruyla, temsil edilen iki sınıfı birbirinden ayırmak mümkün değildir.

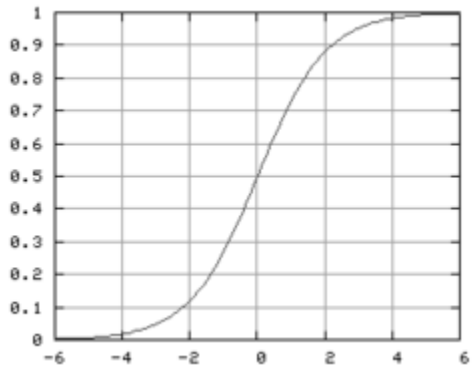
Genelleyecek olursak;

$y_{girdi} = b + \sum x_i \cdot w_i$ ve $f(y_{girdi}) = \begin{cases} 1 & y_{girdi} \geq 0 \\ -1 & y_{girdi} < 0 \end{cases}$ olmak üzere, $y_{girdi} > 0$ ve $y_{girdi} < 0$ bölgelerini birbirinden ayıran karar sınırını $b + \sum x_i \cdot w_i = 0$ denklemiyle gösterebiliriz. Girdi elemanlarının

sayısına bağlı olarak bu denklem bir doğru, bir düzlem veya bir hiper düzlemi temsil eder.

Eğer bu karar sınırı denklemiyle, eğitim girdilerinden beklenen çıktısı +1 olanlar ile -1 olanlar ayrı iki kısımda kalıyorlarsa bu türden problemlere *doğrusal olarak ayrılabilir* problemler denir. Tek katmanlı ağların sadece doğrusal olarak ayrılabilen problemleri öğrenebildiği Minsky ve Papert [1969] tarafından gösterilmiştir. Biraz daha genellenecek olunursa, doğrusal aktivasyon fonksiyonlarından oluşan çok katmanlı ağlar tek katmanlı ağlardan daha kuvvetli değildir. Çünkü doğrusal fonksiyonların birleşimi yine doğrusaldır (Fausett, 1994).

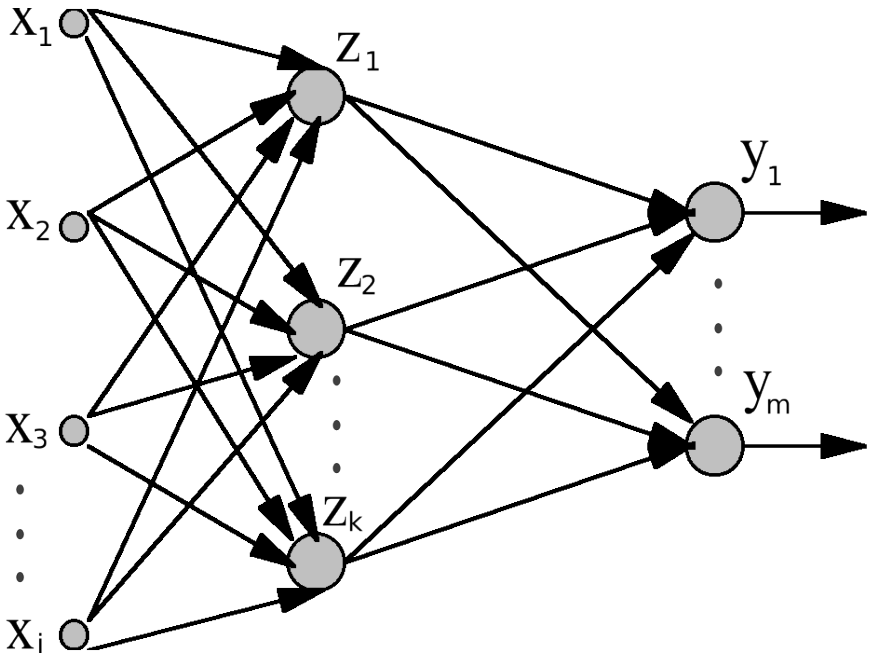
Çok katmanlı algılayıcılar, tek katmanlı algılayıcıların çözemediği doğrusal olmayan problemlerin çözümünde kullanılırlar. *Evrensel tahminleme teoremine (universal approximation theorem)* göre, tek bir gizli katmana sahip çok katmanlı algılayıcı ağı her türlü sürekli fonksiyonu temsil edebilir. Bu teoremin doğruluğu için ağdaki algılayıcılarda kullanılan aktivasyon fonksiyonlarının doğrusal olmaması dışında, sabit olmayan, sınırlı ve monoton artan fonksiyonlar olması gerekir. Örneğin, lojistik fonksiyonlar. Lojistik fonksiyonların genel görüntüsü aşağıdaki Şekil 3.7'de gösterilmiştir.




Şekil 3.7 Lojistik fonksiyonların genel görüntüsü

## 3.6 Geriye Yayılma Algoritması (Backpropagation)

Girdi katmanı ile çıktı katmanı arasına eklenecek gizli katmanlar sayesinde algılayıcının getirdiği kısıtlamaların bir çoğunun üstesinden gelinmesi sağlanır. İlk yapay sinir ağı araştırmacılarının karşılaştığı en büyük problem bu gizli katmanların ağırlıklarının nasıl güncelleneceği olmuştur. Bu amaçla çeşitli kurallar ortaya çıkmıştır. Bunlardan en ünlü olanı *hata geriye yayma,* altmışlı yılların sonlarında Bryson ve Ho tarafından ortaya konulmuştur (Abdi, 1994).




Şekil 3.8 Tek gizli katmana sahip çok katmanlı algılayıcı

Şekil 3.8'deki çok katmanlı ağ üzerinden bu öğrenme algoritmasının nasıl türetildiği incelenmiştir. *j* çıktı hücresi için *n*'nci eğitim verisi sonrası, $d_j(n)$ beklenen değer olmak üzere hata şu şekilde tanımlanır;

$$e_j(n)=d_j(n)-y_j(n)$$

Hata fonksiyonu olarak $E(n)=\frac{1}{2}\sum_{j\in C} e_j^2(n)$ kullanılmaktadır. $C$ çıktı katmanındaki tüm sinir hücrelerinin kümesidir. Burada delta kuralındakine benzer bir yaklaşımla $E(n)$ en düşük hale getirilmeye çalışılır.

Çıktı elemanlarına gelen girdiler toplamı :

$$v_j(n)=\sum_{i=0}^{m} w_{ji}(n)z_i(n)$$

$m$ toplam girdilerin sayısını (sapma hariç), $w_{j0}$ sapma elemanını gösterir, ve böylece $z_0=+1$ olur. Gizli katmandaki elemanlar $z$ ile gösterilmiştir.

Çıktı elemanlarının ürettikleri sonuç : $y_j(n)=f_j(v_j(n))$

Delta kuralındaki gibi , geriye yayılma algoritması da $\frac{\partial E(n)}{\partial w_{ji}(n)}$ kısmi türeviyle orantılı olarak bağlantı ağırlıklarında düzeltme yapar. Buna göre kısmi türev, zincir kuralına uygun olarak yazılırsa:

$$\frac{\partial E(n)}{\partial w_{ji}(n)}=\frac{\partial E(n)}{\partial e_j(n)}\cdot\frac{\partial e_j(n)}{\partial y_j(n)}\cdot\frac{\partial y_j(n)}{\partial v_j(n)}\cdot\frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

Tek tek türevleri alınırsa ;

$$\frac{\partial E(n)}{\partial e_j(n)}=e_j(n) \quad , \quad \frac{\partial e_j(n)}{\partial y_j(n)}=-1 \quad ,$$

$$\frac{\partial y_j(n)}{\partial v_j(n)}=f'_j(v_j(n)) \quad \text{ve} \quad \frac{\partial v_j(n)}{\partial w_{ji}(n)}=z_i(n)$$

böylece $\frac{\partial E(n)}{\partial w_{ji}(n)}=-e_j(n)f'_j(v_j(n))z_i(n)$ halini alır.

Ağırlık düzeltme miktarı $\Delta w_{ji}(n)$ delta kuralında şöyle tanımlanmıştı:

$$\Delta w_{ji}(n) = -\alpha \frac{\partial E(n)}{\partial w_{ji}(n)} \quad , \quad \alpha \text{ öğrenme oranı.}$$

Böylece geriye yayılma algoritması için ağırlık düzeltme miktarı:

$$\Delta w_{ji}(n) = \alpha \, \delta_j(n) \, z_i(n) \quad ,$$

yerel gradyan $\delta_j(n)$ (*hata bilgisi terimi* olarak da adlandırılır) ise şöyle tanımlanır;

$$\begin{aligned} \delta_j(n) &= -\frac{\partial E(n)}{\partial v_j(n)} \\ &= -\frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= e_j(n) \, f'_j(v_j(n)) \end{aligned}$$

### 3.6.1 Geriye Yayılma Algoritmasının Uygulanması

Tek gizli katmana sahip bir çok katmanlı ağ üzerinde geriye yayılma algoritmasının uygulanması:

*Adım 0* : Ağırlıkların ve sapmaların (bias) başlangıç değerlerini ata. Öğrenme oranını ( $\alpha$ ) belirle. ( $0 < \alpha \leq 1$ )

*Adım 1* : Durma koşulu sağlanana kadar 2-9 adımları arasında döngüde kal.

*Adım 2* :Her eğitim verisi için 3-8 adımları uygula.

İleri Yönde*:*

*Adım 3* :Girdi katmanından girdi verileri gizli katmana ilet.

*Adım 4* : Her gizli katman elemanı kendine gelen girdilerden ağırlıklı toplamı hesaplar.

$$z_{girdi-j} = b_j + \sum x_i w_{ij}$$

Aktivasyon fonksiyonunu uygulayarak çıktısını hesaplar ve bir üst katmana (çıktı katmanı) ilet.

$$z_j = f(z_{girdi-j})$$

*Adım 5* : Her çıktı katmanı elemanı ağırlıklı girdiler toplamını hesapla ,

$$y_{girdi-k} = b_k + \sum z_j w_{jk}$$

ve aktivasyon fonksiyonunu uygulayarak çıktısını bul.

$$y_k = f(y_{girdi-k})$$

Hatanın Geriye Yayılması :

*Adım 6* : Her çıktı elemanı kendisine denk gelen 'beklenen değeri' ( $t_k$ ) kullanarak geriye yayılacak hata bilgisini hesapla.

$$\delta_k = (t_k - y_k) f'(y_{girdi-k})$$

Bu hata bilgisi terimini kullanarak kendisine ait bağlantılar için düzeltme miktarlarını hesapla.

$$\Delta w_{jk} = \alpha \delta_k z_j$$

$$\Delta b_k = \alpha \delta_k$$

*Adım 7* : Her gizli katman elemanı bağlı olduğu çıktı elemanlarından gelen hata bilgilerini bağlantı

ağırlıklarını da kullanarak topla.

$$\delta_{girdi-j} = \sum \delta_k w_{jk}$$

Kendi hata bilgisi terimini hesapla.

$$\delta_j = \delta_{girdi-j} f'(z_{girdi-j})$$

Bağlantı güncelleme miktarlarını hesapla.

$$\Delta w_{ij} = \alpha \delta_j x_i$$

$$\Delta b_j = \alpha \delta_j$$

*Adım 8* :Tüm ağırlıklar hesaplanan düzeltme miktarlarıyla güncelle.

$$w_{yeni} = w_{eski} + \Delta w$$
$$b_{yeni} = b_{eski} + \Delta b$$

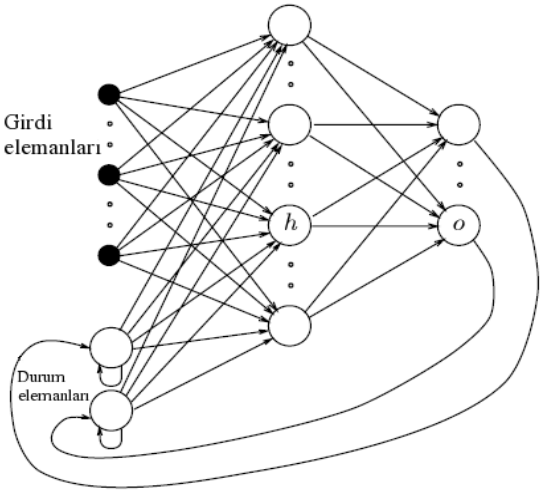
*Adım 9* :Durma koşulunu kontrol et.

## 3.7 Geri Dönüşümlü (Recurrent) Ağlar

### 3.7.1 Jordan Ağı

İlk geri dönüşümlü ağlardan biri Jordan ağıdır. Bu tür ağda, çıktı elemanlarının ürettikleri sonuçlar tekrar girdi katmanına *durum elemanları (state units)* denilen fazladan girdi elemanlarıyla geri bildirilir. Ne kadar girdi elemanı varsa o kadar durum elemanı olabilir. Çıktı ve durum elemanları arasındaki bağlantıların ağırlığı sabit olup +1 değerindedir; öğrenme sadece girdi katmanından gizli katmana olan bağlantılar ile gizli katmandan çıktı katmanına olan bağlantılar için gerçekleşir. Böylece çok katmanlı algılayıcılar için kullanılan tüm

öğrenme kuralları bu tür ağları eğitmekte de kullanılabilir. (Kröse-Smagt, 1996) Şekil 3.9'da örnek bir Jordan ağı gösterilmiştir.

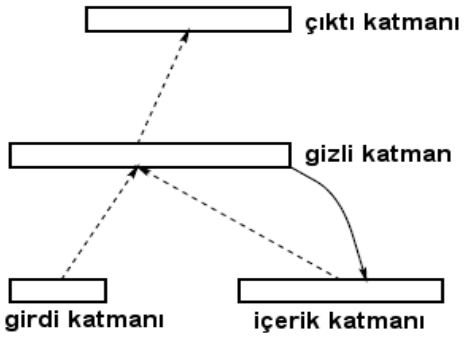



Şekil 3.9 Jordan ağı

### 3.7.2 Elman Ağı

İleri beslemeli ağlar kısa vadeli belleğe sahip değillerdir. Her hangi bir zamandaki çıktılar, o anki girdilerin ve ağın ağırlıklarının bir fonksiyonudur. Gerçek dünyadaki bir çok problem ise girdilerin ardışıklığını tanımlamayı gerektirir. Bu ihtiyacı gidermek için bir çok geri dönüşümlü ağ mimarisi ortaya konmuştur. Bunlar arasında günümüzde en çok kullanılan ise ağın kesikli zaman aralıklarında çalıştığını kabul eden Elman modelidir (Fahlman, 1990).

Bu tür ağ, gizli katman sinir hücresi elemanlarının çıktılarının girdi katmanına geri bildirilmesiyle oluşur. Bunu sağlamak için *içerik (context) elemanları* denilen yapılar kullanır. Bu içerik elemanlarının kendilerine bağlantıları yoktur. Jordan ağında olduğu gibi burada da gizli elemandan içerik elemanına olan bağlantının ağırlığı sabit ve +1 değerindedir. Şekil 3.10'da Elman ağının yapısı gösterilmiştir.



Şekil 3.10 Elman ağı

Elman ağında öğrenme şu şeklide olur :

*Adım 0* : İçerik elemanları 0 değerine eşitlenir.

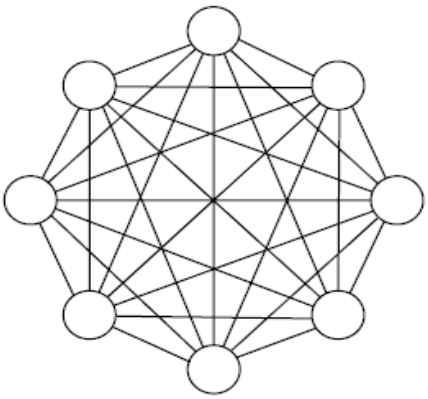
*Adım 1* : Girdi verisi ağa uygulanır.

*Adım 2* : Geriye yayılma öğrenme algoritması uygulanır.

*Adım 3* : 1. adımdan devam edilir.

### 3.7.3 Hopfield Ağı

Hopfield ağı birbirlerine bağlı *N* adet sinir hücresinden oluşur. Bunlar birbirlerine çıktılarını eş zamanlı olmadan ve diğer hücrelerden bağımsız olarak iletir. Çıktı olarak ikili değerler alır. Bu ağın ortaya çıktığı ilk zamanlarda çıktı değerleri olarak 1 ve 0 seçilmiş olmasına rağmen +1 ve -1 değerlerinin kullanılması daha avantajlıdır. Şekil 3.11'de Hopfield ağı gösterilmiştir.



Şekil 3.11 Hopfield ağı

Sistemin durumu çıktı değerleriyle ( $y_k$ ) gösterilir. Sinir hücresi $k$ için $t$+1 döngüsündeki girdiler toplamı ;

$$s_k(t+1)=\sum_{j\neq k} y_j(t)w_{jk}+\theta_k \quad , \quad \theta_k \text{ sapma elemanı.}$$

Aktivasyon fonksiyonu olarak basit bir eşik fonksiyonu kullanılır.

$$y_k(t+1)=\begin{cases} 1 & s_k(t+1)>B_k \\ -1 & s_k(t+1)<B_k \end{cases}$$

Hopfield'ın bu çalışması, gezgin satıcı problemi gibi bir çok günlük problemin çözümü için potansiyele sahip olması nedeniyle büyük ilgi görmüştür (Sondak, 1989). Hopfield ağı, geleneksel hesaplama yöntemlerinden çok daha hızlı bir şekilde kabul edilebilir yollar bulur (Hopfield, 1984).

## 3.8 Rekabete Dayalı veya Rekabetçi (Competitive) Öğrenme

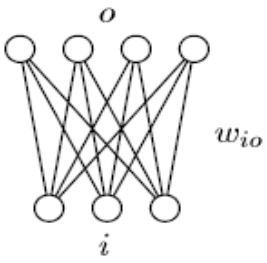
Şimdiye kadar bahsedilen yapay sinir ağı modellerinde, çözülmeye çalışılan probleme ait girdi ve beklenen değerlerden oluşan bir eğitim verisi kullanılmıştır. Bu tür eğitim verisinin olmadığı, sahip olunan

bilginin sadece girdi verilerinin olduğu durumlar da mevcuttur. Böyle durumlarda ilgili bilginin sahip olunan girdi örnekleri içinde bulunması gerekir. Bu türden problemlere örnek olarak şunlar gösterilebilir:

- Kümeleme
- Vektör nicemleme
- Boyut azaltma
- Özellik çıkarma

### 3.8.1 Kümeleme ve Rekabetçi Öğrenme

Rekabete dayalı öğrenme, bir dizi girdi desenini veya verisini kümelere ayıran bir öğrenme yöntemidir. Rekabetçi öğrenmeyi temel alan ağa sadece girdi vektörleri $\boldsymbol{x}$ sağlanır ve bir eğiticisiz öğrenme süreci takip edilir.




Şekil 3.12 Rekabete dayalı öğrenme ağı

Şekil 3.12'de rekabete dayalı öğrenme algoritması uygulanabilecek örnek bir ağ gösterilmiştir. Tüm *o* çıktı elemanları tüm *i* girdi elemanlarına , $w_{io}$ ağırlıklarıyla bağlanmıştır. Bir girdi deseni $\boldsymbol{x}$ , ağa uygulandığında, sadece tek bir çıktı elemanı (*kazanan eleman*) aktif olur. İyi eğitilmiş bir ağda bir kümeye ait tüm desenler için tek bir

kazanan eleman olmalıdır. Kazanan elemanı belirlemek için temel iki yöntem vardır:

1. İç çarpım **:**

Başlangıç olarak girdi vektörü $\boldsymbol{x}$ ve ağırlık vektörü $\boldsymbol{w}_o$ birim uzunluğa göre normalleştirilmiş oldukları kabul edilir. Her çıktı elemanı $o$ , çıktı değerini $y_o$ , girdiler ile ağırlık değerlerinin içi çarpımı olarak hesaplar.

$$y_o = \sum_i w_{io} x_i$$

Daha sonra çıktı değeri en büyük olan hücre $k$ seçilir. Çıktı değerleri şu şekilde sıfırlanır:

$$y_k = 1 \quad ve \quad y_{o \neq k} = 0$$

Bu, ağın rekabetçi olan yönüdür ve çıktı katmanı *kazanan hepsini alır* (*winner-take-all)* katmanı olarak adlandırılır. Kazanan hücre $k$ belirlendikten sonra ağırlıklar şu şekilde güncellenir:

$$\boldsymbol{w}_k(yeni) = \frac{\boldsymbol{w}_k(eski) + \alpha[\boldsymbol{x} - \boldsymbol{w}_k(eski)]}{\left\| \boldsymbol{w}_k(eski) + \alpha[\boldsymbol{x} - \boldsymbol{w}_k(eski)] \right\|}$$

İfadedeki bölen kısmı vektör uzunluğudur ve ağırlık değerlerinin normalleştirilmiş olmasını sağlar. Dikkat edileceği üzere sadece kazanan elemana olan bağlantılar güncellenir.

2. Öklid (Euclidean) Uzaklığı :

Bir önceki yöntemde girdilerin ve ağırlıkların normalleştirilmiş olması gerekliydi. Çünkü normalleşmemiş vektörlerin kullanılması yanlış sonuçlar çıkmasına neden olur. Normalleştirilmemiş girdi verileri için kazanan eleman $k$ 'yı bulmak için öklid uzaklığı kullanılır.

$$k : \left\| \boldsymbol{w}_k - \boldsymbol{x} \right\| \leq \left\| \boldsymbol{w}_o - \boldsymbol{x} \right\| \quad \forall o.$$

Ağırlık güncellemesi ise şu şekilde olur:

$$\boldsymbol{w}_k(yeni) = \boldsymbol{w}_k(eski) + \alpha[\boldsymbol{x} - \boldsymbol{w}_k(eski)]$$

Maliyet Fonksiyonu :

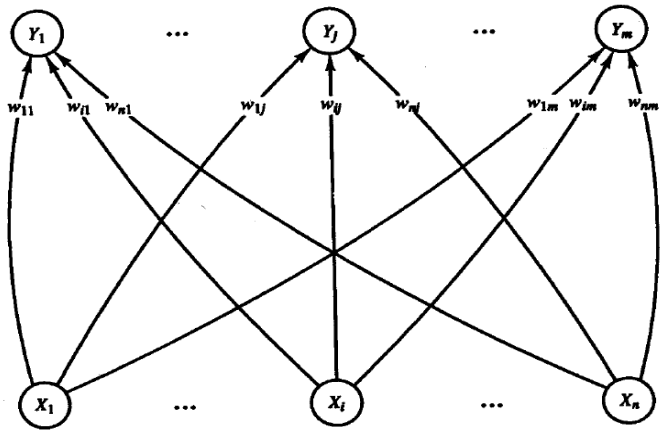
Genel olarak ağın kümelemeyi ne kadar iyi yaptığını ölçmek için hata kareleri kullanılır.

$$E = \sum_p \left\| \boldsymbol{w}_k - \boldsymbol{x}^p \right\|^2$$

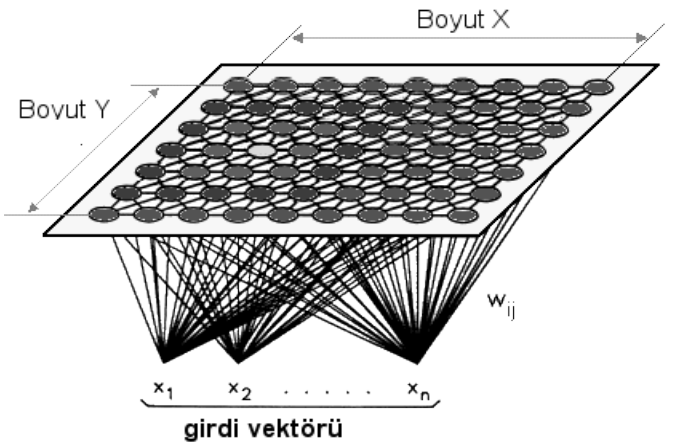
Bu formülde $k$ , $\boldsymbol{x}^p$ girdi vektörü ağa uygulandığında kazanan elemanı gösterir. Rekabetçi öğrenme bu maliyet veya hata fonksiyonunu en aza indirmeye çalışır.

### 3.8.2 Kohonen SOM

Self-Organizing Map (SOM) Teuvo Kohonen tarafından 1982 yılında ortaya konuldu. Kohonen'in geliştirdiği SOM, eğiticisiz öğrenme özelliğine sahiptir. Ağ yapısı olarak tek katmanlı ileri beslemeli ağ yapısına sahiptir. Çıktı katmanı hücreleri genellikle 2 veya 3 boyutlu ızgara görüntüsünde dizilirler. Genellikle girdi verisinin boyutu çıktı katmanının boyutundan çok daha fazladır. Girdi uzayının topolojik özelliklerini koruyarak eğitim verilerinin düşük boyutlu temsil edilmesini sağlarlar. Şekil 3.13'de Som ağının ileri beslemeli ağ olarak gösterimi, Şekil 3.14'de çıktı katmanının 2 boyutlu ızgara şeklindeki görüntüsü verilmiştir.




Şekil 3.13 SOM ağının yapısı




Şekil 3.14 Çıktı katmanın ızgara şeklindeki görüntüsü

### 3.8.3 SOM Öğrenme Algoritması :

SOM'da rekabetçi öğrenme şeklinde bir eğitim uygulanır. Uzaklık fonksiyonu olarak genellikle öklid uzaklığı kullanılır. Ancak burada tek bir kazanandan çok, kazanan eleman merkezde olacak şekilde belli bir çap içerisinde kalan elemanlar kazanan olur.

Adım 0 : Ağırlık değerlerinin ata. Öğrenme oranının ve komşuluk yarıçapının başlangıç değerlerini ata.

Adım 1 : Durma koşulu sağlanana kadar

Adım 2 : Her girdi vektörü $\boldsymbol{x}$ için

Adım 3 :

Her $j$ için hesapla :

$$D(j)=\sum_i (w_{ij}-x_i)^2$$

Adım 4 : $D(j)$ en düşük yapan $J$ değerini bul.

Adım 5 : $J$'nin komşuluk yarı çapı içindeki tüm elemanlar için ağırlıkları güncelle :

$$w_{ij}(yeni)=w\,ij\,(eski)+\alpha[x_i-w_{ij}(eski)]$$

Adım 6 : Öğrenme oranını güncelle.

Adım 7 : Çıktı katmanı için kullanılan topolojik komşuluk çapını belirli aralıklarla azalt.

Adım 8 : Durma koşulunu kontrol et.

Öğrenme oranı $\alpha$ zamana bağlı yavaş azalan bir fonksiyon olarak düşünülür. Kohonen bu fonksiyon için doğrusal azalan bir fonksiyonun pratik hesaplamada yeterli olacağını ifade eder. Ayrıca $\alpha$ başlangıç değeri olarak 0.1'e yakın olmalı ve eğitim sırasında azalmalı ancak her zaman 0.01 değerinin üstünde olmalıdır (Kohonen, 1982).

Ağırlık değerleri güncellenecek çıktı elemanlarının belirlenmesinde kullanılan komşuluk çapı da öğrenme süreci boyunca azalır.

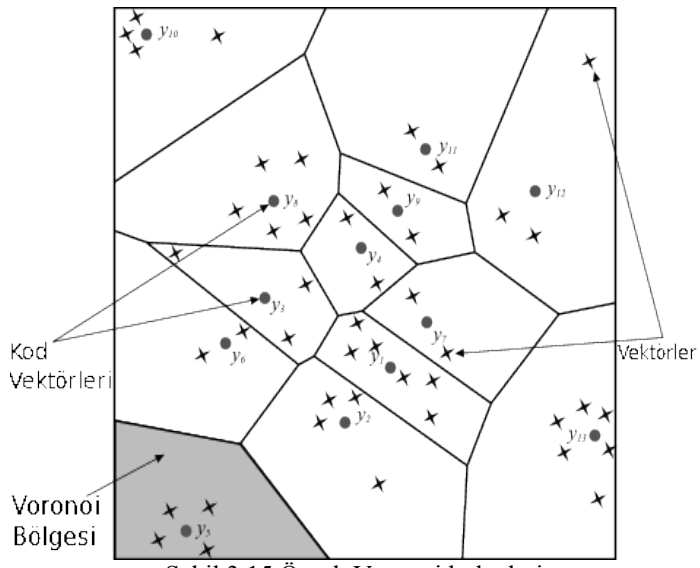
### 3.8.4 Vektör Nicemleme (Vector Quantization)

Vektör nicemleme, önemli bilgileri kaybetmeden sahip olunan verilerin miktarını sıkıştırmak için girdi vektörlerinin kümelere ayrılması işidir. Bir vektör nicemleme şeması girdi uzayını ayrık alt uzaylara ayırır ve her girdi vektörü içine düştüğü alt uzay ismiyle gösterilir. Eğer veri sıkıştırması için kullanılıyorsa, bu alt uzaylar daha az yer kaplayacağı için bu şekilde veri sıkıştırılmış olur.

Bir vektör nicemleyici, $\mathbb{R}^k$ uzayındaki *k* boyutlu vektörleri sonlu sayıdaki $Y=\{\boldsymbol{y_i}: i=1,2,\ldots,N\}$ vektörlerine eşler. Her $\boldsymbol{y_i}$ vektörüne *kod vektörü* denir, tüm bu kod vektörlerine *kod kitabı* denir. Her $\boldsymbol{y_i}$ vektörü için bir tane en yakın komşular bölgesi vardır. Bu bölgelere *Voronoi* bölgeleri denir ve şu şekilde ifade edilir:

$$V_i=\{x\in\mathbb{R}^k: \|\boldsymbol{x}-\boldsymbol{y_i}\|\leq\|\boldsymbol{x}-\boldsymbol{y_j}\|, tüm\ j\neq i\}$$

Şekil 3.15'de Voronoi bölgelerine ayrılmış 2 boyutlu bir girdi uzayı gösterilmiştir.



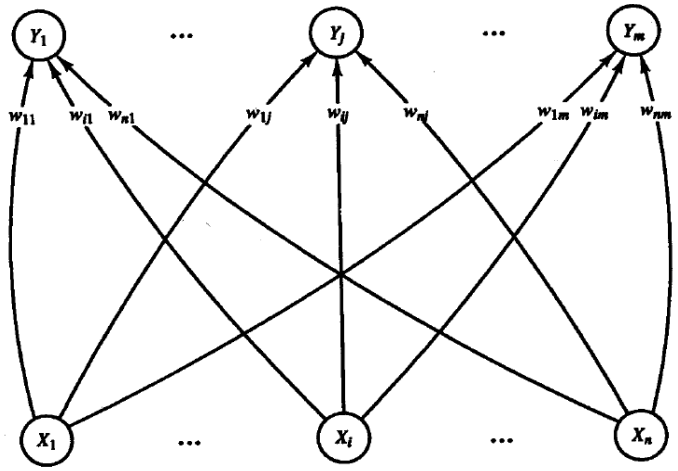

Şekil 3.15 Örnek Voronoi bölgeleri

### 3.8.5 Learning Vector Quantization (LVQ)

LVQ, her çıktı elemanının bir sınıf veya kategoriyi gösterdiği bir desen sınıflandırma yöntemidir. Bir çıktı elemanın ağırlık vektörü genellikle *referans* veya *kod kitabı* olarak tanımlanır. Eğitim sonrası LVQ ağı, bir girdi vektörünü, hangi çıktı elemanının ağırlık vektörüne en yakın ise o sınıfa atar.

Sınıflandırma doğruluğu en az diğer sinir ağı algoritmaları kadarken, hesaplamalar açısından çok daha basittir. Öğrenme ve sınıflandırma hızı diğer algoritmalardan daha yüksek olabilir. Bunların yanında kullanımı çok daha kolaydır (Kohonen, 1990).

Bir LVQ ağının yapısı Kohonen SOM ağının aynısıdır, sadece burada SOM'da olduğu gibi çıktı katmanının topolojik yapısı düşünülmez. Şekil 3.16'da bir LVQ ağı gösterilmiştir.




Şekil 3.16 LVQ ağının yapısı

### 3.8.6 LVQ Öğrenme Algoritması

Eğiticili öğrenme algoritması nedeniyle girdi vektörleri ile birlikte ait oldukları sınıf veya kategori bilgisi de sağlanmalıdır.

$\boldsymbol{x}$ *eğitim vektörü* $(x_1, \ldots x_i, \ldots, x_n)$

$T$ *eğitim vektörü için doğru kategori veya sınıf*

$\boldsymbol{w}_j$ *j çıktı elemanı için ağırlık vektörü* $(w_{1j}, \ldots, w_{ij}, \ldots, w_{nj})$

$C_j$ *j çıktı elemanının gösterdiği kategori veya sınıf*

$\|\boldsymbol{x}-\boldsymbol{w}_j\|$ *girdi vektörü ile j çıktı elemanının ağırlık vektörü arasındaki Öklid uzaklığı*

Adım 0 : Ağırlıkların başlangıç değerlerini ata.

Adım 1 : Durma koşulu sağlanana kadar döngüde kal.

Adım 2: Her girdi vektörü $\boldsymbol{x}$ için

Adım 3: $\|\boldsymbol{x}-\boldsymbol{w}_J\|$ değerini en az yapan $J$'yi bul.

Adım 4: $\boldsymbol{w}_J$ 'yi güncelle:

$$\text{Eğer } T = C_j \text{ ise}$$
$$\boldsymbol{w}_j(yeni) = \boldsymbol{w}_j(eski) + \alpha[\boldsymbol{x} - \boldsymbol{w}_j(eski)]$$
$$\text{Eğer } T \neq C_j \text{ ise}$$
$$\boldsymbol{w}_j(yeni) = \boldsymbol{w}_j(eski) - \alpha[\boldsymbol{x} - \boldsymbol{w}_j(eski)]$$

Adım 5: Öğrenme oranını $(\alpha)$ azalt.

Adım 6: Durma koşulunu kontrol et.

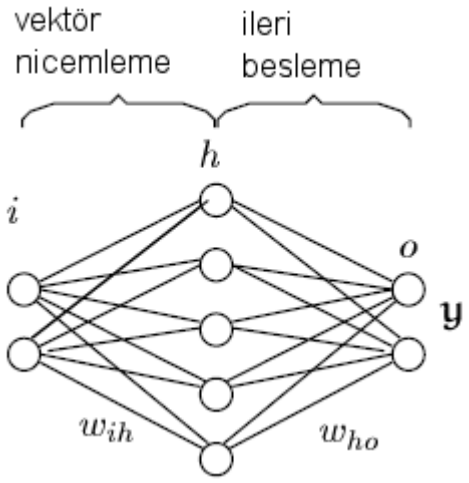
Öğrenme, ya belirli bir tekrarlama sayısına ulaşınca ya da öğrenme oranı belli bir değere gelince durabilir.

Ağırlık değerlerinin başlangıç değerlerinin atanmasında, girdi vektörlerinden ilk *m* tanesi, yani çıktı elemanı sayısı kadar alınıp ağırlıklar bu değerlere atanabilir, bu durumda geri kalan girdi vektörleri de eğitimde kullanılır. Başka bir yol olarak ağırlıklara rastgele sayı değerleri verilebilir.

LVQ öğrenme algoritması daha sonraları geliştirilmiştir ve LVQ2, LVQ2.1 ve LVQ3 olmak üzere farklı algoritmalar ortaya çıkmıştır. İlk LVQ öğrenme algoritmasında, sadece girdi vektörüne en yakın referans vektörü güncellenirken geliştirilen sonraki algoritmalarda kazanan ve ona en yakın olan iki çıktı elemanının referans vektörleri güncellenir.

### 3.8.7 Counterpropagation

Bir çok uygulamada vektör nicemleme yapan ağlar, fonksiyon tahminlemesi yapan başka tür bir ağ ile birleştirilir. Şekil 3.17'de bu ağ yapısı gösterilmiştir.




Şekil 3.17 Counterpropagation

Bu tür ağlar $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ şeklindeki fonksiyonları tahminleyebilirler.

Bu yolla fonksiyon tahminleme bir *arama tablosu* şeklinde gerçekleşir. Girdi vektörü $\boldsymbol{x}$ tabloda $k$ ile gösterilir ve $\forall \neq k : \left\| \boldsymbol{x} - \boldsymbol{w}_k \right\| \leqslant \left\| \boldsymbol{x} - \boldsymbol{w}_o \right\|$ , bu tablodaki fonksiyon değeri $[w_{1\mathrm{k}}, w_{2\mathrm{k}}, \ldots, w_{mk}]$ ise $f(x)$ için bir tahmin olarak ele alınır. Bu türdeki en bilinen ağ *Counterpropagation* ağıdır (Hecht-Nielsen, 1988).

Uygulamaya bağlı olarak, önce vektör nicemlemesi gerçekleştirilip daha sonra fonksiyon tahminleme öğretilebilir, ya da ağın ikisinin de eş zamanlı öğrenmesi sağlanabilir.

## 3.9 Uyarlamalı Rezonans Kuramı (Adaptive Resonance Theory)

Geriye yayılma algoritması düşünülecek olunursa, her türlü sürekli fonksiyonu öğrenebilmesi açısından güçlü bir yöntemdir. Ancak geriye yayılma algoritması çok zaman alan bir yöntemdir. Ağın istenilen noktaya gelebilmesi için binlerce devir gerekebilir ve evrensel (global) en düşük değere ulaşacağı da garanti edilemez. Bir kere eğitildikten sonra ağ tümüyle sabitlenir. Yeni gelen desenleri öğrenmesi için tüm eğitim işinin en baştan tekrarlanması gerekir. Bu yüzden geriye yayılma algoritması için *plastisite* veya *yoğrukluk* (plasticity) özelliği yoktur denir. Eğer çevrim içi (on-line) öğrenmeyi yeni desenler için sağlarsak bu sefer de ağ öğrendiği eski bilgileri çabuk unutmaya başlayacaktır. Bu duruma düşüren öğrenme algoritmalarına da *durağan* (stable) olmayan algoritmalar denir. Bu iki özellik arasındaki tercih etme durumuna *yoğrukluk/durağanlık çelişkisi* (plasticity/stability dilemma) denir.

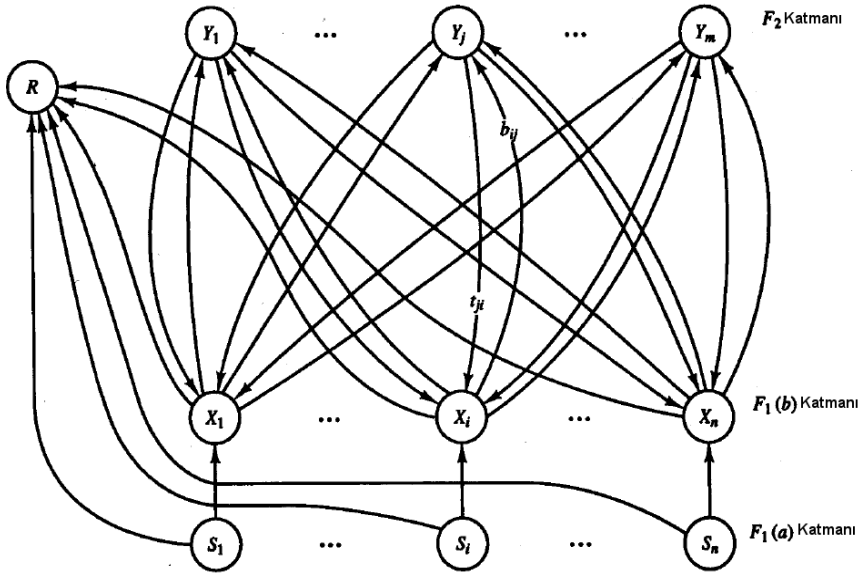
Uyarlamalı rezonans kuramı, Carpenter ve Grossberg tarafından yoğrukluk/durağanlık çelişkisini çözmek amacıyla geliştirilmiştir. İlk model olan ART1, ikili (binary) vektörleri sınıflandırmak için geliştirilmiştir, bir diğer model olan ART2 ise sürekli değerlere sahip vektörleri kabul etmektedir. Bu ağlar, girdileri eğiticisiz öğrenme ile kümelere ayırırlar. Girdi desenleri istenilen sırada sunulabilir.

ART ağları, aynı kümede yer alacak desenlerin benzerlik derecesini kontrol etmeye olanak sağlar. Ağın eğitimi sırasında her eğitim deseni birden fazla kere kullanılabilir. Bir desen ilk başta bir çıktı elemanında yer alırken daha sonra başka bir çıktı elemanında yer alabilir.

Basit bir ART yapısı 3 grup sinir hücresi barındırır; girdi işleme alanı ($F_1$ katmanı), küme elemanları ($F_2$ katmanı), ve aynı kümede yer alan desenlerin benzerlik derecelerini kontrol eden bir mekanizma.

### 3.9.1 ART1

ART1 ikili değerlere sahip vektörleri sınıflandırmak için tasarlanmıştır. Yapısında $F_1$ katmanı, $F_2$ katmanı ve benzerlik derecesini ayarlayan sıfırlama elemanı bulunur. Şekil 3.18'de ART1 ağının yapısı ve bağlantıları gösterilmiştir.




Şekil 3.18 ART1 yapısı

ART1 Öğrenme Algoritması :

$n$ *girdi vektöründeki eleman sayısı*

$m$ *oluşturulabilecek en fazla küme sayısı*

$b_{ij}$ $F_1(b)'$*deki* $X_i$ *elemanlarından* $F_2'$*deki* $Y_j$ *elemanlarına olan bağlantıların ağırlıkları*

$t_{ji}$ $F_2'$*deki* $Y_j$ *elemanlarından* $F_1(b)'$*deki* $X_i$ *elemanlarına olan bağlantıların ağırlıkları*

$\rho$ *kontrol parametresi*

$\boldsymbol{s}$ *ikili girdiler vektörü* $(n$ *elemanlı* $)$

$\boldsymbol{x}$ $F_1(b)$ *katmanı için çıktı* $(aktivasyon)$ *vektörü*

$\|\boldsymbol{x}\|$ $\boldsymbol{x}$ *vektörünün normu ,* $x_i$ *değerleri toplamı*

Adım 0: Parametrelerin başlangıç değerlerini ata:

$$L > 1,$$
$$0 < \rho \leq 1$$

Ağırlık başlangıç değerlerini ata:

$$0 < b_{ij} < \frac{L}{L-1+n},$$
$$t_{ji} = 1$$

Adım 1: Durma koşulu sağlanana kadar döngüde kal.

Adım 2: Her eğitim girdisi için.

Adım 3: $F_2$ elemanlarının çıktı değerlerini sıfıra eşitle. $F_1(a)$ Elemanlarını girdi vektörü **s** ' e eşitle.

Adım 4: **s**' in normunu hesapla:

$$\|s\| = \sum_i s_i$$

Adım 5: $F_1(a)$ 'den $F_1(b)$ katmanına girdi sinyalini gönder:

$$x_i = s_i$$

Adım 6: Her $F_2$ elemanı için :

$$Eğer\ y_j \neq -1\ ise,$$
$$y_j = \sum_i b_{ij}\, xi$$

Adım 7: Eğer sıfırlama doğruysa 8-11 adımları uygula

Adım 8: Tüm j elemanları için $y_J \geq y_j$ koşulunu sağlayan ***J*** elemanını bul. Eğer $y_J = -1$ ise bu desen sınıflandırılamaz demektir.

Adım 9: $F_1(b)$ 'nin çıktısını ( $\boldsymbol{x}$ )

hesapla:

$$x_i = s_i t_{Ji}$$

Adım 10: $\boldsymbol{x}$ 'in normunu bul:

$$\|x\| = \sum_i x_i$$

Adım 11: Sıfırlama durumunu kontrol et :

$$Eğer \frac{\|x\|}{\|s\|} < \rho \quad ise$$
$$y_J = -1 \; ve \; Adım \, 7' den \, devam \, et$$

$$Eğer \frac{\|x\|}{\|s\|} \geq \rho \quad ise \; Adım \, 12' den \; devam \, et$$

Adım 12: $\boldsymbol{J}$ elemanı için ağırlıkları güncelle:

$$b_{iJ}(yeni) = \frac{L x_i}{L - 1 + \|\boldsymbol{x}\|}$$
$$t_{Ji}(yeni) = x_i$$

# 4. İLGİLİ ÇALIŞMALAR

Yapay sinir ağları için çeşitli programlama dillerinde bir çok kütüphane ve araç geliştirilmiştir. Bu bölümde, yaygın olarak kullanılan ve genel kabul görmüş olan; SNNS, FANN, JOONE ve Matlab Neural Network Toolbox incelenmiştir.

## 4.1 SNNS

Stuttgart Neural Network Simulator (SNNS) yapay sinir ağları için Stuttgart Üniversitesi'nde IPVR enstitüsü tarafından geliştirilmiştir bir simülatördür. Sinir ağı uygulamaları ve araştırmaları için etkin ve esnek bir simülasyon ortamı oluşturmak amacıyla yapılmıştır.

SNNS 4 ana bileşenden oluşur:

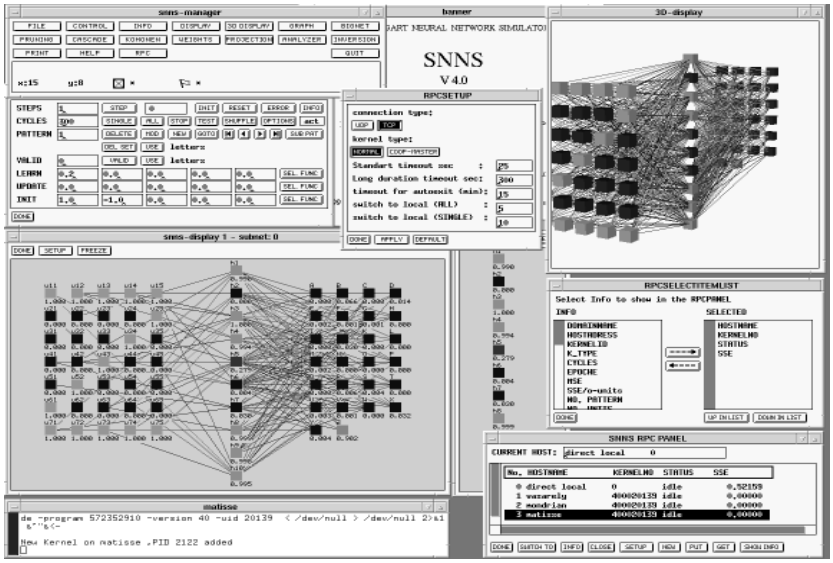
1. Simülasyon çekirdeği
2. Grafik arabirim
3. Toplu çalıştırma arabirimi
4. Ağ derleyicisi: snns2c

Simülasyon çekirdeği, sinir ağları yapısı üzerinde çalışır ve ilgili tüm işleri gerçekleştirir. Ağ derleyicisi *snn2c* yardımıyla eğitilmiş ağlar C koduna çevrilerek programlarda gömülü çalışması sağlanır. Kullanıcı tanımlı olarak aktivasyon fonksiyonlarının, çıktı fonksiyonlarının ve öğrenme algoritmalarının basit C programları olarak yazılıp simülatör çekirdeğine bağlanmaları mümkündür. SNNS'nin desteklediği sinir ağ mimarileri ve öğrenme algoritmaları şunlardır :

- Çevrim içi (on-line) , Momentumlu ve Yığın Geriye Yayılma
- Counterpropagation, Quickprop, Backpercolation 1, RProp
- RBF, ART1, ART2, ARTMAP
- Cascade Correlation, Recurrent Cascade Correlation
- Dinamik LVQ
- Backpropagation through time, Quickprop through time
- SOM
- TDNN (time-delay networks) Geriye Yayılma ile
- Jordan ağları, Elman ağları, Associative Memory

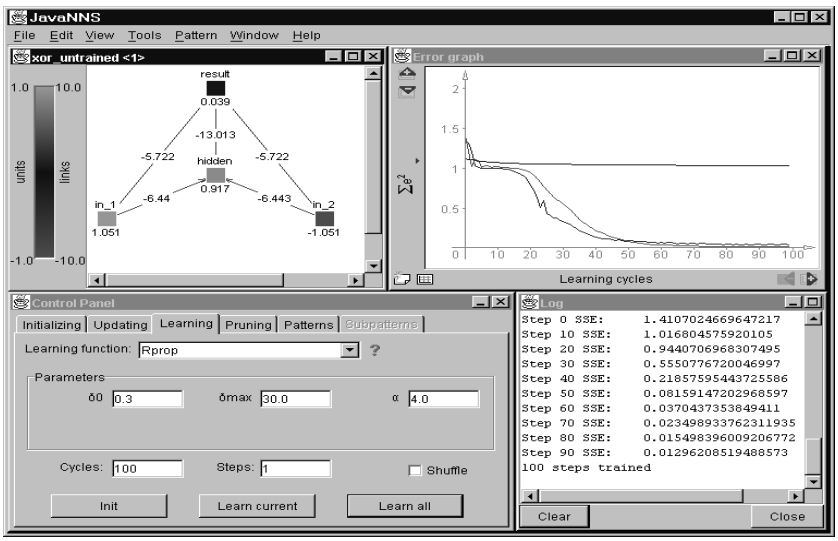
Grafik arabirim XGUI, çekirdek üzerinde çalışır. Sinir ağlarının grafiksel gösterimini ve simülasyon sırasından kontrol edilmelerini sağlar. Grafik arabirim, sinir ağlarının doğrudan yaratılmasını, değiştirilmesini ve görselleştirilmesini destekler. Karmaşık ağlar çabuk ve kolay bir şekilde yaratılabilir.

SNNS tümüyle ANSI-C 'de geliştirilmiştir. Simülasyon çekirdeği bir çok makinede ve işletim sisteminde çalışabilmektedir. Grafik arabirimi XGUI, X11 pencere sisteminde çalışmaktadır. Şekil 4.1'de XGUI gösterilmiştir.




Şekil 4.1 SNNS grafik arabirimi XGUI

SNNS'nin grafik arabirimi daha sonraları Java ortamı kullanılarak tekrar yazılmıştır ve JavaNNS olarak isimlendirilmiştir. JavaNNS anlaşılacağı üzere tümüyle simülatör ortamının Java'da gerçekleştirilmesi değildir. JavaNNS'de simülatör çekirdeği yeniden yazılmadan aynen kullanılmıştır. Şekil 4.2'de JavaNNS gösterilmiştir.




Şekil 4.2 JavaNNS

SNNS yapay sinir ağı simülatörü olarak gelişmiş bir araçtır. Yazılım geliştirme sürecinde kullanılması yönünden ele alınırsa, eğitilmiş ağların C koduna çevrilerek yazılımlara eklenebilmesini sağlamaktadır. Bu haliyle yapay sinir ağlarının yazılımlara eklenebilmesi mümkün olmaktadır. Ancak sadece eğitilmiş ağların eklenebilmesi yapay sinir ağının kullanıldığı yazılımın esnekliğini ortadan kaldırır. Yazılımın kullanıldığı problemle ilgili girdi verilerinin değişiklik göstermesi durumunda ağın yeniden eğitilmesi gerekliliği olur ki bu da tekrar simülatörün kullanılmasını daha sonra yazılımın güncellenmesini gerektirir.

Bir örnekle açıklanacak olunursa, basit bir karakter tanıma programı tasarlandığı düşünülürse, girdi olarak verilen bir karakterin resmini tanımlayacak sistem için eğitilecek ağ, Latin harflerinden oluşan bir eğitim veri kümesiyle eğitilmiş olsun. Eğer bu yazılım kullanıcı tarafından farklı bir alfabenin karakterleri için veya el yazısı için kullanılmak istenirse bu durumda SNNS içinden tekrar ağın eğitilmesi ve yazılıma eklenmesi gerekir. Bilinen alfabeler için bu yapay sinir ağı genişletilip eğitilebilir. Ancak, özellikle el yazısı tanıma yeteneğine sahip bir yazılım isteniyorsa mutlaka kullanıcı tarafında eğitilebilir bir yazılıma ihtiyaç olunacaktır. SNNS böyle bir durumda yetersiz kalır. Öğrenebilen yazılımlar geliştirmek için yazılım içerisinde tümüyle çalışır bir yapay sinir ağı kütüphanesinin kullanılması gerekliliği vardır.

## 4.2 FANN

FANN (Fast Artificial Neural Network Library), çok katmanlı algılayıcıları kullanmayı sağlayan, C programlama dilinde yazılmış bir yapay sinir ağı kütüphanesidir. Ayrıca bir çok programlama dilinden,

C++, Perl, Python gibi, erişilebilmesini sağlayan bağlayıcı arayüzleri vardır.

Kütüphanenin desteklediği öğrenme algoritmaları :

- Geriye yayılma algoritması
- Yığın öğrenmeli geriye yayılma algoritması
- RPROP
- Quickprop
- Arttırmalı geriye yayılma algoritması

Yapay sinir ağlarının kullanılması ile ilgili problemlerden biri uygulamadaki performans sorunları olmuştur. Bu açıdan FANN, sinir ağlarının hızlı çalışmasını sağlamak hedefini göz önünde tutarak geliştirilmiştir. Desteklediği sinir ağı modelleri ve öğrenme algoritmaları açısından sadece ileri beslemeli ağlar ile geriye yayılma algoritmalarını desteklemesi, ancak kısıtlı sayıda problem türü için kullanabilmesini sağlar.

## 4.3 JOONE

JOONE (Java Object Oriented Neural Engine), Java ortamında geliştirilmiş bir yapay sinir ağı kütüphanesidir. Modüler bir yapıya sahiptir. Çekirdek kısmı görsel arabirimden ayrı geliştirilmiştir. Kod tasarımı olarak yeni öğrenme algoritmaları veya yeni sinir ağı modelleri eklenebilecek bir yapıya sahiptir. Genetik algoritmalar kullanılarak sinir ağı iyileştirilmesi mümkündür. Joone çekirdek kısmı, çoklu kullanım (multi-threaded) özelliğine sahiptir. Ayrıca sinir ağının eğitimi için dağıtık olarak çalışabilecek bir ortam da sunulur.

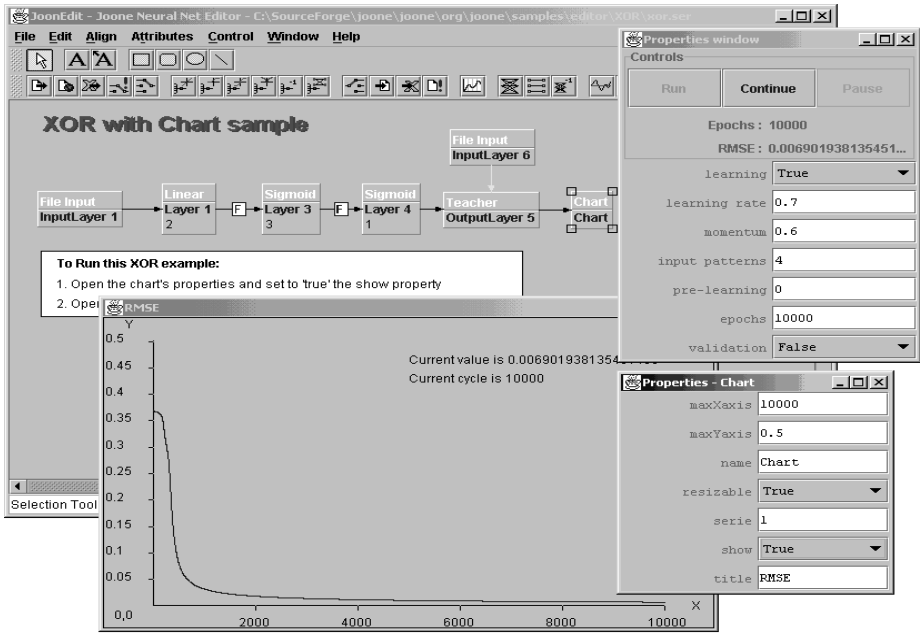
Çekirdek motoru (core engine) olarak tanımlanan kısmın desteklediği özellikler şunlardır :

- Eğiticili öğrenme algoritmaları:
    - Geriye yayılma
    - Yığın geriye yayılma
    - RPROP
- Kohonen SOM ve Principal Component Analysis
- Sinir ağlarının serialization mantığıyla dosyada saklanabilmesi.
- Veri okumak için girdi/çıktı bileşenleri:
    - Virgül ayracı kullanılmış ASCII basit metin dosyaları
    - Excel dosyaları
    - Veritabanı bağlantısı
- Ağın öğrenmesini kontrol eden ve ağla ilgili parametreleri değiştirmeye yarayan bileşenler.
- Girdi verisini ön işlemden geçirmeye yarayan eklentiler.

JOONE çekirdeğini kullanan grafik tabanlı bir düzenleyici bulunmaktadır. Düzenleyicinin genel görüntüsü Şekil 4.3'de verilmiştir. Bu araçla sinir ağlarının yaratılması, eğitilmesi ve görselleştirilmesi mümkündür. Bu aracın özellikleri şu şekildedir :

- Çekirdeğin desteklediği her tür modeli yaratabilme
- Sinir ağının dosyaya kaydedilmesi ve dosyadan yüklenmesi.
- Sinir ağ elemanlarının parametrelerinin değiştirilebilmesi
- Sinir ağının doğruluğunu kontrol etme
- Ön işlem ve kontrol eklentilerinin kullanılabilmesi

- Makro editörü
- Sinir ağı eğitme
- Sinir ağı çıktı değerlerinin görselleştirilmesi



Şekil 4.3 JOONE grafik tabanlı düzenleyicisi

JOONE kütüphanesini kullanarak, XOR problemini çözen örnek kod şu şekilde oluşturulabilir:

Öncelikle XOR problemi için girdi dosyası oluşturulması gerekir.

```
0;0;0
0;1;1
1;0;1
1;1;0
```

Her sütun birbirinden noktalı virgülle ayrılmış olmalıdır. Problemin çözümü için 3 katmanlı bir sinir ağı oluşturmak gerekir.

1. XOR için sadece 2 girdi değeri olduğundan girdi katmanı 2 elemanlı olmalıdır,
2. Gizli katmana 3 eleman eklenir.
3. XOR'un tek çıktı değeri olduğundan, çıktı katmanı 1 elemanlı olmalıdır.

Bu katmanlardan gizli katman ve çıktı katmanı için aktivasyon fonksiyonu olarak sigmoid fonksiyonu atanır. Bu 3 katmanı oluşturmak için gerekli kod şu şekilde olur:

```
LinearLayer girdi = new LinearLayer();
SigmoidLayer gizli = new SigmoidLayer();
SigmoidLayer cikti = new SigmoidLayer();
```

Sahip oldukları eleman sayıları atanır :

```
girdi.setRows(2);
gizli.setRows(3);
cikti.setRows(1);
```

Katmanları birbirlerine bağlamak için iki tane bağlantı nesnesi yaratmak gerekir. Burada *FullSynapse* sınıfından bağlantı kullanılmıştır. Bu tür bağlantıda bir katmandaki her hücre bağlandığı katmandaki tüm hücrelerle bağlantı oluşturmuş olur.

```
FullSynapse baglantiGirdiGizli = new FullSynapse();
FullSynapse baglantiGizliCikti = new FullSynapse();
```

Önce girdi katmanı ile gizli katman bağlanır:

```
girdi.addOutputSynapse(baglantiGirdiGizli);
gizli.addInputSynapse(baglantiGirdiGizli);
```

Daha sonra gizli katman ile çıktı katmanı bağlanır:

```
gizli.addOutputSynapse(baglantiGizliCikti);
cikti.addInputSynapse(baglantiGizliCikti);
```

Bu işlemlerden sonra ağ için gerekli ayarlamaları yapmayı sağlayan *Monitor* sınıfından bir nesne oluşturulur ve gerekli parametreleri atanır:

```
Monitor monitor = new Monitor();
monitor.setLearningRate(0.8);
monitor.setMomentum(0.3);
```

Bu *Monitor* sınıfından yaratılan nesne tüm katmanlara atanır:

```
girdi.setMonitor(monitor);
gizli.setMonitor(monitor);
cikti.setMonitor(monitor);
```

Uygulamanın, ağla ilgili durum bilgilerini alabilmesi için *org.joone.engine.NeuralNetListener* arayüzünü (interface) gerçekleştirmesi gerekir. Bundan sonra uygulamanın kendini *monitor* nesnesine kayıtlaması gerekir :

```
monitor.addNeuralNetListener(this);
```

Girdi dosyasını ağa gönderebilmek için *org.joone.io.FileInputStream* sınıfından bir nesne yaratıp girdi dosyası ile ilgili parametreler düzenlenmelidir:

```
FileInputSynapse girdiDosyasiBaglantisi = new FileInputSynapse();
girdiDosyasiBaglantisi.setAdvancedColumnSelector("1,2");
girdiDosyasiBaglantisi.setFileName("XOR.txt");
```

*setAdvancedColumnSelector()* metodu ile girdi verilerinin hangi sütun aralığında olduğu belirtilir.

*girdiDosyasiBaglantisi* nesnesi ilk katman olarak yarattığımız girdi katmanına bağlanır:

```
girdi.addInputSynapse(girdiDosyasiBaglantisi);
```

Sinir ağının öğrenebilmesini sağlamak için *org.joone.engine.learning.TeachingSynapse* sınıfından bir nesne yaratılması gerekir. Yaratılan *monitor* nesnesi öğretici nesneye atanır.

Eğitim verisinin ve beklenen değerlerin neler olduğu bu öğretme işini yapacak olan nesnede, girdi katmanındakine benzer bir şekilde tanımlanır.

```
TeachingSynapse egitici = new TeachingSynapse();
egitici.setMonitor(monitor);

FileInputSynapse ornekVeriler = new FileInputSynapse();
ornekVeriler.setFileName("XOR.txt");
ornekVeriler.setAdvancedColumnSelector("3");

egitici.setDesired(ornekVeriler);
```

Oluşturduğumuz *TeacherSynapse* nesnesi aynı zamanda ağdaki son katmanın çıktısı olarak kullanılır.

```
cikti.addOutputSynapse(egitici);
```

Tüm katmanlar *start()* metoduyla çalıştırılır. Katmanların türediği sınıflar *java.lang.Runnable* arayüzünü gerçekleştirdikleri için hepsi ayrı iş parçacığı (thread) olarak çalışır.

```
girdi.start();
gizli.start();
cikti.start();
```

Son olarak eğitimle ilgili tüm parametreler atanır ve eğitim başlatılır.

```
// Girdi verisi sayısı
monitor.setTrainingPatterns(4);
// Eğitim verisi kaç defa ağa öğretilecek
monitor.setTotCicles(2000);
// Ağ öğrenme durumunda
monitor.setLearning(true);
// Ağ öğrenmesi başlatılıyor
monitor.Go();
```

JOONE kütüphanesi öncelikli olarak Java ortamını kullanarak bu ortamın getirdiği avantajlara sahiptir. Platformdan bağımsız olarak çalışabilme ve dağıtık olarak öğrenmenin gerçekleştirilebilmesi göze

çarpan özellikleridir. Belli başlı ağ yapılarını ve öğrenme algoritmalarını destekleyerek sinir ağlarının uygulamalara eklenebilmesini sağlamaktadır.

Kütüphane kullanım açısından değerlendirilecek olunursa, örnek kodda da görüleceği üzere basit bir sinir ağını kurmak ve bunu eğitmek için gerekli işlemler kodun karmaşık olmasına neden olmuştur. Bu durumun nedenlerine bakmak gerekirse, örnek kod üzerinden sırayla takip ederek şu tespitler yapılabilir. Öncelikle katmanların birbirlerine bağlanması işlemi her katman ikilileri için kullanıcı tarafından tek tek yapması kodun uzamasına neden olur. Eğitim verisinin hem girdi katmanı için, daha sonra da öğretme işini gerçekleştiren nesne için tanımlanması gereksiz yere kod yazımına neden olmaktadır. Girdi verilerinin tek bir seferde tanımlanması ve gerekli olan yerde bu tanımlanmış halinin tekrar tanımlamaya gerek olmadan kullanılmasını sağlamak daha iyi bir çözüm olabilir.

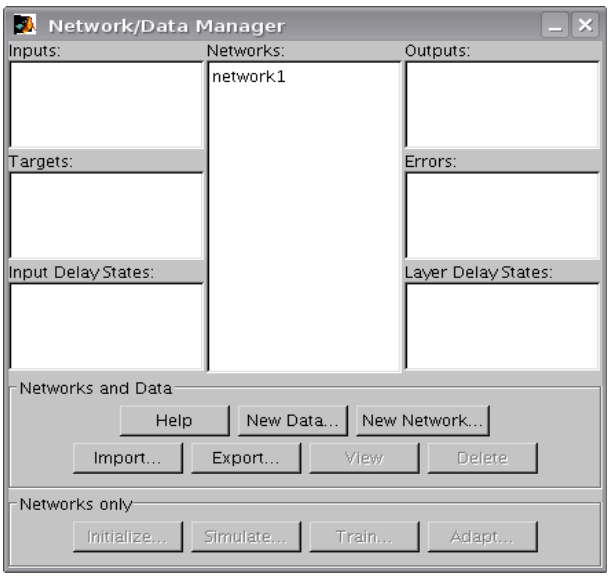
Kütüphanenin tasarımı ile ilgili olarak birkaç göze çarpan noktaya da değinmek gerekir. Katmanların aktivasyon fonksiyonlarına göre tanımlanmaları, kullanıcı tarafından yeni bir aktivasyon fonksiyonu kullanılmak istendiğinde o fonksiyonu barındıran yeni bir katman sınıfı türetilmesini gerektirir. Katman ve aktivasyon fonksiyonlarının birbirinden ayrı olması ve katmana kullanılmak istenen aktivasyon fonksiyonunun atanması daha iyi bir yöntem olabilir.

Genel olarak kütüphanenin kullanımına bakacak olursak yapay sinir ağlarının teorik olarak literatürde tanımlanan yapısıyla kodda oluşturulan sinir ağı yapısı birebir örtüşmemektedir. Örneğin; öğrenme algoritmasının çıktı katmanına bağlanması. Bu durum teorik bilgilerden kütüphanenin kullanımına geçilmesinde uyum sorununa yol açabilir, bu da kütüphanenin kolay kullanılabilirlik özelliğini olumsuz etkileyebilir.

## 4.4 Matlab Neural Network Toolbox

Matlab, bir sayısal hesaplama ortamı ve bir programlama dilidir. Matris işlemleri, fonksiyon ve veri çizimi, algoritma gerçekleştirimi gibi işlemleri gerçekleştirmeyi sağlar. Toolbox denilen paketler sayesinde görüntü işleme, kontrol tasarımı gibi birçok konuya özgü çözümler sunar.

Matlab neural network toolbox, yapay sinir ağlarının tasarlanmasını, gerçekleştirilmesini, görselleştirilmesini ve simülasyonunu sağlayan, Matlab için geliştirilmiş bir araçtır. Genel kabul görmüş birçok sinir ağı modeli ve algoritması için ayrıntılı bir ortam sunar. Grafik kullanıcı arayüzü (Şekil 4.4) ile sinir ağlarının tasarlanmasını ve kontrol edilmesini sağlar.




Şekil 4.4 Matlab Neural Network Toolbox kullanıcı arayüzü

Eğiticili ve eğiticisiz öğrenen sinir ağ türlerini destekler. Eğiticili ağlardan ileri beslemeli (feed-forward), radyal tabanlı, dinamik ve LVQ ağlarını destekler. İleri beslemeli ağlar için geriye yayılma algoritması ve türevlerini, radyal tabanlı ağlardan genelleştirilmiş regresyon ve

olasılıksal sinir ağlarını (probabilistic neural network), dinamik ağlardan Elman, Hopfield, NARX, focused time-delay ve distributed time-delay ağlarını destekler. Eğiticisiz ağlardan rekabetçi ve SOM türünde ağları destekler. Girdilerin ön, çıktıların son işlemden geçmesine imkan verir.

Matlab, teknik programlama için kullanılan bir ortam olması nedeniyle yazılım geliştirme ortamı olarak değerlendirmek güçtür. Programlama dili olarak ise günümüz yazılım geliştirme dillerine kıyasla yetenekleri oldukça sınırlı bir dildir. Yapay sinir ağlarını desteklemesine rağmen bu aracın yazılım geliştirme sürecinde kullanılması zordur. Ayrıca ücretli bir yazılım olması dolayısıyla son kullanıcıya yönelik program geliştirmek maliyetli bir iştir.

# 5. GELİŞTİRİLEN YAZILIMIN TASARIMI VE GERÇEKLEŞTİRİMİ

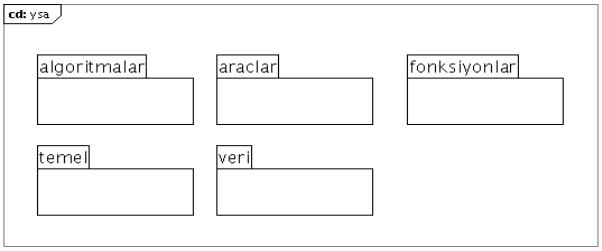
Geliştirilen yapay sinir ağı kütüphanesi nesne yönelimli programlama yaklaşımıyla tasarlanmıştır. Mümkün olduğunca basit ve kolay kullanılabilir olması hedeflenmiştir. Kütüphanenin geliştirilmesinde öncelikli hedef iyi bir tasarım oluşturmaktır. Bu nedenle çalışma hızı ve bellek kullanımıyla ilgili performans etkenleri ikinci planda değerlendirilmiştir.

Kütüphanenin desteklediği öğrenme algoritmaları :

- Adaline kuralı
- Algılayıcı öğrenmesi
- Geriye yayılma
- Yığın geriye yayılma
- LVQ
- SOM

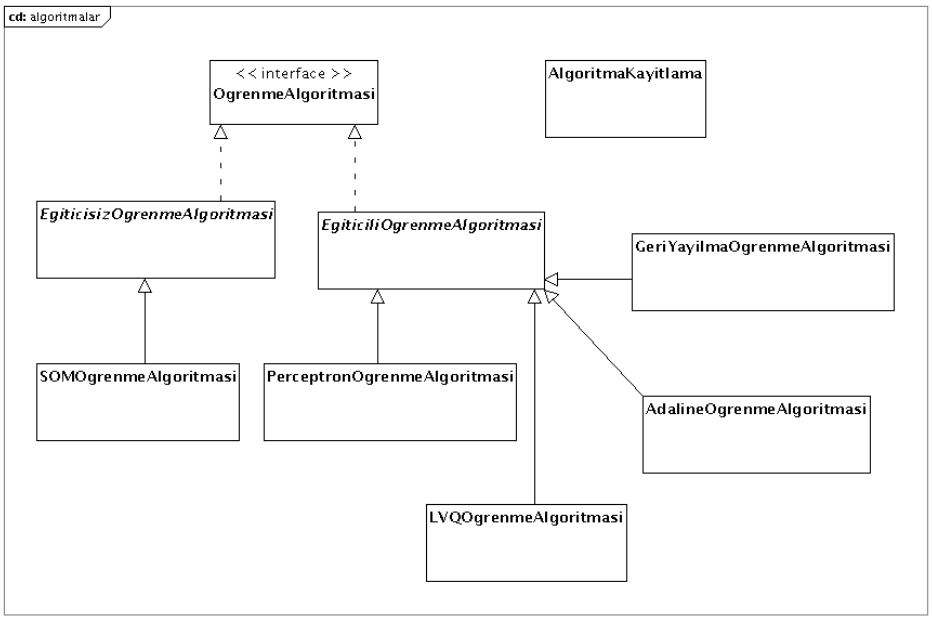
## 5.1 Kütüphaneyi Oluşturan Paketler

Kütüphane beş ana paketten oluşmaktadır; algoritmalar, araclar, fonksiyonlar, temel ve veri paketleri. Şekil 5.1'de bu paketler gösterilmiştir.




Şekil 5.1 Kütüphaneyi oluşturan paketler

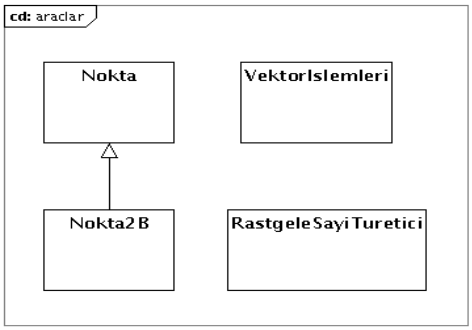
*algoritmalar* paketi, sinir ağlarını eğitmek için kullanılan öğrenme algoritmalarını içerir. Öğrenme algoritmaları *EgiticiliOgrenmeAlgoritması* ve *EgiticisizOgrenmeAlgoritması* olmak üzere iki sınıftan türerler. Şekil 5.2'de paketin genel görüntüsü görülebilir.




Şekil 5.2 algoritmalar paketi

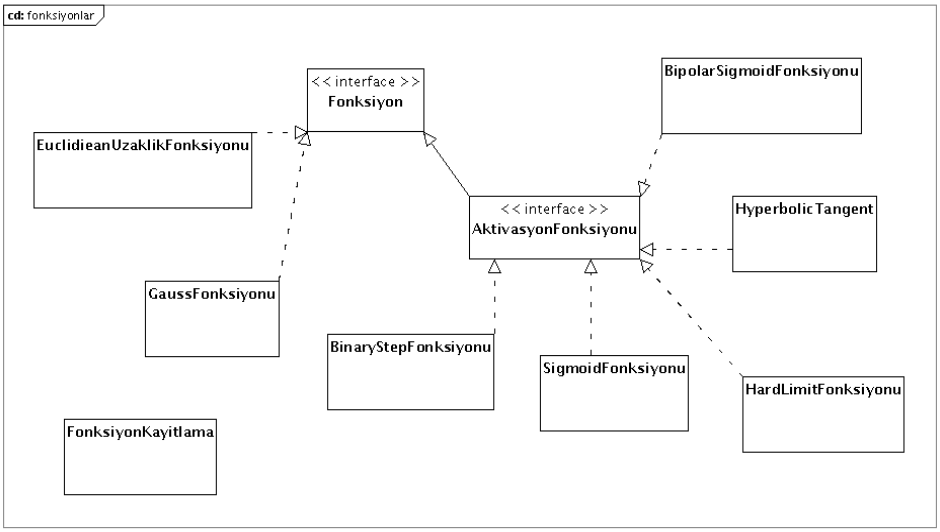
*araclar* paketi, yapay sinir ağlarıyla doğrudan ilgili olmayan, ancak temel işleri yapan sınıflardan oluşur. *Nokta2B* sınıfı, iki boyutlu noktayı

temsil eder ve SOM katmanında sinir hücrelerine konum atamak için kullanılır. Öklid uzaklığı bulunacağı zaman sinir hücrelerinin sahip oldukları Nokta2B konum nesneleri üzerinden işlem yapılır. *VektorIslemleri* sınıfı Java veri yapılarından *Vector* sınıfından nesneler üzerinde, iki vektörün toplanması çıkarılması gibi geometrik vektör işlemlerini gerçekleştirmeye yarayan metotları içerir. *RastgeleSayiTuretici* sınıfı tüm ağ bağlantılarının ağırlıklarının atanmasında istenilen aralıkta rastgele sayılar üretmekte kullanılır. Şekil 5.3'de paketin genel görüntüsü gösterilmiştir.




Şekil 5.3 araçlar paketi

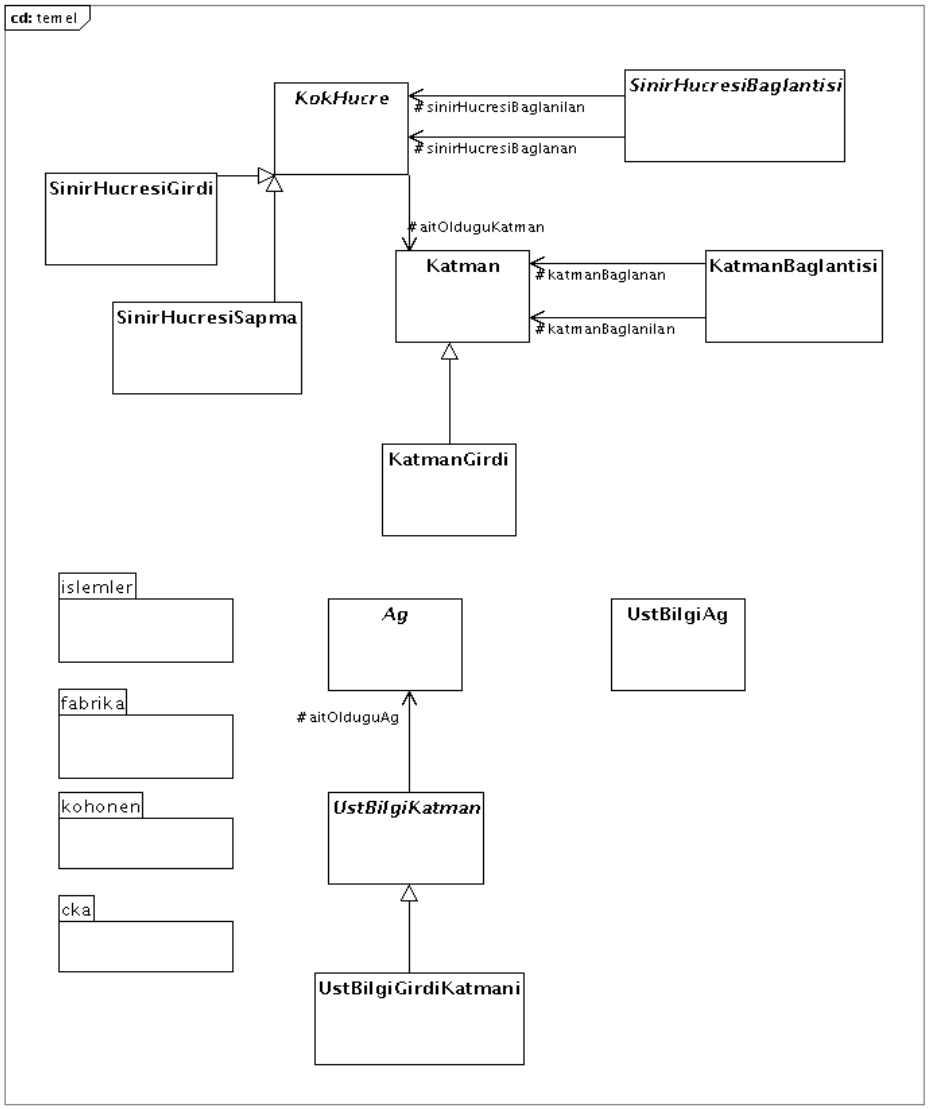
*fonksiyonlar* paketi, başta aktivasyon fonksiyonları olmak üzere sinir ağlarında kullanılan fonksiyonların tanımlandığı pakettir. *AktivasyonFonksiyonu* arayüzü tüm aktivasyon fonksiyonları için temeldir. Bu arayüzü gerçekleştiren tüm sınıfların $double\ f(double\ x)$ ve $double\ fBirinciTurev(double\ x)$ metotlarını gerçekleştirmeleri gerekir. Kullanıcı bu arayüz sınıfını kullanarak kütüphanede olmayan farklı aktivasyon fonksiyonlarını yaratıp kütüphane üzerinde kullanabilir. Paketin genel yapısı Şekil 5.4'de gösterilmiştir.




Şekil 5.4 fonksiyonlar paketi

*temel* paketi, içinde başka paketleri de barındıran ve sinir ağlarını yapısal olarak oluşturmayı sağlayan sınıflardan oluşur. Bu paket içerisinde öncelikli olarak temel sinir ağı hücreleri tanımlanmıştır. Tüm hücreler *KokHucre* sınıfından türerler. Yapay sinir ağı tanımlarına uygun olarak *Katman* sınıfı bu *KokHucre'*den türemiş her tür sinir hücresini içinde barındırır. *SinirHucresiBaglantisi* iki *KokHucre*'den türemiş hücreyi tutar ve aralarındaki bağlantıyı temsil eder. Ayrıca bu bağlantının ağırlını tutar ve ağırlık güncelleme işlemlerini sağlar. *KatmanBaglantisi* sınıfı iki katman arasındaki *SinirHucresiBaglantisi* nesnelerinden oluşan tüm sinir hücresi bağlantılarının listesini tutar. *Ag* sınıfı yapısal olarak ağa ait olan nesneler olan *Katman* ve *KatmanBaglantisi* nesnelerini tutar ve ağın çalışmasıyla ilgili metotları içerir. *UstBilgiAg* sınıfı ağ bileşenlerine ait yapısal bilgileri üst bilgi şeklinde tutmayı sağlayan temel sınıftır. *UstBilgiKatman* sınıfı ve bu sınıftan türeyen katman üst bilgisi sınıfları *UstBilgiGirdiKatmani, UstBilgiKatmanAlgilayici ve UstBilgiKatmanKohonen,* katmana ait sinir hücre sayısı, aktivasyon

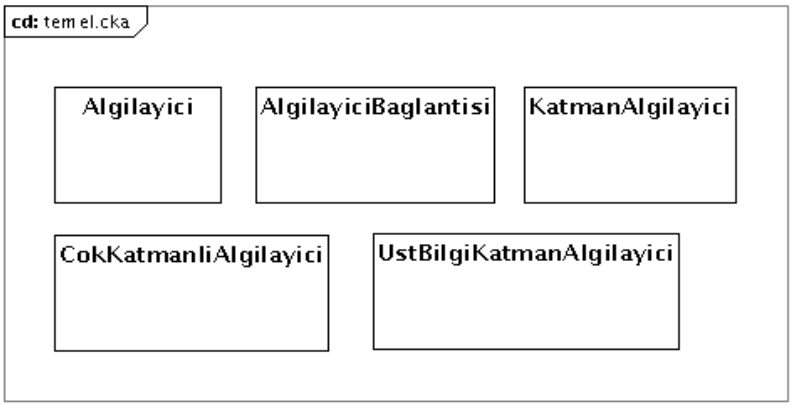
fonksiyonu türü ve sapma hücresi olup olmaması gibi bilgileri içerirler. Bu üst bilgi sınıfları *temel*,*fabrika* paketindeki sinir ağı oluşturma sınıfı *AgOlusturucu* yardımıyla ağın yapısal olarak oluşturulmasını sağlarlar. Şekil 5.5'de bu paketin barındırdığı sınıflar ve içerdiği diğer paketler gösterilmiştir.




Şekil 5.5 temel paketi

*temel.fabrika* paketi, ağa ait üst bilgileri içeren nesneden, ağın yapısal olarak yaratılmasını sağlayan sınıfları içerir. Yapısal olarak oluşturmaktan kasıt; sinir hücrelerini yaratmak ve aktivasyon fonksiyonlarını atamak, hücre ve katman bağlantılarını yapmak gibi işlemlerin gerçekleştirilmesidir. *AgOlusturucu* sınıfı kütüphanenin desteklediği ağları kendine verilen ağ üst bilgisi nesnesini kullanarak oluşturur. *Baglayici* sınıfındaki metotlar ile de katmanlar birbirlerine bağlanır. *Baglayici* sınıfı *AgOlusturucu* sınıfı tarafından kullanılır ve geliştiricinin kullanmasına gerek yoktur.

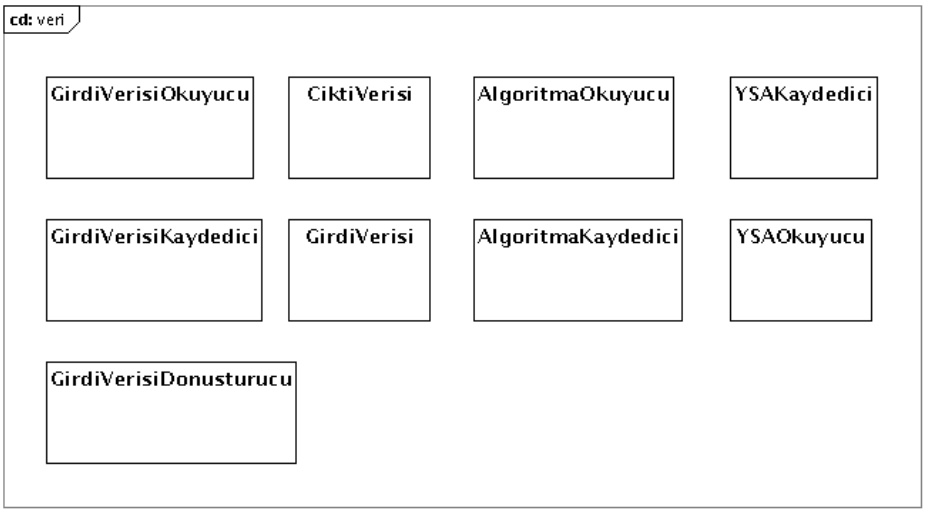
*temel.cka* ve *temel.kohonen* paketleri sırasıyla, çok katmanlı algılayıcı ve SOM-LVQ ağlarını oluşturmak için gerekli olan, *temel* paketindeki sınıflardan genişletilmiş sınıfları içerir. Şekil 5.6'da temel.cka paketi gösterilmiştir.




Şekil 5.6 temel.cka paketi

*veri* paketi; girdi, çıktı verileri ile sinir ağlarının ve öğrenme algoritmalarının, dosyada saklanması ve dosyadan okunması ile ilgili sınıfları içerir. Tüm dosyalar XML biçiminde kaydedilir. Yapay sinir ağlarının, *YSAKaydedici* sınıfı yardımıyla XML biçiminde saklanması sayesinde tüm ağın üst bilgilerini ve bağlantı ağırlık bilgilerinin dosyada

saklanması sağlanır. *YSAOkuyucu* sınıfı ile de dosyadan, sinir ağının tekrar oluşturulup program içerisinde kullanılması sağlanır. Benzer şekilde *AlgoritmaKaydedici* ve *AlgoritmaOkuyucu* sınıfları, öğrenme algoritmalarının ayarlandıkları parametrelerle birlikte saklanmasını ve tekrar kullanılmasının sağlarlar. Ayrıca *GirdiVerisiDonusturucu* sınıfı ile de metin ve Excel dosyalarından girdi verilerinin, kütüphanenin kullandığı biçime dönüştürülmesi sağlanır. Şekil 5.7'de paketin gösterimi verilmiştir.




Şekil 5.7 veri paketi

## 5.2 Kütüphanenin Kullanımına Bir Örnek: XOR

Daha önceden bahsedilen XOR problemi için tek gizli katmana sahip bir sinir ağının adım adım oluşturulması gösterilmiştir. Ağı yaratmak için ağ ile ilgili üst bilgi sınıfları kullanılır. Öncelikle girdi katmanı üst bilgisi oluşturulur. Sinir hücre sayısı ve varsa sapma (bias) hücresinin olduğu belirtilir. Hücre sayısı girdi sayısı kadar olmalıdır , yani 2 tane.

```
UstBilgiGirdiKatmani girdiUst = new UstBilgiGirdiKatmani();
girdiUst.setSinirHucresiSayisi(2);
girdiUst.setSapmaHucresiVar(true);
```

Daha sonra gizli katmanda ve çıktı katmanında kullanılacak aktivasyon fonksiyonu yaratılır. Gizli katman için üst bilgi yaratılıp, girdi katmanından farklı olarak seçilen aktivasyon fonksiyonu atanır. Daha önce de değinildiği gibi kullanıcı tarafından yeni aktivasyon fonksiyonları oluşturulup burada kullanılabilir.

```
AktivasyonFonksiyonu akFonk = new BipolarSigmoidFonksiyonu();
UstBilgiKatmanAlgilayici gizliUst = new UstBilgiKatmanAlgilayici();
gizliUst.setAktivasyonFonksiyonu(akFonk);
gizliUst.setSinirHucresiSayisi(4);
gizliUst.setSapmaHucresiVar(false);
```

Çıktı katmanı üst bilgisi de gizli katman için olduğu gibi yaratılır ve hücre sayısı, çıktı sayısı kadar yani 1 tane olacak şekilde atanır.

```
UstBilgiKatmanAlgilayici ciktiUst = new UstBilgiKatmanAlgilayici();
ciktiUst.setAktivasyonFonksiyonu(akFonk);
ciktiUst.setSinirHucresiSayisi(1);
ciktiUst.setSapmaHucresiVar(false);
```

Oluşturulan bu üst bilgiler, katmanların birbirlerine bağlanma sırasına göre ağ üst bilgisi nesnesine eklenir.

```
UstBilgiAg agUstBilgi = new UstBilgiAg();
agUstBilgi.katmanUstBilgiEkle(girdiUst);
agUstBilgi.katmanUstBilgiEkle(gizliUst);
agUstBilgi.katmanUstBilgiEkle(ciktiUst);
```

Kullanılacak öğrenme algoritması seçilir. Bu örnek için geriye yayılma algoritması kullanılmıştır. Algoritma ile ilgili öğrenme oranı, hedeflenen hata gibi parametreler belirlenir.

```
GeriYayilmaOgrenmeAlgoritmasi algo = new GeriYayilmaOgrenmeAlgoritmasi();
algo.setOgrenmeOrani(0.02);
algo.setHedeflenenMSE(0.00);
algo.setYiginOgrenmeEtkin(true);
```

Son olarak *AgOlusturucu* sınıfının statik *agOlustur()* metodu kullanılarak ağ yaratılır. Öğrenme algoritmasına eğiteceği ağ atanır. Girdi verisi ve devir sayısı belirtilerek öğrenme işlemi başlatılır.

```
geriBeslemeAgi = (CokKatmanliAlgilayici)AgOlusturucu.agOlustur(agUstBilgi);
algo.setAg(geriBeslemeAgi);
algo.baslat("XORveri.xml", 10000);
```

Öğrenme algoritmaları Java'nın *Thread* sınıflarından türedikleri için öğrenme işleminin bitip bitmediğinin takibi için en basit haliyle başka bir *Thread* nesnesi içerisinden şu şekilde kontrol edilebilir:

```
if (! algo.isAlive()) {

}
```

Öğrenmiş sinir ağını test etmek için basit bir girdi verisi oluşturup ağı bu veriyle çalıştırmak yeterlidir. Ağdaki *getSonCikti*() metodu ağın en son çalışmasında üretilen çıktı değerlerini *Vector<Double>* veri yapısı olarak döndürür. Tek çıktımız olacağı için hızlı bir şekilde ilk elemanı seçerek çıktı değerine ulaşabiliriz.

```
Vector<Double> test = new Vector<Double>();
test.add(new Double(1));
test.add(new Double(1));
geriBeslemeAgi.calistir(test);
double cikti = geriBeslemeAgi.getSonCikti().get(0).doubleValue();
```

## 5.3 Görsel Geliştirme Aracı

Günümüzde yazılım geliştirmek için kullanılan ortamlar, basit metin editörlerinin çok ötesinde araçlardır. Tümleşik geliştirme ortamı (Integrated Development Environment - IDE) olarak adlandırılan programlar, yazılım geliştirmek için gerekli olan tüm araçları bünyesinde barındırarak geliştirmenin daha kolay ve hızlı yapılabilmesine olanak sağlarlar.

Geliştirilmiş olan yapay sinir ağı kütüphanesi her ne kadar doğrudan kullanılıp yazılımlara eklenebilse de kütüphanenin kullanımını kolaylaştıracak bir görsel geliştirme aracı yazılım geliştirme sürecini hızlandıracaktır. Programcıların bir çoğunun tümleşik geliştirme ortamlarını kullandığı düşünülecek olunursa, bu ortamlar içerisinde çalışabilen bir yapay sinir ağı görsel geliştirme aracı tasarlamak uygun olacaktır. Bu amaçla geliştirilen yazılım Eclipse tümleşik geliştirme ortamı üzerinde çalışan bir araçtır. Bu aracın geliştirilmesinde Eclipse Vakfı bünyesinde geliştirilen EMF,GEF ve GMF kullanılmıştır.

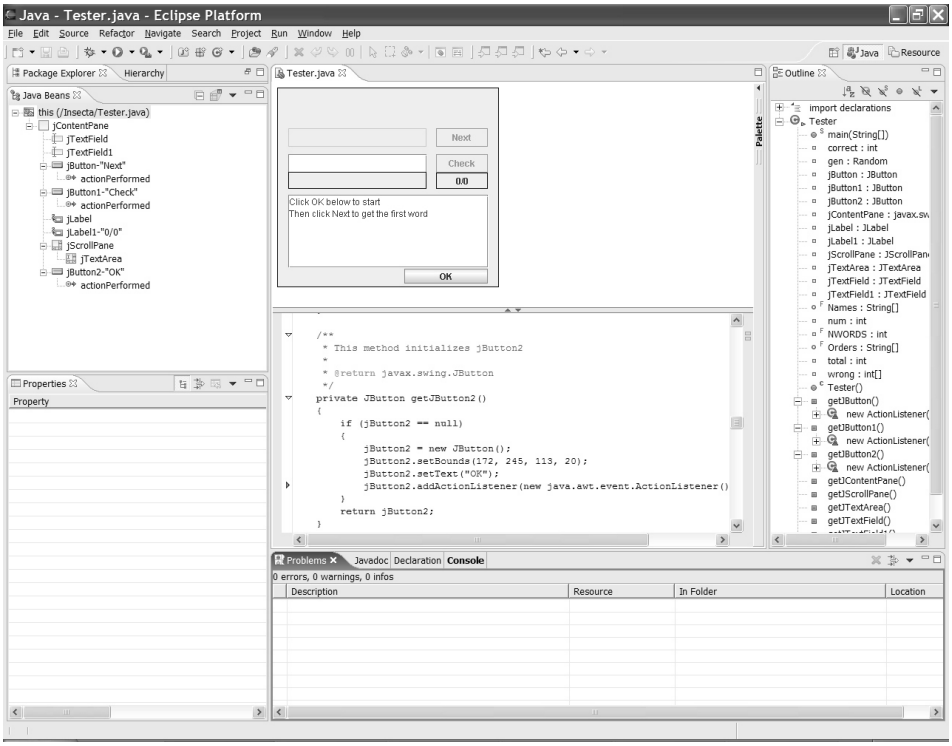
### 5.3.1 Eclipse Tümleşik Geliştirme Ortamı

Eclipse, açık kaynak kodlu bir tümleşik geliştirme ortamıdır. Ana odak noktası Java ile ilişkili teknolojiler olsa da esnek yapısı sayesinde C ve Python gibi farklı diller için de kullanılmaktadır.

2001 yılında IBM tarafından başlatılan proje, Java'nın ana grafik sistemi olan Swing yerine bulunduğu platformdaki özellikleri doğrudan kullanan SWT'yi kullanarak Java dünyasında tartışmalara yol açmıştır. Hızlı arayüzü, şık görünümü ve çok kuvvetli özellikleriyle kısa zamanda Java geliştiricileri arasında en popüler geliştirme ortamı olmuştur. 2005 yılında Eclipse projesi yönetimi Eclipse Vakfına bırakılmıştır (Wikipedia).

Eclipse oluşturduğu platformun üzerine tüm işlevselliğini eklentiler (plugin) kullanarak sağlamaktadır. Bu eklenti mekanizması sayesinde Java dışındaki bir çok programlama dillerini de destekler hale getirilebilir. C/C++, Fortran, Ruby, Python, PHP ve başka programlama dilleri için eklentiler bulunmaktadır. Eklenti mantığı sayesinde her türlü

araç Eclipse üzerinde çalışır hale getirilebilir. Şekil 5.8'de Eclipse ortamının genel görüntüsü gösterilmiştir.




Şekil 5.8 Eclipse tümleşik geliştirme ortamı

## 5.3.2 EMF (Eclipse Modeling Framework)

EMF, bir yapısal model üzerine araçlar ve uygulamalar geliştirmek için kod üretmeyi sağlayan hem bir araç hem de bir Java tabanlı iskelettir (framework). Nesne yönelimli modelleme ile oluşturulmuş modellerin süratle etkin, doğru ve kolay değiştirilebilir Java koduna çevirmeyi sağlar.

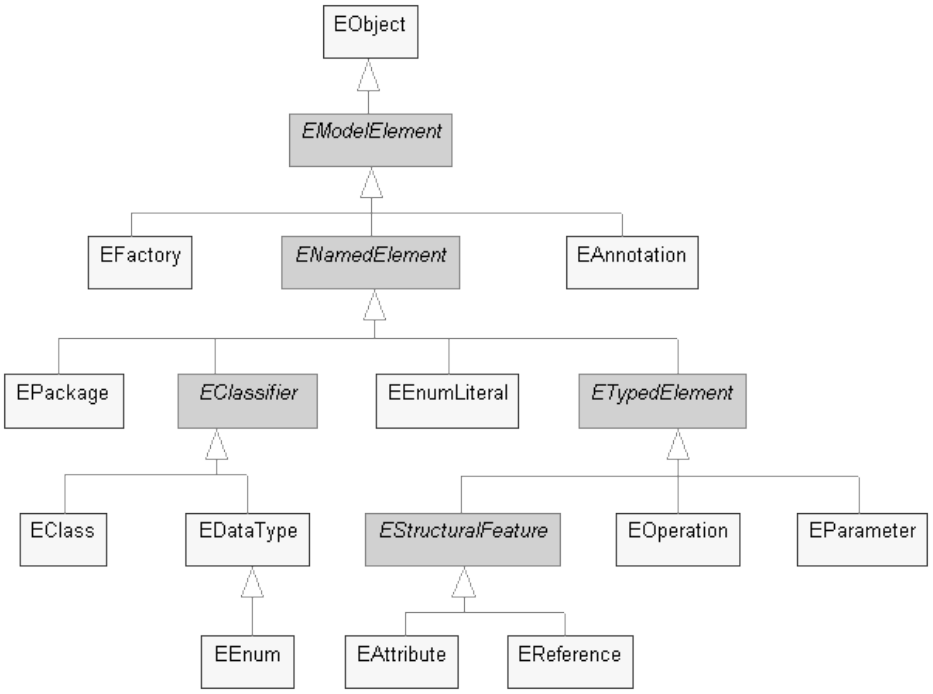
Modelleme denildiği zaman genellikle sınıf diyagramları, işbirliği (collaboration) diyagramları veya durum diyagramları akla gelmektedir.

Bu tür diyagramlar için UML (Unified Modeling Language), standart gösterimleri tanımlamıştır. UML diyagramları kullanılarak bir uygulamanın tam bir modeli tanımlanabilir. Bu türden modelleme genellikle pahalı nesne yönelimli inceleme ve tasarım araçlarını gerektirirken, EMF düşük maliyetli bir alternatiftir. Bunun nedeni ise EMF'nin UML'de kullanılan birimlerin çok küçük bir altkümesine ihtiyaç duymasıdır. Yani sadece sınıfların, sınıfların özelliklerinin ve ilişkilerinin basit tanımları kullanılır ki bu da tam ölçekli bir grafik modelleme aracına gereksinimi ortadan kaldırır.

EMF model tanımları için XMI (XML Metadata Interchange) kullanır. Bu biçimde modeller oluşturmak için şu yollardan biri izlenebilir:

- Bir XML düzenleyicisiyle doğrudan XMI belgesinin oluşturulması.
- Rational Rose gibi bir modelleme aracından XMI belgesinin alınması.
- Java arayüzlerine model özelliklerini dipnot koyarak (annotate)
- XML Schema ile modelin tanımlanması

EMF modeli yaratıldıktan sonra, ilgili Java gerçekleştirim sınıfları oluşturulabilir. Bu oluşturulan sınıflara istenilen metotlar eklenebilir. Yeniden modelden kod yaratmada, önceki kodda yapılan değişiklikler eğer modelde yapılan değişikliklere bağlı eklemeler değilse aynen korunur. EMF modelleri, Ecore üst modelinde (Şekil 5.9) bulunan elemanlar kullanılarak oluşturulur. EObject her modelin kök nesnesidir ve java.lang.Object'e denktir.




Şekil 5.9 Ecore üst modeli

Ecore veri tipleri ve bunlara karşılık gelen Java ilkel tip veya sınıfları şekil 5.10'da gösterilmiştir. Ecore özel oluşturulmuş veri tiplerini de desteklemektedir.

| Ecore Veri Tipi | Java İlkel (Primitive) Tip veya Sınıf |
|---|---|
| EBoolean | boolean |
| EChar | char |
| EFloat | float |
| EString | java.lang.String |
| EByteArray | byte[ ] |
| EBooleanObject | java.lang.Boolean |
| EFloatObject | java.lang.Float |
| EJavaObject | java.lang.Object |

Şekil 5.10 Ecore veri tipleri ve Java'daki karşılıkları

### 5.3.3 GEF (Graphical Editing Framework)

GEF mevcut uygulama modelinden zengin grafik düzenleyici oluşturmayı sağlar. *org.eclipse.draw2d* eklentisi grafiklerin gösterimi için düzen (layout) ve çevirme (rendering) aracıdır. GEF bir model görünüm denetleyicisi (MVC; model-view-controller) mimarisi kullanır. Bu sayede görünüm üzerinden yapılan değişiklikler modele de uygulanır (Eclipse Foundation). GEF temel olarak şu problemlere çözüm getirir:

1. Bir modelin grafiksel olarak gösterimi
2. Kullanıcının bu modelle etkileşimi
   - Fare ve klavye girişlerini işleme
   - Bu girişleri yorumlama
   - Modeli güncelleme
   - Yapılan değişiklikleri geri/ileri alma
3. Çalışma ortamı işlevselliği
   - İşlemler ve menüler
   - Araç çubukları
   - Kısayol tuşları

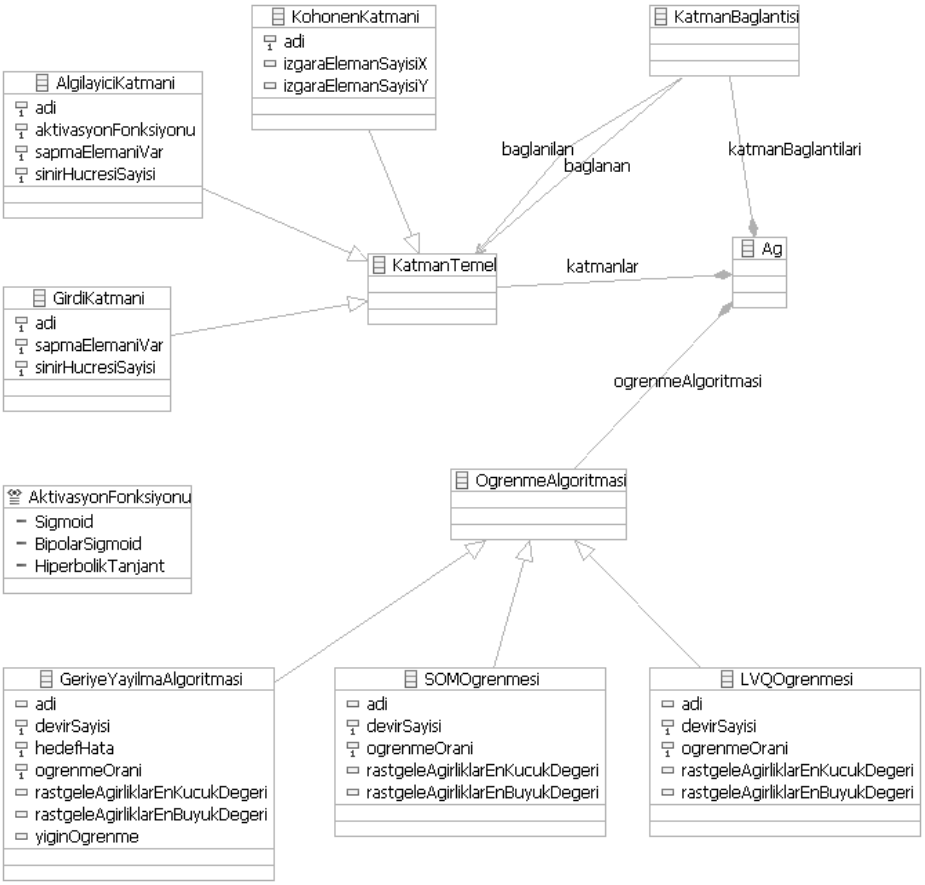
### 5.3.4 GMF (Graphical Modeling Framework)

GMF, GEF ortamı ile EMF modelleme ortamı arasında köprü kuran bir teknolojidir. Bahsedildiği gibi GEF grafik düzenleyiciler oluşturmayı sağlarken , EMF ise yüksek seviyeden verileri yönetme ve kalıcı halde tutmaya yarayan bir modelleme teknolojisidir. GMF bu ikisi

arasında bağlantı kurması sayesinde oluşturulan grafik editör EMF tarafından kontrol edilen veriyi gösterebilir.

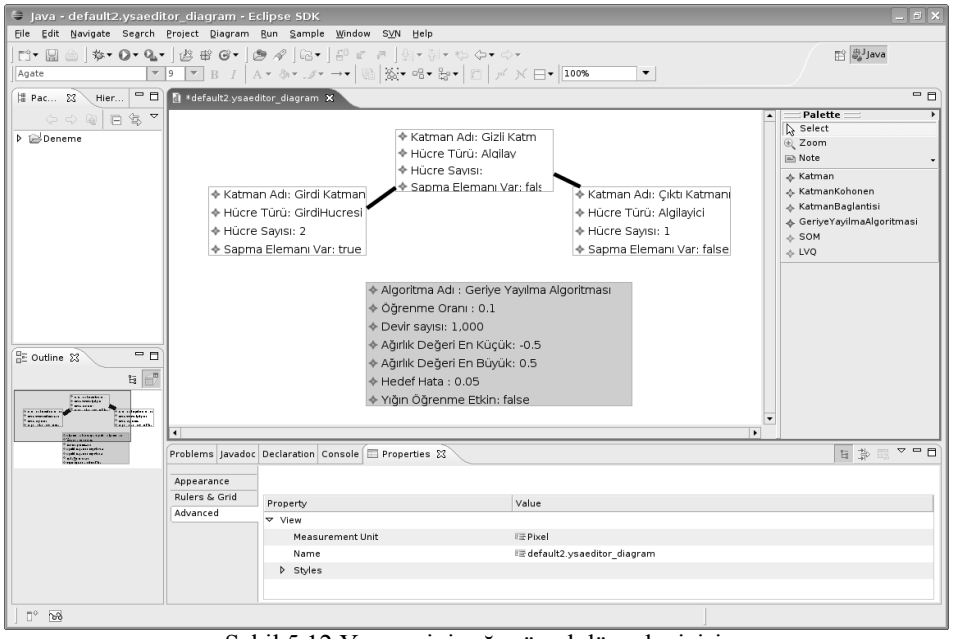
### 5.3.5 Yapay Sinir Ağı Görsel Geliştirme Ortamı

Bahsedilen teknolojiler ve araçlar yardımıyla yapay sinir ağları için bir görsel geliştirme ortamı geliştirilmiştir. Bu ortam iki kısımda hazırlanmıştır ; yapay sinir ağı görsel düzenleyicisi ve sinir ağı eğitimi kısmı.



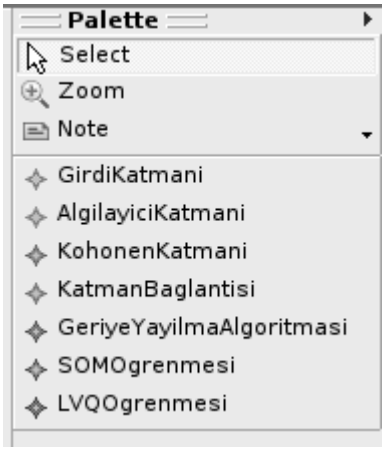

Şekil 5.11 Oluşturulan EMF modeli

Görsel düzenleme aracı için öncelikle, kullanılacak verilerin EMF yardımıyla bir modeli oluşturulmuştur. Şekil 5.11'de bu oluşturulan model gösterilmiştir. Bu model üzerinde GMF yardımıyla önce grafiksel tanımlamalar oluşturulmuştur. Bu tanımlamalar modeldeki her sınıf ve eleman için basit şekilsel gösterimlerden oluşmaktadır. Daha sonra grafiksel tanımlamaları düzenleyicide oluşturmaktan sorumlu olacak araçların tanımlanması gerçekleştirilir. Bu aşamadan sonra oluşturulmuş olan; veri modeli, grafiksel tanımlar ve araç tanımlarının birleştirilmesi gereklidir. Bu aşamada hangi grafiksel elemanın hangi araçla çizileceği ve hangi model elemanını nasıl temsil edeceği gibi eşleştirmeler yapılır. En son olarak bu oluşturulan tanımlardan kod yaratma işlemi gerçekleştirilir. Bu aşamada GMF gerekli tüm tanımlamalara uygun olarak çalışacak grafik düzenleyicisinin kodunu otomatik olarak oluşturur. Şekil 5.12'de anlatılan yöntemle gerçekleştirilen yapay sinir ağı görsel düzenleyicisi eklentisi gösterilmiştir.




Şekil 5.12 Yapay sinir ağı görsel düzenleyicisi

Aracın ekran görüntüsünde tek gizli katmana sahip bir sinir ağı oluşturulmuştur. Sağ tarafta görünen paletten (Şekil 5.13) seçilecek sinir ağı elemanları düzenleyici üzerinde sürükle bırak mantığıyla oluşturulabilir.




Şekil 5.13 Yapay sinir ağı görsel elemanlar paleti

Bu araçla, ağ katmanları ile ilgili gerekli bilgiler düzenlenebilir. Ağı eğitmek için gerekli öğrenme algoritması yaratılıp tüm parametreleri düzenlenebilir. Ağ elemanları ile ilgili düzenlemeler doğrudan şekiller üzerinden (Şekil 5.14) yapılabileceği gibi *Properties* penceresinden (Şekil 5.15) de seçili elemanla ilgili değişiklikler yapılabilir.

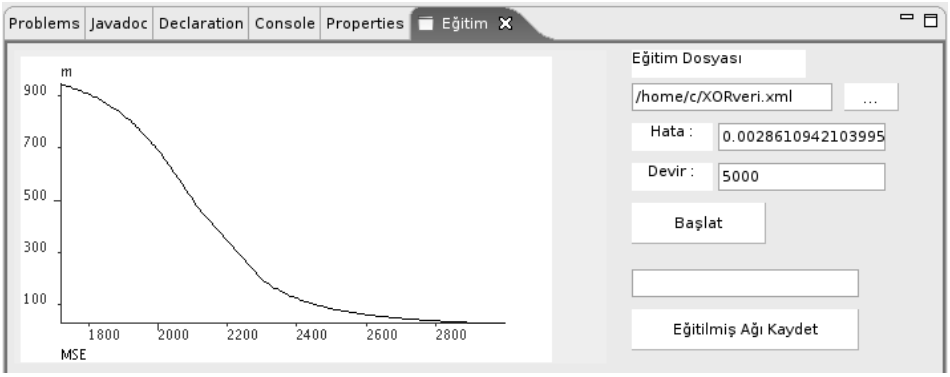

Şekil 5.14 Yapay sinir ağı görsel elemanları


Şekil 5.15 Ağ elemanlarının özellikler penceresi

Düzenleyicide oluşturulan bu ağ daha sonra kütüphane içerisindeki *YSAOkuyucu* ve *AlgoritmaOkuyucu* sınıfları sayesinde, geliştirilen programa eklenebilir. Böylece bir kaç satırla düzenleyicide oluşturulan sinir ağı ve öğrenme algoritması programa eklenmiş olur. Aşağıda bir örnek kod verilmiştir.

```
YSAOkuyucu okuyucu = new YSAOkuyucu();
Ag cokKatAg = okuyucu.dosyadanOku(dosyaAg);

AlgoritmaOkuyucu algoOkuyucu = new AlgoritmaOkuyucu();
OgrenmeAlgoritmasi algoritma = algoOkuyucu.dosyadanOku(dosyaAlgo);
```

Oluşturulan sinir ağının eğitimi için geliştirme aracının *Eğitim* pencereleri (Şekil 5.16 ve Şekil 5.17) kullanılabilir. Bu eklentiler geriye yayılma algoritması ile SOM öğrenme algoritması içindir. Ağın eğitimi için gerekli eğitim verisi seçilip *Başlat* düğmesi ile eğitim başlatılır. Geriye yayılma algoritması eğitim penceresi, eğitimle ilgili anlık hata, devir bilgileri ile anlık hata bilgisinin grafiksel olarak gösterimini içermektedir. SOM eğitim penceresi ise yine anlık devir bilgisini ve SOM katmanın u-Matris gösterimini içermektedir.




Şekil 5.16 Geriye yayılma algoritması eğitim penceresi


Şekil 5.17 SOM eğitim penceresi

SOM öğrenme algoritması için geriye yayılma algoritmasında olduğu gibi hata bilgisi kullanılmadığından ağın eğitiminin başarısını ölçmek için SOM katmanın kümeleme veya sınıflandırma işini ne kadar iyi temsil ettiğini belirlememiz gerekir. Bunun için SOM katmanın görselleştirilmesi yaklaşımıyla çeşitli yöntemler geliştirilmiştir. Bu yöntemlerden basit ve etkin bir yöntem olan u-Matris gösterimi eğitim penceresinde kullanılmıştır. U-Matris gösterimi şu şekilde olur:

$n$, *SOM katmanındaki hücre*
$NN(n)$, *n hücresine komşu hücreler*
$\boldsymbol{w}(n)$, *n hücresinin ağırlık vektörü*

$$U-uzaklık(n)=\sum_{m\in NN(n)} d(\boldsymbol{w}(n),\boldsymbol{w}(m))$$

$d(x,y)$, *SOM algoritmasında kullanılan uzaklık fonksiyonu*

Bu şekilde SOM katmanındaki her hücre için bir U-uzaklık değeri hesaplanmış olur. U-uzaklıklarından oluşan u-Matris, genellikle gri tonda resim olarak gösterilir (Ultsch, 2003).

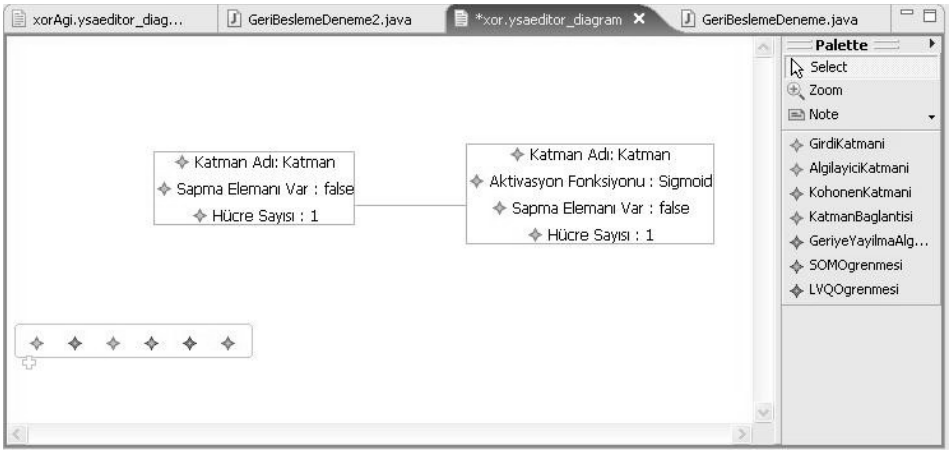
Eğitim sonucu oluşan ağ yine eğitim pencereleri üzerinden kaydedilebilir ve eğitilmiş ağ daha önce bahsedilen YSAOkuyucu sınıfı yardımıyla programlara eklenebilir.

Bu araç sayesinde bir sinir ağının görsel olarak oluşturulması, eğitilmesi ve yazılıma eklenmesi tek bir ortam içerisinden sağlanmış olur.

## 5.4 Geliştirilen Araçla Örnek Bir Uygulama Geliştirme Süreci

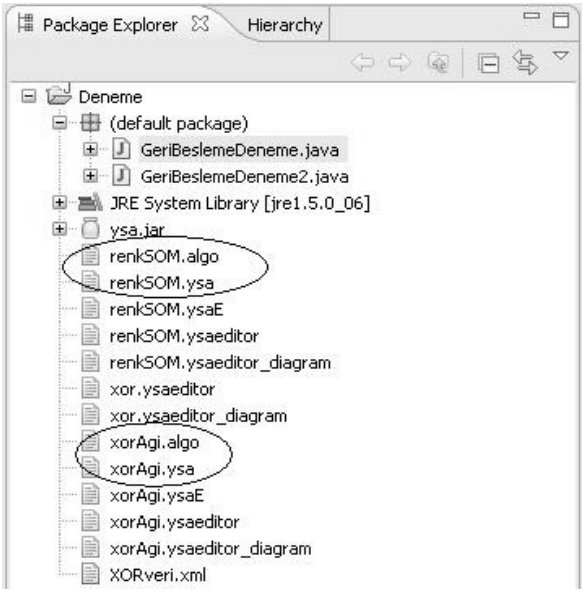
Eclipse ortamı içerisinden yapay sinir ağı görsel geliştirme aracı eklentisi ile bir ağ oluşturmak ve kod içerisinden bu oluşturulan ağı çağırmak üzere izlenmesi gereken adımlar aşağıda anlatılmıştır.

Öncelikle Eclipse ortamında New→Other içerisinden "YsaEditor Diagram" türünden yeni bir dosya yaratılır. Açılan dosya üzerine "Palette" üzerindeki sinir ağı elemanları istenilen ağ modeli oluşturulacak şekilde eklenir ve "KatmanBaglantısı" elemanı ile katmanlar istenilen sırada bağlanır (Şekil 5.18) . Sinir ağını eğitmek için istenilen öğrenme algoritması yine Palette üzerinden sayfa üzerine sürekle bırak mantığıyla eklenir. Tüm ağ elemanları üzerinde istenilen sinir hücresi sayısı, öğrenme parametreleri gibi ayarlar istenilen şekilde ayarlanır.



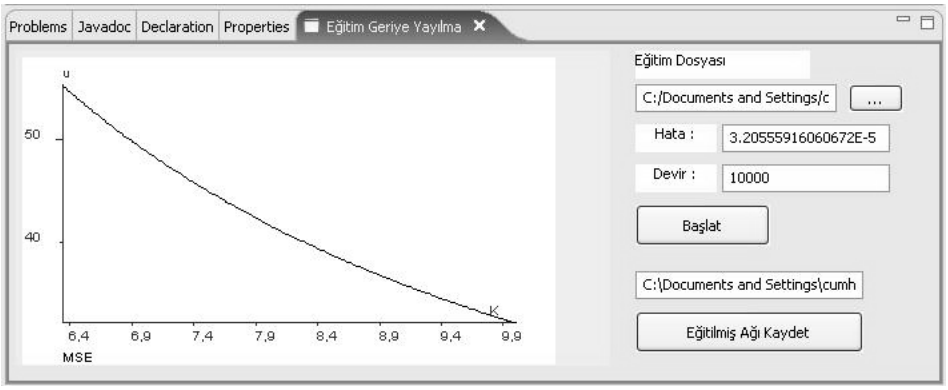

Şekil 5.18 Görsel olarak ağın oluşturulması

Ağ görsel olarak oluşturulduktan sonra YSA menüsü altında "Ağı Derle" komutuyla uzantısı ysa ve algo olan iki dosya, diyagram dosyasının bulunduğu dizinde yaratılır. YSA uzantılı dosya ağ yapısını tutan, algo uzantılı dosya ise öğrenme algoritması bilgilerini tutan XML biçiminde iki dosyadır (Şekil 5.19).




Şekil 5.19 ysa ve algo uzantılı dosyalar

Bu dosyalar yaratıldıktan sonra eğer eğitim gerçekleştirilmek istenirse Window→Show View→Other içerisinden uygun algoritma eğitme penceresi açılır. Eğitim penceresi içerisinden eğitim verisi seçilip "Başlat" komutuyla istenilen ayarlarla oluşturulan öğrenme algoritması çalıştırılır (Şekil 5.20). Eğitim sonunda bu eğitilmiş ağ istenilirse eğitim penceresi üzerinden "Eğitilmiş Ağı Kaydet" komutu ile saklanır. Eğitilmiş ağ, eğitim sonucu oluşan ağırlık bilgileri ile birlikte kaydedilir .




Şekil 5.20 Oluşturulan ağın eğitimi

Görsel olarak ağın oluşturulması ve eğitilmesi işlemlerinden sonra geliştirilen yazılımın kodu içerisinden bu ağların eklenmesi istenirse *YSAOkuyucu* ve *AlgoritmaOkuyucu* sınıfları kullanılarak bu gerçekleştirilir.

Eğitilmiş bir ağın okunması ve girdi verisi verilip çıktı alınması örneği :

```
YSAOkuyucu okuyucu = new YSAOkuyucu();
CokKatmanliAlgilayici geriBeslemeAgi;
geriBeslemeAgi = (CokKatmanliAlgilayici) okuyucu.dosyadanOku("xorAgi.ysaE");

Vector<Double> test = new Vector<Double>();
test.add(new Double(1));
test.add(new Double(1));
geriBeslemeAgi.calistir(test);
double cikti = geriBeslemeAgi.getSonCikti().get(0).doubleValue();
```

Algoritma okuma ve kod içerisinden ağ eğitimi örneği :

```
AlgoritmaOkuyucu algoOkuyucu = new AlgoritmaOkuyucu();
GeriYayilmaOgrenmeAlgoritmasi algo;
algo = (GeriYayilmaOgrenmeAlgoritmasi) algoOkuyucu.dosyadanOku("xorAgi.algo");

YSAOkuyucu ysaOkuyucu = new YSAOkuyucu();
CokKatmanliAlgilayici ag;
ag = (CokKatmanliAlgilayici) ysaOkuyucu.dosyadanOku("xorAgi.ysa");

algo.setAg(ag);
algo.baslat("XORveri.xml");
```

## 5.5 Kütüphane Özellikleri Karşılaştırması

Geliştirilen kütüphanenin, daha önce bahsedilen yapay sinir ağı kütüphaneleri ve araçları ile belli başlı özellikler açısından karşılaştırılması yapılmıştır (Şekil 5.21).

| | *JOONE* | *FANN* | *Matlab NN* | *SNNS* | *Geliştirilen Kütüphane* |
|---|---|---|---|---|---|
| **Yazılım Kütüphanesi mi?** | Evet | Evet | Hayır | Hayır | Evet |
| **Tümleşik Geliştirme Ortamı Entegrasyon Aracı** | Yok | Yok | Yok | Yok | Var |
| **Görsel Düzenleme Aracı** | Var | Yok | Yok | Var | Var |
| **Ağı Dosyada Saklama Biçimi** | Binary | Binary | Binary | Binary | XML |
| **Programlama Dili** | Java | C | Matlab | C | Java |

Şekil 5.21 Karşılaştırma tablosu

Geliştirilen kütüphaneyi farklı kılan iki özellik tabloda görülmektedir. Öncelikle geliştirilen kütüphanenin, Eclipse tümleşik

geliştirme ortamına, yazılım geliştirme sürecinde yapay sinir ağlarını eklemeyi hızlandıracak entegrasyon aracıyla birlikte dahil olması ilk ayırt edici özelliğidir. Diğer ayırt edici özelliği ise oluşturulan yapay sinir ağının dosyada saklanması için XML biçiminin kullanılmasıdır. Bu sayede kütüphanenin farklı versiyonları ile kaydedilmiş sinir ağlarının problemsiz kullanılması şansı arttırılmış olunur. Ayrıca XML biçimi kaydedilen sinir ağlarının farklı yazılımlar tarafından okunabilmesini ve hatta kullanılabilmesini de sağlar.

## 5.6 Kütüphane Performans Karşılaştırması

Her ne kadar geliştirilen kütüphane öncelikli olarak performans değerlerinin iyi olmasını hedeflemese de kabul edilebilir bir çalışma hızına sahip olması gerekir. Bu amaçla geliştirilen kütüphanenin JOONE ve Matlab ortamları ile ağ eğitimleri sürelerinin karşılaştırılması yapılmıştır.

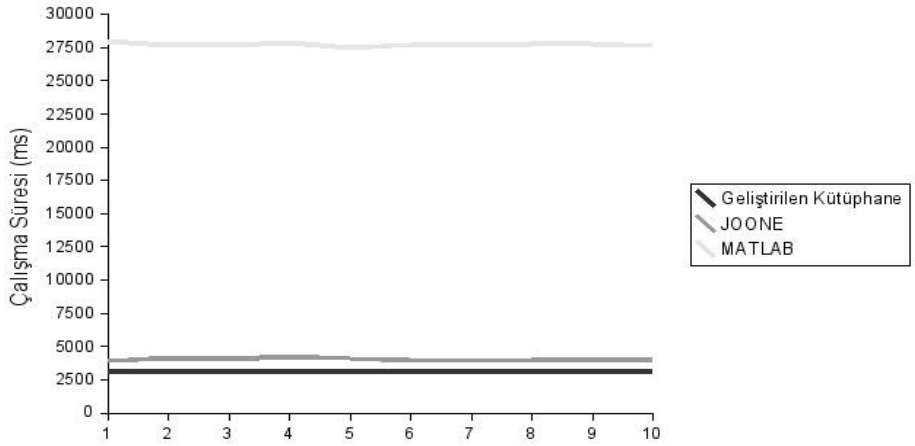
Eğitim için oluşturulan ağ tek gizli katmana sahip 3 katmanlı bir ağdır. Katmanların hücre sayıları sırasıyla 2,4,1 olarak XOR problemine uygun olacak şekilde belirlenmiştir. Test ortamı için kullanılan bilgisayarın işletim sistemi Linux, işlemcisi AMD Athlon 2500+ ve belleği 1GB'dir. Öğrenme algoritması olarak geriye yayılma algoritması kullanılmış ve 10000 devir olacak şekilde eğitim verisi için çalıştırılmıştır. JOONE için, kütüphaneyle birlikte gelen örnek kod olan *XOR_using_NeuralNet.java* dosyası temel alınarak oluşturulmuştur. Matlab için oluşturulan kod aşağıda verilmiştir.

```
P = [ [0;0] [0;1]  [1;0] [1;1]];
T = [ 0 1 1 0 ];

net = newff ( minmax(P) , [4 1],{'logsig' 'logsig'} , 'traingd' );

t = cputime
net = init(net);

net.trainParam.epochs = 10000;
net.trainparam.goal = 0.0;

net.trainParam.show=NaN;

net = train(net,P,T);

e = cputime -t
```

Tüm ağlar için eğitim 10'ar defa tekrarlanmış ve her eğitimin çalışma süreleri kaydedilmiştir. Aşağıdaki tabloda (Şekil 5.22) eğitim süreleri mili saniye cinsinden verilmiştir. Elde edilen değerlerin grafik olarak gösterimi Şekil 5.23'de verilmiştir.

| Deneme | **Geliştirilen Kütüphane** | **JOONE** | **MATLAB** |
|---|---|---|---|
| 1 | 3140 | 3914 | 27960 |
| 2 | 3062 | 4159 | 27640 |
| 3 | 3055 | 4091 | 27660 |
| 4 | 3109 | 4256 | 27840 |
| 5 | 3054 | 4097 | 27500 |
| 6 | 3094 | 3980 | 27710 |
| 7 | 3057 | 3901 | 27660 |
| 8 | 3081 | 3958 | 27770 |
| 9 | 3075 | 4041 | 27760 |
| 10 | 3111 | 3938 | 27610 |
| Ortalama | *3084* | *4034* | *27711* |

Şekil 5.22 Eğitim süreleri




Şekil 5.23 Eğitim sürelerinin grafiksel olarak gösterimi

Elde edilen değerler tüm ağ yapıları ve öğrenme algoritmaları için aynı veya benzer değerler elde edileceği anlamına gelmez. Değerlere bakıldığında geriye yayılma algoritması için geliştirilen kütüphanenin çalışma hızı açısından herhangi bir dezavantajı olmadığı söylenebilir. Kütüphane tarafından desteklenen diğer ağ modelleri ve öğrenme algoritmaları da test için kullanılanlar ile aynı temel yapıyı paylaşırlar. Bu nedenle geliştirilen kütüphanedeki tüm modeller ve algoritmalar için benzer sonuçlar alınabileceği söylenebilir. Kütüphanenin geliştirilmesi aşamasında, tasarım ve geliştirme araçlarıyla entegrasyon ön planda tutulmasına rağmen performans açısından da kullanılabilir olduğu sonucuna varılabilir.

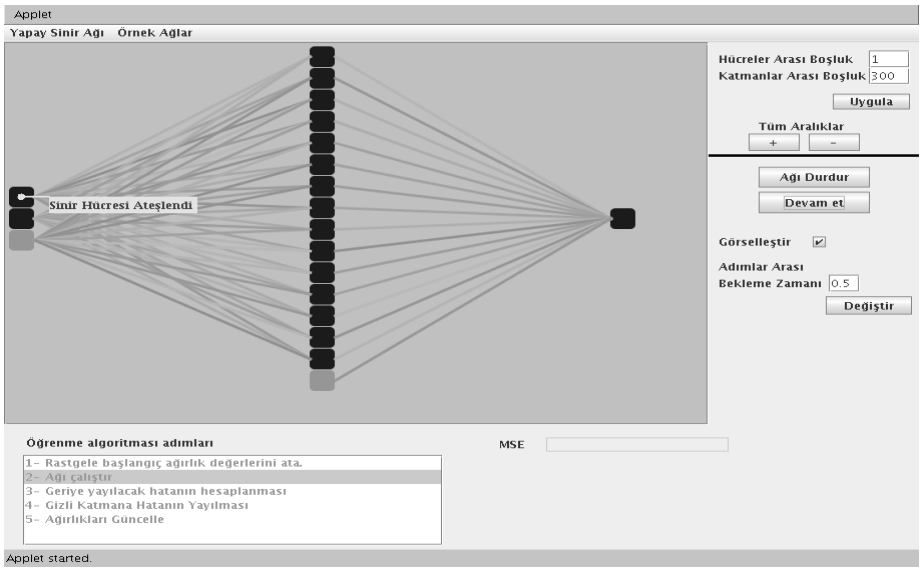
## 5.7 Yapay Sinir Ağlarını Öğrenme Aracı

Geliştirilen yapay sinir ağı kütüphanesi kullanılarak bir öğrenme aracı gerçekleştirilmiştir. Bu araç için, Java Applet teknolojisi tercih edildiğinden İnternet üzerinden kullanılabilmesi de mümkün olmaktadır.

http://yzgrafik.ege.edu.tr/ysa-applet.html adresinden programa erişilebilir.

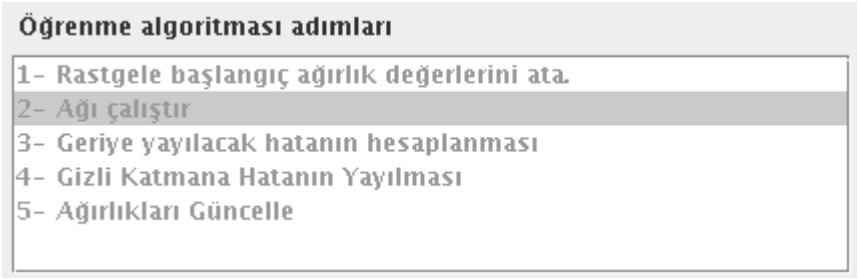
Temel olarak yapay sinir ağının yapısını, yani sahip olduğu elemanların (sinir hücreleri, bağlantılar vb.) görselleştirilmesi ile öğrenme algoritmalarının işleyişinin görselleştirilmesini sağlar. Bunun yanında aynı araçla yapay sinir ağı oluşturmak, eğitmek ve bu eğitilmiş ağı kaydetmek de mümkündür. Bu araçla yapay sinir ağlarını öğrenmek isteyenler için bir ortam oluşturulmuştur.

İleri beslemeli ağlar ile Adaline, algılayıcı ve geriye yayılma öğrenme algoritmalarını destekler. Kütüphanenin desteklediği diğer yapay sinir ağları modelleri ve öğrenme algoritmaları da eklenebilir. Bu araç, geliştirilen kütüphanenin kullanılmasına bir örnek oluşturması ve yapay sinir ağlarının yapısının ve algoritmalarının görselleştirilmeyle öğretilmesinin temel olarak nasıl olabileceğini göstermesi açısından önemlidir. Yapay sinir ağlarını ve öğrenme algoritmalarını görselleştiren araç Şekil 5.24'de gösterilmiştir.




Şekil 5.24 Yapay sinir ağı öğrenme aracı

Bu araçla, yaratılan sinir ağının çalışmasının adım adım nasıl gerçekleştiği gözlemlenebilir. Öğrenme algoritmasında hangi adımın uygulandığı (Şekil 5.25) ve sinir ağı elemanları üzerinde hangi işlemlerin ve değişikliklerin yapıldığı (Şekil 5.26) takip edilebilir. Ayrıca hücrelerin çıktı değerleri ile bağlantıların ağırlık değerleri de gözlemlenebilir (Şekil 5.27 ).




Şekil 5.25 Öğrenme algoritması adımları


Şekil 5.26 Sinir hücresi üzerinde uygulanan işlem


Şekil 5.27 Hücre ve bağlantı bilgisi pencereleri

# 6. SONUÇLAR VE TARTIŞMALAR

Bu çalışmada temel olarak yapay sinir ağları tekniğinin yazılımlarda kullanılmasını kolaylaştıracak araçların geliştirilmesi üzerinde durulmuştur. Görsel araçların genelde kullanımı kolaylaştırması ve hızlı sonuç üretmeyi sağlaması açısından yazılım geliştirme süreçlerinde de yoğunlukla kullanılması bu yönde araştırma ve geliştirme yapmaya yönlendirmiştir.

Geliştirilen kütüphane yapay sinir ağlarının topolojisine mümkün olduğunca uygun olacak şekilde hazırlanmıştır. Bu sayede teorik bilgiye sahip bir kimsenin geliştirilen kütüphaneyi kullanırken uyum süresinin kısa olması sağlanabilir.

Yapay sinir ağlarının görsel ifadesinin kolay ve anlamlı olması dolayısıyla, geliştirilen kütüphane üzerinden yapay sinir ağlarının oluşturulması için görsel düzenleme ve eğitim aracının gerçekleştirilmesi mümkün olmuştur. Bu sayede hem kütüphanenin kullanımını öğrenme gereksinimi azaltılmış olur, hem de hızlı bir şekilde görsel ortamda oluşturulan ağlar yazılımlara eklenebilir.

Eclipse geliştirme ortamının işlevsel özellikler açısından güçlü ve kolayca genişletilebilir olması, geliştirilen yapay sinir ağı kütüphanesinin bu ortama entegre bir şekilde kullanılabilmesini sağlamıştır. Yazılım geliştirme için gerekli tüm araçlar aynı ortamda ve birlikte çalışır olmalı, tümleşik geliştirme ortamı felsefesine uygun olarak geliştirilen yapay sinir ağları kütüphanesi ve araçları da tümleşik yazılım geliştirme ortamına dahil edilmiştir.

Nesne yönelimli olarak geliştirilmesi nedeniyle gerçekleştirilen kütüphane ve araçlar diğer sinir ağları modellerinin de eklenebilmesine olanak sağlar.

Kütüphanenin ve görsel aracın geliştirilmesinde öncelikle izlenen yaklaşım yazılım geliştirme sürecine entegrasyonu sağlamaktır. Bunun dışında işlem hızı, bellek kullanımı gibi performans konuları ikinci planda düşünülmüştür. Ancak elde edilen karşılaştırmalar ve veriler sonucunda, uygulama aşamasında performans kriterleri açısında kullanımı engelleyecek durumların oluşmadığı söylenebilir.

Günümüzde öğrenebilen yazılımların geliştirilmesi önem kazanırken, bu amaca hizmet edecek araçların da geliştirilmesi ve bir altyapı oluşturulması gerekir. Bu amaç doğrultusunda bakıldığında yapılan bu çalışma, zeki yazılımların geliştirilmesi için kullanılabilecek bir araç olarak değerlendirilebilir.

# KAYNAKLAR DİZİNİ

**Abdi, H.**, 1994, A Neural Network Primer, Journal of Biological Systems, vol.2, 247-283pp

**Asakawa K., Takagi H.** , 1994, Neural networks in Japan , Commun. ACM vol. 37, 106-112pp

**Eclipse Foundation**, *http://www.eclipse.org*

**Fahlman E. S.,** 1990, The recurrent cascade-correlation architecture, NIPS-3: Proceedings of the 1990 conference on Advances in neural information processing systems 3, 190-196pp.

**Fausett L.V.**, 1994 , Fundamentals of Neural Networks, Prentice-Hall.

**Haykin S.**, 1999, Neural Networks A Comprehensive Foundation, Prentice Hall

**Hebb DO.**, 1949, The Organization of Behavior, John Wiley

**Hecht-Nielsen R**. , 1988, Applications of counterpropagation networks, Neural Networks. Vol. 1, 131-139 pp.

**Hopfield JJ.,** 1984, Neurons With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons, Proc National Academy of Science ,81: 3088-3092pp.

**Klimasauskas C.** , 1989 , Neural networks: a new technolgy for information processing , SIGMIS Database vol.20, 21-23pp

**Kohonen, T.**; 1990 , Improved versions of learning vector quantization

Neural Networks, 1990., 1990 IJCNN International Joint Conference vol.1 , 545 – 550 pp.

**Kohonen T.**, 1988, An Introduction to Neural Computing, Neural Networks vol.1, pp.3-16

**Kohonen T.**, 1982, Self-organized formation of topologically correct feature maps, Biological Cybernetics, vol.43, 59-69 pp.

**Kröse B., Smagt P.,** 1996, An Introduction to Neural Networks, The University of Amsterdam

**McCulloch WS, Pitts W.,** 1943, A Logical Calculus of the Ideas Immanent in Nervous Activity. Bull Mathematical Biophysics, 115-133pp.

**Minsky M., Papert S.**, 1969 , Perceptrons: An Introduction to Computational Geometry, MIT Press

**Rosenblatt F.**, 1962, Principles of Neurodynamics, Spartan

**Russell S. J., Norvig, P.**, 1995, Artificial Intelligence: A Modern Approach, Prentice-Hall.

**Sondak N. E. , Sondak V. K.,** 1989, Neural networks and artificial intelligence, SIGCSE '89: Proceedings of the twentieth SIGCSE technical symposium on Computer science education, 241-245pp.

**Ultsch A.,** 2003 University of Marburg, Department of Computer Science, Technical Report, Nr. 36

**Widrow B., Lehr M. A.**, 1990, 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation, Proc. IEEE, 78:1415–1442.

**Wikipedia**, *http://www.wikipedia.org*

# ÖZGEÇMİŞ

**Adı Soyadı** : Ahmet Cumhur KINACI

**Doğum Tarihi** : 20.06.1980

**Doğum Yeri** : Sandıklı

**Medeni Hali** : Bekar

**Uyruğu** : T.C.

**Eğitim**

Yüksek Lisans : 2003- Ege Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Lisans : 1998-2003 Orta Doğu Teknik Üniversitesi
Fen Edebiyat Fakültesi
İstatistik Bölümü

Ortaokul-Lise : 1991-1998 Ankara Fethiye-Kemal Mumcu Anadolu Lisesi

**Araştırma Alanları** : Yapay zeka, yapay sinir ağları, yazılım mühendisliği

**Yabancı Dil** : İngilizce